GAMS – Documentation

Contents

Chapter 1

GAMS Documentation Center

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

- Preface
- Release Notes - 46.4.1 Maintenance release (April 16, 2024)
- User's Guide
  - Installation and Licensing - Guides on installing GAMS for various platforms and using a GAMS license
  - Tutorials and Examples - Step-by-step guides including a number of examples
  - GAMS Language and Environment - Guide through components of GAMS Language and the environment for executing a GAMS model
- Solver Manuals - Manuals of solvers available in the distribution
- Tools Manuals - Manuals of tools available in the distribution
- Application Programming Interfaces - Manuals of Application Programming Interfaces
- Appendix - Glossary, bibliography, and list of third-party codes

1.1 Model Libraries

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

The following model libraries are available:

- **GAMS Model Library** - includes GAMS models representing interesting and sometimes classic problems, ranged from production and shipment by firms, investment planning, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and many more.

- **GAMS Test Library** - includes GAMS models developed for testing and quality control, both for the GAMS base module and the many solvers distributed with the GAMS system.
- **GAMS Data Library** - includes GAMS models demonstrating various utilities to interface GAMS with other tools and applications such as spreadsheets and database interface.
- **GAMS EMP Library** - includes GAMS Extended Mathematical Programming (EMP) models that illustrate and test the capabilities of GAMS/EMP.
- **GAMS API Library** - includes GAMS Models used as scripts to compile and execute application programs in various programming languages interfacing to GAMS.
- **FIN Library** - includes GAMS practical financial optimization models described in the book Practical Financial Optimization: Decision Making for Financial Engineers by Consiglio, Nielsen and Zenios,
- **NOA Library** - includes GAMS nonlinear optimization applications models based on the book Nonlinear Optimization Applications Using the GAMS Technology by Neculai Andrei.
- **PSOPT Library** - includes GAMS optimization models based on the book Power System Optimization Modelling in GAMS by by Alireza Soroudi.

See Accessing Model Libraries on how to access a GAMS model from the model libraries.

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

1.2 Further Help

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

There are a number of contributed documentations that have been contributed by GAMS users as well as presentations, books, posters, and advertisements contributed by people working with GAMS.

You can also visit our upcoming courses and workshops.

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

Chapter 2

Preface

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

Figure 2.1 Covers of GAMS User's Guide and Solver Manual

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

As the pace of development increased, it became clear that we needed a new documentation scheme, one that would allow us to update the document continuously as language features, models and model libraries, solvers, and tools/utilities were added to or updated in the system. Consequently, we converted the existing documentation to MS Word, expanded, updated, and edited it as appropriate, and sent it to the copy shop to be printed. This was a group effort spearheaded by Ramesh Raman, whose name was included as an author in the User's Guide first included (along with a CD) in shipments of our 18.0 release in Feb 1999. The solver manuals in this release were collected from various sources (with varying styles, content, and organization) and put into a separate binder. A further push was required to unify this solver documentation (using LaTeX) to use a common style and organization: this unified Solver Manual was included with the 21.0 release of May 2003. In addition to hard-copy, all of these documents were included as PDF in the GAMS system and freely available via the Web. Thus, the shift away from published documentation to manuals produced in-house went hand-in-hand with a shift from printed documents to viewing and searching PDF documents.

During this time Bruce McCarl had been regularly teaching GAMS courses: basic and advanced, targeting a general audience but with a distinct flavor of agricultural economics. As part of this he developed course material to use in his classes, and he was in a good position to notice that the GAMS User Guide was not keeping up with all the updates to the software. With some encouragement from GAMS, Bruce started the ambitious project of creating a new GAMS User's Guide, one that would be more complete, more to his taste with an increase in tutorial flavor, and designed to take full advantage of being a Web document: linked, cross-referenced, searchable, and indexed. The end result was the McCarl Guide that emerged as a PDF based on a Word document in 2002. This was redone in the more convenient CHM format with the 22.1 release of Mar 2006 and renamed as the *Expanded GAMS Guide (McCarl)* in release 22.3 of Nov 2006. We also kept the original GAMS User's Guide, by this time as PDF generated from LaTeX source. With its improved content, organization, and navigability, Bruce's new guide was a boon to all GAMS users, not just those taking his courses. We owe a debt of gratitude to Bruce for this and many other contributions.

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

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

Chapter 3

Release Notes

3.1 Release Types

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

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

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

3.2 Release History

3.3 46 Distribution

3.3.1 46.1.0 Major release (February 17, 2024)

3.3.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Simon Wesley Bowen, Ruud Egging-Bratseth, Wolfgang Britz, Katja Jensen, Bruce McCarl, Scott McDonald, Evangelos Panos, Hailie Petrick, Alfonso Rodriguez Osuna, and Tom Rutherford.

3.3.1.2 Platforms

- As announced, dropped support for macOS 11 (Big Sur).

3.3.1.3 GAMS System

GAMS

- Added new put_utility commands `stdOut` and `stdErr` to send a message to standard output and error streams independent of the GAMS log file.
- Added new command line parameter gdxSymbols to select symbols that get exported when command line parameter GDX is set.
- Print warning to the log if execute_load or execute_loaddc is called without specifying any symbols to be loaded.
- Changed default for option threads to 0. The new meaning of setting `threads=0` is that the solver will decide on the number of processors to use. In particular, the following changes in behavior can be observed for `threads=0`:
  - MIP solvers called by BARON no longer use multiple processors.
  - CBC may use multiple processors in linear algebra subroutines, but not for parallelization in root node processing or the tree search.
  - COPT, CPLEX, GUROBI, KNITRO, and XPRESS now decide automatically on the number of processors to use.
  - IPOPT(H) and LINDO / LINDOGlobal use 1 processor.
  - MOSEK now uses the number of processor cores instead of the number of virtual processors to determine the number of threads (i.e., hyperthreading is disregarded).
  - SCIP now decides automatically on the number of processors to use in MILP presolving and LP solving. It uses 1 thread for some linear algebra routines.
  - MIP solvers called by SHOT now decide automatically how many processors to use.
- Added new option value 22 for dumpOpt that writes processed input with all comments into a separate dump file for each block.
- Removed misleading message about matched rows for CNS models.
- Fixed a potentially wrong result of a sameAs statement, when it is used in a loop construct with a singleton set which gets changed in the loop execution.
- Fixed a wrong dump file, when $onVerbatim was used with $onPut or EmbeddedCode.
- Fixed a potential crash with dumpOpt and dumpOptGdx.

3.3.1.4 Documentation

- The McCarl GAMS User Guide will be dropped with the next major release.

Embedded Python Code Facility

- As announced, the command line parameter `pyMultInst` has been dropped.
- Fixed a problem with empty code blocks.

GAMS Connect

- Added new option skip for the Concatenate agent that allows to indicate if sets or parameters should be skipped.
- Added new ExcelReader and ExcelWriter agents for reading and writing Excel files.
- `PandasExcelReader` and `PandasExcelWriter` are deprecated and will be removed in a future release. Use ExcelReader and ExcelWriter instead.
- Harmonized the sorting behavior of Connect agents. Please check the Usage section for details.
- Improved exception handling for GDXWriter with better error message in case of duplicate records and `duplicateRecords` set to `all`.
- Fixed a bug where GAMSWriter would not raise an error in case of duplicate records and `duplicateRecords` set to `all`.
- Fixed that GAMSWriter and GDXWriter agents were changing the Connect database when dropping duplicate records.

GMSPython

- The Python version of GMSPython has been upgraded to `3.12.1`.
- Added the package `scipy` to GMSPython.

3.3.1.5 Solvers

Antigone, CBC, Ipopt, SCIP, SHOT

- Updated MKL to version 2024.0 on Linux and version 2023.0.0 on Windows.

BARON

- New libraries 24.1.30.
  - Improved algorithms for integer linear and nonlinear programs, including local search, reformulations, relaxations, cutting planes, and presolve.

CBC

- New libraries nauty 2.8.8.

CONOPT

- New libraries 4.32.
  - Bounds larger than Lim_Variable will now be reset to inf (analogously for -inf). Previously, CONOPT terminated with an error.
  - Fixed projection of initial point into bounded interval after linear infeasibility model.
  - Fixed issue with Hessian evaluation when multithreading is enabled.
- Solver alias `conopt` now equals `conopt4`. Use `conopt3` to solve with CONOPT 3.

CPLEX

- Added possibility to define multiple mipstopexpr in one option file. If any of the given stop expressions is true, the algorithm stops.

GUROBI

- New libraries 11.0.0.
  - MINLP: Can now use spatial branching and outer approximation to solve models with non-linear constraints in a form that is supported by Gurobi, instead of using static piecewise-linear (PWL) approximations.
  - New option concurrentmethod: Chooses continuous solvers to run concurrently.
  - New option tunedynamicjobs: Enables distributed tuning using a dynamic set of workers.
  - New option mixingcuts: Mixing cut generation.
  - New option funcnonlinear: Controls whether general function constraints are treated as nonlinear functions or via PWL approximation.
  - New option solutiontarget: Specify the solution target for LP.
  - New option value `-2` for tuneresults.
  - Changed both upper bound and default for option tunetimelimit to infinity.
  - Changed default for option nonconvex to accept models with non-convex objective or constraints.
  - Changed default for option dofuncpieces to use the relative error approach.
- New option nlreform: Reform nonlinear equations to Gurobi general constraints.
- Added possibility to define multiple mipstopexpr in one option file. If any of the given stop expressions is true, the algorithm stops.

GUSS

- In rare (mostly internal) places GUSS was previously called `SCENSOLVER`, e.g. in `gmscmpun.txt` files or the system attribute `solverNames`. This synonym has been removed and replaced by `GUSS`.

JAMS

- Update solution reporting for VI agents: for VI functions that have been flipped, the value of the matching variable is now returned in the equation marginal, instead of its negative.
- Fixed model reformulation for VI agents: corrected reformulation in cases where VI constraints were flipped.

KESTREL

- The client-side implementation now strictly enforces TLS 1.2 or later.

KNITRO

- New libraries 14.0.0.
  - Performance improvements on mixed-integer and QP/QCQP problems.
  - Improvemed presolve.
  - Improvemed quasi-Newton Hessian approximations.
  - New option linsolver_nodeamalg: Controls the node amalgamation setting for the MA57, MA86, and MA97 linear solvers.
  - New option linsolver_ordering: Sets the ordering method used for the linear system solver.
  - New option linsolver_scaling: Enables scaling for the linear system solver.
  - New option scale_vars: Specifies the strategy for scaling variables.
  - New option presolveop_substitution_tol: Tolerance for applying a substitution.
  - New option presolveop_substitution: Determine whether or not to enable the Knitro presolve operation to substitute out variables when possible.
  - New option bar_globalize: Specifies the globalization strategy used in the interior-point algorithms.
  - New option bar_maxmu: Specifies the maximum allowable value for the barrier parameter μ used with the barrier algorithms.
  - New option mip_cut_flowcover: Specifies rules for adding flow cover cuts.
  - New option mip_cut_probing: Specifies rules for adding probing cuts.
  - New option mip_heuristic_localsearch: Specifies whether or not to enable the MIP local search heuristic.
  - New option ms_initpt_cluster: The strategy for clustering initial points in multi-start.
  - New option value `4` for ms_terminate.
  - New option values `-1` and `4` for blasoption.
- For discrete models the initial point is now added as MIP start in addition to the root relaxation start.

Octeract

- As announced, Octeract has been removed.

PATH

- Avoid multi-threaded Jacobian evaluation for models with external equations (aka `=X=` rows) as thread-safe external evaluation cannot be assumed.

SCIP

- Added sassy 1.1 (9847fa1) as graph preprocessor for symmetry detection.

XPRESS

- Added possibility to define multiple mipstopexpr in one option file. If any of the given stop expressions is true, the algorithm stops.

3.3.1.6 Tools

$LibInclude files

- As announced, the libinclude files `gdxservice`, `linalg`, and `win32` have been removed. Please use the corresponding tools from the GAMS tools library instead.

Cholesky, Eigenvalue, Eigenvector, Invert

- As announced, the tools cholesky, eigenvalue, eigenvector, and invert have been removed. Please use the linear algebra tools from the GAMS tools library instead.

GAMS IDE

- Fixed an error reading certain reference files.

GAMS Studio

- New version 1.17.2.
  - Added install license button for the "GAMS Licensing" dialog.
  - Added highlighting of the system log tab when new messages are available. In case of warnings and errors the tab will be focused.
  - Added new options "Compare Defaults", "Compare Domains", and "Ignore Order" to the GDX Diff dialog.
  - Added the ability to resume to an ongoing GAMS Engine job after Studio is restarted.
  - Added symbol sub types to the "Type" column of the symbol table in the GDXViewer.
  - GDXViewer export improvements:
    - The dialog now uses the ExcelReader instead of the PandasExcelReader (deprecated). This changes the following aspects:
      - All GAMS special values are exported as string representation per default.
      - Visual appearance of the generated Excel files has changed.

        - Exported data does not contain the "value" and "element_text" headers anymore.
        - No extra blank row between column header and data anymore.
        - Labels are not written into merged cells anymore.
    - Added support for specifying custom values for GAMS special values.
- Navigator improvements:
    - Improved performance after large searches.
    - Fixed duplicate entries for files that are also opened in a tab.
    - Fixed file navigation when starting path with ...
    - Fixed problem handling long path names.
    - Fixed directories being listed as files.
    - Fixed filesystem navigation for absolute path on Windows.
    - Fixed filesystem navigation improperly handling file switching.
    - Fixed that unnecessary spaces broke Navigator modes.
- Stability improvements, bug fixes, and minor enhancements, e.g.:
    - Added Esc as a shortcut to abort the search.
    - Added abort of ongoing search when search dialog is closed (but not hidden).
    - Improved LST viewer performance.
    - Improved performance when searching very large folders.
    - Improved file handling for unexpected events while writing a file, i.e., write to temporary file and rename after success.
    - Fixed crash when closing a project during a GAMS debug run.
    - Fixed crash when opening file from welcome page.
    - Fixed crash when loading an empty lxi/lst file.
    - Fixed crash in lxiViewer with invalid model index.
    - Fixed crash in project explorer on macOS with invalid model index.
    - Fixed crash in Connect Editor when adding ExcelWriter schema data.
    - Fixed problem where "Replace All" in unopened files appended extra characters.
    - Fixed stop and interrupt button for macOS and Linux.
    - Fixed LST viewer sometimes jumping to wrong position in file.
    - Fixed help view showing empty page when document linked to a page in new tab or new window.
    - Fixed unresponsive `Abort` button in Search Dialog.
    - Fixed that search was not properly interrupted when pressing `Abort`.
    - Fixed deactivated search elements when project settings are focussed.
    - Fixed unwanted partial matches in the Search Dialog file filter section.
    - Fixed project explorer not updated when using "save as".
    - Fixed encoding resets to UTF-8 after restart.
    - Fixed Windows interrupt not working in some cases.
    - Fixed `Open As Text` button in Connect Editor not showing file contents in text editor.
    - Fixed `Save` for newly added data by drag and drop schema in Connect Editor.
    - Fixed overwritten clp when moving/copying a file from one project to another.

GDX2HAR/HAR2GDX

- As announced, the tools gdx2har and har2gdx have been removed. The tools can be accessed from the `CoPS web site`.

GDXRank, GDXRename

- As announced, the tools gdxrank and gdxrename have been removed. Please use the GDX service tools from the GAMS tools library instead.
- The `$libInclude rank` functionality now uses `executeTool 'alg.rank'` to sort a one dimensional parameter instead of the gdxrank tool.

MCFilter

- As announced, the tool mcfilter has been removed. This discussion at the GAMS World Forum gives an alternative.

MOO

- A new LibInclude tool moo has been added. It provides methods for multi-objective optimization in GAMS (e.g. Augmented Epsilon Constraint).

MSAppAvail and Shellexecute

- As announced, the tools msappavail and shellexecute have been removed. Please use the Windows only tools from the GAMS tools library instead.

3.3.1.7 APIs

C++

- Added client-side enforcement of TLS 1.2 or later in `GAMSJob::runEngine`.
- New libraries curl 8.6.0.

C4U

- As announced, the C4U API has been removed from the distribution.

GAMS Transfer Matlab

Attention

GAMS Transfer Matlab has been moved and restructured! The API

- is located in `api/matlab` (previously: `apifiles/Matlab/api`) and
- comes as the Matlab package `gams.transfer` (previously: `GAMSTransfer`).

- Now available for macOS on ARM64 CPUs.
- Added `gams.transfer.setup` and `*.c` MEX source files of internal interface to `gams.transfer.cmex`. Calling `gams.transfer.setup` allows to build GAMS Transfer Matlab from source. Check with `mex -setup` which C compiler is enabled in Matlab.

GAMS Transfer Python

- Symbol property `.container` now holds a reference to the `Container` instead of a `weakref`.
- Relaxed axis uniqueness requirement when setting records with `uels_on_axes=True`.
- Fixed error encountered with non-unique axis elements when setting records with `uels_on_axes=True`.
- Fixed that the `pivot` method failed for DataFrames with non-`str` column names.
- Fixed failure with `getSparsity` by adding new `isValid` test to efficiently detect domain violations in certain situations.
- Fixed bug that did not allow mixed case `gdx` file extensions.

GAMS Transfer R

- New libraries 3.0.0.
  - GDX API changed to the new GDX C++ API, which allows for significant performance improvements.
  - Performance improvements in reading GDX file.
  - The `systemDirectory` argument in the `Container` constructor is deprecated and will be removed in the future.
  - Bug fix: `Container` property `summary` changed from a method to active binding.
  - Bug fix: `Symbol` method `equals` incorrectly returned `FALSE` when the symbol records contain `NA`.
  - Fixed overriding `PKG_CXXFLAGS` in Makevars.

GDX

- New libraries 7.10.1 (synchronized with open-source release of GDX expert-level API):
  - Added `gdxAllowBogusDomains` as boolean option to toggle allowing potentially unsafe writing of records to symbols with one dimensional sets as domain, when GDX has no lookup table for the elements of this set. This can happen when `gdxStoreDomainSets` was disabled by the user to save memory. For backwards compatibility, this is enabled by default. When the user explicitly disables it, e.g. via `gdxAllowBogusDomainsSet(false)`, then using a one dimensional set as domain will cause a GDX error (`ERR_NODOMAINDATA`).
  - Added boolean property `gdxMapAcronymsToNaN` (disabled by default) that maps all acronym values to special value "Not a Number" (NaN).

GEV

- Add alternative function `gevSwitchLogStatEx` to better handle cases where the current log/status destination is independent of the one switched to.

GMO

- Added `gmoObj_None` to enum `gmoObjectiveSense` for model instances without objective. This return value of `gmoSense()` was possible before, but is used more consistently now.

Matlab

Attention

The (object-oriented) Matlab API has been renamed, moved and restructured! The API

- is now called GAMS Control (previously: object-oriented GAMS API),
- is located in `api/matlab` (previously: `apifiles/Matlab/api`),
- comes as the Matlab package `gams.control` or its alias `gc` (previously: `GAMS`),
- now has sub-packages `gams.control.engine`, `gams.control.globals` and `gams.control.options`, and
- removed `GAMS` from class names. For example, use `gams.control.Workspace` (or the alias `gc.Workspace`) instead of `GAMS.GAMSWorkspace`. The old class names are still available as aliases to the new names, but are deprecated and may be removed in an upcoming release. See also mapping of old to new class names.

Furthermore, the enumeration classes have changed. They are now normal classes that behave like enumeration classes for better support of software products that do not support enumerations, e.g., Octave. This has changed:

- Enumeration values are now constant properties of the class. Syntax for using those values does not change.
- The new enumeration-like classes have properties `select` and `value` that store the current selection (name of the enumeration value) and value of the enumeration, respectively. These properties can be changed to other enumeration selections or values. Use `select` when printing the enumeration.
- Removed method `lookup`. Use the class constructor instead.
- Removed method `value`. Use the property `value` instead.
- Setting the environment variable `GAMS_MATLAB_ENUM_ALT` has no effect anymore.

- Added option value `SplitBlocksDumpWithComments` to option `DumpOpt`: Write processed input with all comments into a separate dump file for each block.
- Removed option `PyMultInst`.

.Net

- Changed the `TargetFramework` of the GAMS .Net API and all dependent examples from 4.5 to 4.6.2.

PAL

- Added routine `palExistingConfigFiles()` that returns an ordered list of GAMS configuration files.

Python

- The `gams.transfer.numpy` API has been moved to `gams.core.numpy`.
- Added new value `KeepFilesOnError` to **DebugLevel** which keeps temporary files only in case of an error. Its internal numerical value is `1` and all values for `DebugLevel.KeepFiles` or greater have been increased by one.
- Changed the default of **GamsWorkspace** from `DebugLevel.Off` to `DebugLevel.KeepFilesOnError` with the effect that temporary files do not get deleted anymore in case of an error.
- The client-side implementation of `gams.control.GamsJob.run_engine()` now enforces TLS 1.2 or later when using `urllib3` prior to version 2.0.
- Improved log output of **GamsJob.run()** to be always live (per line).
- Fixed log output of **GamsJob.run()** containing duplicate carriage returns for line endings on Windows.
- Fixed **GamsOptions.optdir** property used with **GamsModelInstance.instantiate()** having no effect in **GamsModelInstance.solve()**.

3.3.1.8 Model Libraries

FIN Library

- Updated **CreditImmunization : Factor Immunization model for corporate bonds.**.
- Updated **MeanVar : Mean-variance efficient portfolios**.
- Updated **MeanVarShort : Mean-variance model allowing short sales**.
- Updated **PutCall : Put/Call efficient frontier model.**.
- Updated **ReadData : Reads excel files and converts them to gdx format**.
- Updated **SelectiveHedging : Scenario Optimization for selective hedging**.
- Updated **ThreeStageSPDA : A three stage stochastic programming model for SPDA**.

GAMS Data Library

- Removed `embeddedMultiInstance`.
- Added **EngineSolve.gms : Demonstrate how to submit just a solve statement to Engine**.
- Added **EngineSolveAsync.gms : Demonstrate how to asynchronously submit solve statements to Engine**.
- Added **moo01.gms : Solve scalable multi-objective knapsack model**.
- Added **moo02.gms : Solve multi-objective power generation model**.
- Updated **connect01.gms : Complex Connect Example with Spreadsheets**.
- Updated **connect05.gms : Simple Connect Example for Excel**.

GAMS Model Library

- **dice** and **dicex**: Fixed big-M and ensured `fval` to be integral after solve in `dice`.
- Added **boxpacking.gms : Container Packing Problem**.
- Added **cvrp.gms : Capacitated Vehicle Routing Problem**.
- Updated **cta.gms : Controlled Tabular Adjustments**.
- Updated **herves.gms : Herves (Transposable Element) Activity Calculations**.
- Updated **qmeanvar.gms : Financial Optimization: Risk Management using MIQCP**.
- Updated **sddp.gms : Multi-stage Stochastic Water Reservoir Model solved with SDDP**.

GAMS Test Library

- Removed `har1`.
- Added **unload16.gms : Test command line parameter gdxSymbols**.
- Added **emp34.gms : JAMS: test of flipping functions with VI**.
- Added **emp35.gms : JAMS: test of flipping constraints with VI**.
- Added **moo1.gms : Test libInclude tool moo**.
- Added **caxlsr.gms : Test Connect agent ExcelReader**.
- Added **caxlsrw.gms : Test Connect agents ExcelReader and ExcelWriter**.
- Added **gamsincr.gms : Test GAMS incrementalMode**.
- Updated **capcode.gms : Test Connect agent PythonCode**.

PSOPT Library

- Updated **DED-PB : Price based Dynamic Economic Load Dispatch**.
- Updated **DED-wind : Dynamic Economic Load Dispatch considering Wind generation**.
- Updated **DED : Dynamic Economic Load Dispatch**.
- Updated **DEDESS : Cost based Dynamic Economic Dispatch integrated with Energy Storage**.
- Updated **DEDESSwind : Cost based Dynamic Economic Dispatch integrated with Energy Storage and Wind**.
- Updated **ESSDCOPFwind : DC-OPF integrated with Energy Storage and Wind**.
- Updated **MultiperiodACOPF24bus : Multi-period AC-OPF for IEEE 24-bus network considering wind and load shedding**.
- Updated **MultiperiodDCOPF24bus : Multi-period DC-OPF for IEEE 24-bus network considering wind and load shedding**.
- Updated **PBUC : Price based Unit commitment**.
- Updated **RampSenDED : Ramp rate sensitivity analysis for Dynamic Economic Load Dispatch**.

3.3.2 46.2.0 Minor release (March 05, 2024)

3.3.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Jinggang Guo, Scott McDonald, and Anne Sofie Skak-Iversen.

3.3.2.2 GAMS System

Embedded Code Facility

- Fixed error if embedded Python/Connect code arguments (`gams.arguments`) contain single quotes.

GAMS Connect

- Fixed ExcelReader error when reading from empty sheet.
- Fixed ExcelWriter error when writing an empty symbol.
- Improved ExcelReader error messages for insufficient ranges.

3.3.2.3 Solvers

BARON

- New libraries 24.3.1.
  - Performance improvements for QCQPs.

CONOPT

- New libraries 4.33.

MOSEK

- New libraries 10.1.27.

3.3.2.4 Tools

GAMS Studio

- New version 1.17.3 with bug fixes and minor enhancements:
  - Added "Last Projects" to the welcome page.
  - Added close button to remove an entry from the welcome page.
  - Added search and replace for block-edit.
  - Fixed that project file got cleared when opening a file of the same base name.

3.3.2.5 APIs

DCT

- Fixed a wrong error message in case of an overflow in `dctSetBasicCountsEx`.

3.3.2.6 Model Libraries

GAMS Test Library

- Updated **caxlsr.gms : Test Connect agent ExcelReader**.
- Updated **caxlsrw.gms : Test Connect agents ExcelReader and ExcelWriter**.
- Updated **connectsub.gms : Test substitution for Connect**.
- Updated **embpy01.gms : Test for embedded code facility**.
- Updated **embpy02.gms : Test for embedded code facility**.

3.3.3 46.3.0 Minor release (March 19, 2024)

3.3.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Nick Sahinidis, and Feng-Sheng Wang.

3.3.3.2 GAMS System

GAMS

- New Put_Utility `xsave`: Writes a compressed save file of the current state of execution.
- New dollar control option $xsave[.keepCode] to create a compressed work file during compilation.

Embedded Python Code Facility

- Fixed a problem with code blocks with comments only.
- Fixed `%` character not being diplayed correctly when using `gams.printLog`.

GAMS Connect

- Added a warning to the `ExcelWriter` agent in case `openpyxl<3.1.0` is used which can lead to wrong data being exported.
- Fixed bug where `Concatenate` agent would not preserve the UEL order.

3.3.3.3 Installer

- Fixed incorrect capitalization in macOS uninstall script when asking for confirmation to remove GAMS.

3.3.3.4 Solvers

BARON

- New libraries 24.3.10.

CONOPT

- New libraries 3.17P.
- Fixed error with too large threadf value in CONOPT 4. Will be clipped at 64 now.
- Fixed interfaces for solving model instances without equations but bound constraints (only relevant when calling solver directly via GMO API).

COPT

- Fixed library load problems on some Linux systems.

GUROBI

- New libraries 11.0.1.

HiGHS

- Variable level values are no longer passed as initial solution to an LP solve, as this disabled LP presolving.

MOSEK

- New libraries 10.1.28.

3.3.3.5 Tools

GDXDUMP

- Fixed a problem with long element text when writing in CSV format.

MOO

- The tool will now throw an error in case of an empty dynamic objective set.
- Fixed bug where objective values of zero would not be fixed while calculating the payoff table.
- In addition to raising a warning, the tool will now skip objectives with min=max after the payoff table calculation. If there is not more than one objective without min=max left, the methods are skipped and the pareto points from the payoff table are returned.
- Fixed bug where sandwiching returned incorrect savepoint files for anchor points.

GAMS Studio

- New version 1.17.4.
  - Fixed crash on close a tab with active block edit selection.
  - Fixed font initialization error.
  - Fixed project not created when using "Open File".
  - Fixed behavior of running GAMS Engine job on Studio exit.
  - Fixed keep file modified when saving the file fails.

3.3.3.6 APIs

Python

- Added minimum version `3.1.0` for `openpyxl` in `pip` extras for `connect`.

3.3.3.7 Model Libraries

GAMS Test Library

- Updated **caconcat.gms : Test Connect agent Concatenate**.
- Updated **connectsub.gms : Test substitution for Connect**.
- Updated **embpy01.gms : Test for embedded code facility**.
- Updated **embpy02.gms : Test for embedded code facility**.
- Updated **moo1.gms : Test libInclude tool moo**.

3.3.4 46.4.0 Minor release (April 02, 2024)

3.3.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Torbjoern Jansson, Daniel Oliveira, and Evangelos Panos.

3.3.4.2 GAMS System

GAMS

- Modified behavior of dumpOpt for value 22 to not split into further dump files when processing an include file.

Embedded GAMS Code Facility

- Fixed bug with embedded GAMS code when executing in a directory with a long path name.

GAMS Connect

- Fixed bug in `SQLReader` agent where `indexSubstitutions` on stacked indices in case of more than one value column would result into `NaN`.

3.3.4.3 Solvers

Convert

- Fixed issue writing opening parenthesis for function arguments in new line.

3.3.4.4 Tools

GAMS Studio

- New version 1.17.5.
  - Fixed: Prevent Alt keys from closing GDX Viewer filter dialogs.

GDXDUMP

- Fixed bug when writing symbol text that starts with `(`.

3.3.4.5 Model Libraries

GAMS Data Library

- Updated **GMSPythonCheck.gms : Consistency check for GMSPython**.

GAMS Test Library

- Updated **casqlr.gms : Test Connect agent SQLReader**.

3.3.5 46.4.1 Maintenance release (April 16, 2024)

3.3.5.1 Acknowledgments

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

3.3.5.2 GAMS System

GAMS

- Fixed a bug causing $declareAndLoad to throw an error when an alias to the universe should be declared.

3.3.5.3 Solvers

CBC

- New libraries CoinUtils 2.11.11.

CONOPT

- Fixed default setting of LFITER and RVTIME to solver default.

KESTREL

- Removed EMP from the list of supported model types since Kestrel currently does not support `EMP` models.

3.3.5.4 APIs

.NET

- Fixed `GAMS.GAMSOptions.OptDir` property used with `GAMS.GAMSModelInstance.Instantiate` having no effect in `GAMS.GAMSModelInstance.Solve`.

3.4 45 Distribution

3.4.1 45.1.0 Major release (October 14, 2023)

3.4.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Saran Chanpuang, Aidana Ermatova, Michael Ferris, Adderly Huerta, Auke Greijdanus, Mogens Bech Laursen, and Christoph Pahmeyer.

3.4.1.2 Platforms

- We will drop support for macOS 11 (Big Sur) with the next major release.
- Added support for macOS 14 (Sonoma).

3.4.1.3 GAMS System

GAMS

- Added an additional note to the log about the expiration date of the license used.
- Extended the $declareAndLoad capability to also declare new acronyms found in the input GDX file.
- Minor performance improvement for execute_unload.
- Fixed a bug which caused a wrong dump file when $onEmpty is active.
- Fixed a broken message in the log when a $call gets executed with a command that exceeds 255 characters.
- Fixed a data-dependent bug that resulted in an unexpected execution error and was triggered by re-loading data at execution time in a loop via Embedded Code.
- Fixed sys15 being ignored in option statements.

Embedded Python Code Facility

- We plan to drop the command line parameter `pyMultInst` with a future major release. The ability to maintain independent Python sessions can not be maintained in a reliable way.

GAMS Connect

- The CSVWriter now supports unstacking multiple dimensions to header rows.

GMSPython

- The Python version of GMSPython has been updated to `3.8.17`.

3.4.1.4 Documentation

- Added list of 156 third-party codes that are redistributed with GAMS.

3.4.1.5 Installer

- The Windows installer does not create the `GAMS` sub directory in the users documents directory anymore.

3.4.1.6 Solvers

CONOPT

- New libraries 4.31.
  - Added option Flg_AdjIniP to disable sometimes expensive adjustments to initial point.
  - Added option Flg_NoDefc to disable definitional constraints.
  - Added evaluation of post-triangular constraints if a CNS model was interrupted.
  - Improved AttemptCNS.
  - Changed the logic for improving the conditioning of the basis to fix problems with superbasic slacks.
  - Function evaluation errors are made silent after ill-conditioned steps.
  - Reset reduced Hessian and superbasis after singularities or numerical problems.
  - Fixed a problem in SQP when a critical bound was infinite. This could have given huge steps and domain errors.
  - Fixed an error in the pre-processor where minimax constraints could be converted into an incorrect equality.

CONVERT

- Fixed writing `eps` for `0` as variable coefficients in GAMS format.

COPT

- New libraries 7.0.1.
  - Performance improvements.
  - New option BarStart: method to compute starting point for Barrier solver.
  - New option FAPHeurLevel: level of fix-and-propagate heuristic.

CPLEX, GUROBI, MOSEK, FICO XPRESS

Attention

The bare-bone link mode (previously OsiCplex, OsiGurobi, OsiMosek, OsiXpress), which required only a GAMS base license and a standalone solver license to solve LP or MIP, has been **removed**. Please contact `sales@gams.com` to arrange for a solver-link license to be added to your GAMS license.

GUROBI

- New libraries 10.0.3.

HiGHS

- New libraries 1.6 (c070f1253).
  - Improved handling of user-interrupt.
  - New option mip_min_logging_interval.
- Added functionality to write solve trace files. New options solvetrace, solvetracenodefreq, and solvetracetimefreq.

Ipopt

- New libraries METIS 5.2.1.
- Fix in METIS adapter code used by HSL linear solvers.

MOSEK

- New libraries 10.1.13.
  - Improved branching variable selection: Automatic choice of variable selection method and adjustment of strong branching work limits. New options MSK_IPAR_MIO_VAR_SELECTION and MSK_IPAR_MIO_MIN_REL.
  - Knapsack cover cuts are now enabled by default, that is, changed default of option MSK_IPAR_MIO_CUT_KNAPSACK_COVER from `MSK_OFF` to `MSK_ON`.
  - Added conflict analysis based on dual rays. New option MSK_IPAR_MIO_DUAL_RAY_ANALYSIS_LEVEL.
  - New option MSK_IPAR_MIO_MAX_NUM_RESTARTS.
  - New option MSK_IPAR_PRESOLVE_LINDEP_NEW.
  - Removed options `MSK_IPAR_CHECK_CONVEXITY` and `MSK_IPAR_LOG_CHECK_CONVEXITY`.
  - Removed value `MSK_STARTING_POINT_SATISFY_BOUNDS` for option MSK_IPAR_INTPNT_STARTING_POINT.

SCIP Optimization Suite

- New libraries PaPILO 2.1 (9ef45bfa).
- New libraries SoPlex 6.0 (5a9b664d).
- New libraries SCIP 8.0 (6c8c7c8d90).

SHOT

- New libraries 1.1 (424b5e48).

3.4.1.7 Tools

GAMS Studio

- New version 1.16.1.
  - Removed the "Go To" dialog, as the functionality is integrated into the Navigator.
  - Improvements for the GDX Viewer:
    - Added support for `Shift + Left Click` to select/deselect multiple labels in the GDX Viewer filters.
    - Added thousands separator to symbol record count and symbol number in GDX Viewer.
  - Added HTTPS support to check for update feature, removed dependency on C4U library.
  - Added authorization provider support to GAMS Engine.
  - Stability improvements, bug fixes, and minor enhancements, e.g.:
    - Improved performance of the Navigator for sessions with a lot of files.
    - Fixed check for GAMS license.
    - Fixed crash when reading a corrupted Reference File.
    - Fixed crash on using nested `$echo` commands.
    - Fixed partly saved auto-save files.
    - Fixed recovering outdated auto-save files.

Info-ZIP

- Updated `gmszip` to Zip `3.0-13` on Linux and macOS.
- Updated `gmsunzip` to UnZip `6.0-28` on Linux and macOS.

3.4.1.8 APIs

C

- As announced, removed `gcmt.h`, `gcmt.c`, `gclgms.c`, and `gcdllinit.h` from `apifiles/C/api`.

C++

- Added function `GAMSJob::runEngine` to run jobs on GAMS Engine, rather than locally.
- Adjusted `GAMSJob::run` public function signature.
- Added `transportEngine` example to show usage of new Engine feature.

C4U

- We plan to remove the C4U API from the distribution with an upcoming major release.

GDX

- The user-specified special value mapping set via `gdxSetReadSpecialValues` now also affects the type-specific default values (level, lower, upper, marginal, scale) for empty 0-dimensional (scalar) variables and equations.
- Reverted `gdxSetDomain` to write `0` instead of `-1` as domain index for unknown domains.
- Adjust default behavior to store negative zero as zero. This is more space-efficient due to our internal storage scheme and is consistent with pre-GAMS-38 behavior.
- Fixed incorrect default record for empty equation symbols with equation type different than "equals".

GMD

- Added new function `gmdFindUel` (counterpart for `gmdGetUelByIndex`) which queries the UEL index for a given UEL label. Returns false on failure. UEL index is set to -1 in case the UEL label is not found in the UEL table.
- Performance improvements when dealing with larger datasets due to more efficient symbol and UEL handling.

GMO

- Fixed a bug in functions `gmoGNLNZ64` and `gmoGNLNZ`: they computed counts incorrectly in case of degenerate quadratic forms like `x * (1 + eps*y)`.

GAMS Transfer Matlab

- **Breaking**: Renamed records field `text` of Sets to `element_text`.
- Added symbol `UniverseAlias` to represent aliases to the universe set.
- Added possibility to change symbol name case with `Container.renameSymbol`.
- Added possibility to reorder UELs by record order with `Symbol.reorderUELs` (passing no arguments).
- Added `Container.getSets`, `Container.getParameters`, `Container.getVariables`, `Container.getEquations`, and `Container.getAliases` to get list of symbol objects of corresponding type.
- Added possibility to get/remove all symbols with `Container.getSymbols` or `Container.removeSymbols`, respectively.
- Added `Container.lowerUELs`, `Container.upperUELs`, `Symbol.lowerUELs`, and `Symbol.upperUELs` to convert (all) UELs to lower or upper case, respectively.
- Added columns `where_min` and `where_max` to output of `Container.describeParameters`.
- Changed column names in output of `Container.describe*` methods: `dim` → `dimension`, `num_recs` → `number_records`, `num_vals` → `number_values`, `min_value` → `min`, `max_value` → `max`, `mean_value` → `mean`.
- Removed `*_marginal`, `count_*`, `where_max_abs_value`, and `is_alias` columns in output of `Container.describe*` methods.
- Removed `ConstContainer`.
- Added GDX library unload before each read/write operation.
- Changed used GDX library to `gdxcclib64`.
- Fixed bug that limited the number of used UELs in `Symbol.transformRecords` and `Symbol.getUELs`.
- Fixed `Container.eq` in case of containers with different number of symbols.

GAMS Transfer Python

- Officially removing the beta label.
- Improved performance for both reading/writing of GAMS data from/to GDX and GMD.
- Allow users to update symbol records by repeating calls to the symbol constructor (new objects will not be created).
- Added new symbol/container methods to easily modify UEL strings (`lowerUELs`, `upperUELs`, `lstripUELs`, `rstripUELs`, `stripUELs`, `capitalizeUELs`, `casefoldUELs`, `titleUELs`, `ljustUELs`, and `rjustUELs`).
- Added new container methods to retrieve symbol objects (`getSets`, `getAliases`, `getParameters`, `getVariables`, and `getEquations`).
- Added support for reading data that originated with different encodings.
- Enabled different GDX/GMD `read`/`write` modes (defaults should be used to maintain performance, except in rare instances).
- Passing `None` (new default) to <`container`>`.removeSymbols()` will remove all symbols.
- Passing `None` (new default) to <`container`>`.getSymbols()` will return a list of all symbol objects.
- Renamed symbol property `ref_container` to `container`.
- Added `eps_to_zero=True` (changes default behavior) argument to <`container`>`.write()` to control writing all GAMS EPS (i.e., `-0.0` in GTP) special values to `0.0`.
- Enabled the symbol property `name` (and <`container`>`.renameSymbol()`) to change the casing of a symbol name.
- `Container` property `system_directory` can now accept either `str` or `PathLike` objects.
- Added `symbols` argument to `Container` methods (`getDomainViolations`, `hasDomainViolations`, `countDomainViolations`, `dropDomainViolations`, `getDuplicateRecords`, `hasDuplicateRecords`, `countDuplicateRecords`, `dropDuplicateRecords`, `isValid`).
- Reorganized the `describe`$*$ tables for clarity and brevity.
- Added `Container.hasSymbols()` method to test for symbols in a container (can also use the `in` operator).
- Updated the symbol `summary` properties for all symbol types.
- Added a container `summary` property.
- Added support for setting GAMS special values with `str` types when setting records with MultiIndex DataFrames.
- Function `setRecords` no longer drops zero valued records when setting records with MultiIndex DataFrames.
- Added new symbol methods to drop (potentially) unwanted records: `dropZeros` (parameters only), `dropDefaults`, `dropNA`, `dropUndef`, `dropMissing`, and `dropEps`.
- Exposed a new public property `default_records` for variables and equations.
- Added support for Pandas 2.1.0.
- Function `toDense` now checks categorical ordering before creating an array (for `relaxed` domain symbols).
- Fixed a bug when reading in an Alias symbol from GMD objects.
- Fixed a bug where a symbol's UEL (categorical) order was not maintained from GDX.

GAMS Transfer R

- New libraries 2.6.
  - Officially removing the beta label.
  - **Breaking**: removed all `Const*` classes. `ConstContainer` is no longer supported. Use `Container` instead.
  - **Breaking**: Symbol field `refContainer` renamed to `container`.
  - **Breaking**: removed `getUniverseSet` method from `Container`. Use `getUELs` instead.
  - Records columns that are not specified by the user in a dataframe are not auto completed to save memory. For example, a set with only the domain columns will now not have the `element_text` columns. A default value is assumed for the missing attributes.
  - New field `defaultValues` for symbols of type `Parameter`, `Variable`, and `Equation`.
  - Reorganized `describe*` `Container` methods for clarity and brevity.
  - Refined field `summary` for `Symbol` objects for clarity.
  - `Container` objects now have a `summary` field.
  - New convenience methods for `Container` objects: `getSets`, `getAliases`, `getParameters`, `getVariables`, and `getEquations`.
  - Enabled registering any UEL using `uelPriority` argument in `Container` method `write`.
  - `Container` method `write` now supports `mapped` write mode with the help of new argument `mode`.
  - Following `Container` methods do not support `list` input for `symbols` argument: `describeSets`, `describeAliases`, `describeParameters`, `describeVariables`, `describeEquations`, `read`, `removeSymbols`, `getSymbols`. Use `vector` inputs instead.
  - Argument `types` in the `Container` methods `listVariables` and `listEquations` does not support the input of type `list`. Use `vector` inputs instead.
  - Argument `uelPriority` in the `Container` method `write` does not support the input of type `list`. Use `vector` inputs instead.
  - Fixed bug in failed `Symbol` constructor call resulting in symbol being added to the `Container`.
  - Fixed bug in registering unused UELs when not all symbols from the container are written to a GDX file.
  - Fixed bug in read for `Variable` and `Equation` classes where the `lower` and `upper` attributes were interchanged.
  - Fixed bug in the Symbol method `equals` where identical domain symbols exist in different containers.
  - Fixed bug in Symbol methods `reorderUELs` and `toDense` for symbols with `relaxed domain`.

Jupyter Notebooks

- The newly introduced back-end class gams.magic.GamsInteractive can be used to translate GAMS Jupyter notebooks into standalone Python scripts.

Matlab

- Added GAMS Engine support:
  - Added classes **GAMSEngineConfiguration**, **GAMSEngineJob**, **GAMSEngineJobBuilder**, and **GAMSEngineRunParameters**.
  - Added method **GAMSJob.runEngine()**.

  - Added example `transport_engine.m` to show how to run a job with GAMS Engine from within the GAMS Matlab API (requires Matlab with at least Java SE 11).

- Added classes **AbstractRunParameters**, **GAMSRunParameters**, **MIIMode**, **Replace**, **SolveOpt**, and **PrintStream** (thin wrapper around a Java print stream).
- Changed method signatures of **GAMSJob.run()** and **GAMSModelInstance.solve()** to use **PrintStream** for output handling.
- Changed handling of Java null pointers: Instead of raising an error, GAMS Matlab now uses `[]` to indicate null.
- Fixed internal handling of Java lists, sets, and maps.

PAL

- Added new functions to query the expiration date for time-limited licenses (`palLicenseGetEvalDate`) and the maintenance end-date for perpetual licenses (`palLicenseGetMaintDate`).

Python

- As announced, dropped support for Python 3.7.
- The `gams.numpy` API has been moved to `gams.transfer.numpy` and is now considered an internal API of `gams.transfer`. The documentation as well as the examples `g2np_example[1|2].py` have been removed and users are encouraged to use `gams.transfer` as a replacement.
- Renamed the `gams` Python wheel to `gamsapi`, this change will not impact import statements, but users will need to use install differently with `pip install gamsapi`.
- New optional parameter `container` for `gams.connect.ConnectDatabase` constructor that allows to use an already existing instance of `gams.transfer.Container`.
- Fixed wrong `model_status/solver_status` after calling `gams.control.GamsModelInstance.solve()` with `debug=True`.

3.4.1.9 Model Libraries

GAMS API Library

- Dropped `Pgams2numpy`.
- Updated **PBuildXPLevelAPI.gms : Test building and installing the GAMS Python API from source distribution**.

GAMS Data Library

- Updated **GMSPythonLib.gms : GMSPYTHONLIB compatibility check**.

GAMS Model Library

- Improved numerics of **fdesign.gms : Linear Phase Lowpass Filter Design**.

GAMS Test Library

- Updated **gams2numpy01.gms : Test gams.core.numpy Python API string mode**.
- Updated **gams2numpy02.gms : Test gams.core.numpy Python API raw mode**.
- Updated **gamsjupyter01.gms : Test GAMS Jupyter Notebooks**.
- Added **knitro02.gms : KNITRO test suite - model setup issue**.
- Added **knitro03.gms : KNITRO test suite - model setup issue**.

3.4.1.10 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
ALPHAECP 2.11	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓
BARON	✓	✓	✓	✓
CBC 2.10	✓	✓	✓	✓
CONOPT 3	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓
COPT 7.0	✓	✓	✓	✓
CPLEX 22.1	✓	✓	✓	✓
DECIS	✓	✓	✓	✓
DICOPT 2	✓	✓	✓	✓
GUROBI 10.0	✓	✓	✓	✓
GUSS	✓	✓	✓	✓
IPOPT 3.14	✓	✓	✓	✓
HiGHS 1.6	✓	✓	✓	✓
KESTREL	✓	✓	✓	✓
KNITRO 13.2	✓	✓	✓	✓
LINDO 14.0	✓	✓	✓	✓
LINDOGLOBAL 14.0	✓	✓	✓	✓
MILES	✓	✓	✓	✓
MINOS 5.6	✓	✓	✓	✓
MOSEK 10	✓	✓	✓	✓
NLPEC	✓	✓	✓	✓
OCTERACT 4	✓	✓		
ODHCPLEX 7	✓	✓		
PATH	✓	✓	✓	✓
QUADMINOS 5.6	✓	✓	✓	✓
SBB	✓	✓	✓	✓
SCIP 8.0	✓	✓	✓	✓
SHOT 1.1	✓	✓	✓	✓
SNOPT 7.7	✓	✓	✓	✓
SOPLEX 6.0	✓	✓	✓	✓
XPRESS 41.01	✓	✓	✓	

3.4.2 45.2.0 Minor release (October 30, 2023)

3.4.2.1 Acknowledgments

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

3.4.2.2 Solvers

CBC

- New libraries Cbc 2.10.11 and Clp 1.17.9.

COPT

- New libraries 7.0.2.
  - New option SolTimeLimit to set a timelimit (for the whole solve) that is only effective after a feasible solution has been found.

3.4.2.3 APIs

C++

- Added CMake option `USE-GCC` for example subproject to change library selection to prefer GCC over clang.

GAMS Transfer R

- New libraries 2.8.
  - Fixed bug in reading symbols from GDX with unused UELs from domain.
  - Fixed bug in symbol method `getDomainViolations` when the domain symbol has `NULL` records.

Java

- Fixed wrong `ModelStatus/SolveStatus` after calling `GAMSModelInstance.solve()` with `debug` option set to `true`.

.Net

- Fixed wrong `ModelStatus/SolveStatus` after calling `GAMSModelInstance.Solve()` with `debug=true`.

3.4.3 45.3.0 Minor release (November 11, 2023)

3.4.3.1 Acknowledgments

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

3.4.3.2 GAMS System

GAMS

- Fixed a problem with command line parameter etLim which was ignored in many cases at compile time.

3.4.3.3 Solvers

Ipopt

- New libraries Ipopt 3.14.13.
- New libraries MUMPS 5.6.2.

MOSEK

- New libraries 10.1.16.

SCIP Optimization Suite

- New libraries SCIP 8.0 (6f841064d0).
- New libraries PaPILO 2.1 (ee0677c4).

3.4.3.4 Tools

GAMS Studio

- New version 1.16.2 with bug fixes and minor enhancements:
  - Enhanced GAMS Engine dialog:
    - Compute remaining time depending on instance factor and always show remaining seconds.
    - Added support for `JobTag` for job identification.
    - Fixed alignment.
    - Fixed error message when server not found.
  - Fixed rare crash in search.
  - Fixed search result not clickable after changing search term.

GAMS Tools Library

- Modified tool ExcelTalk to provide better error messages. Moreover, the matching of the workbook name is done on the base name (without the path) of the file.

3.4.3.5 APIs

Python

- Added support for Python 3.12.

Control

- Fixed unwanted side effect when passing `engine_options` dictionary to `gams.control.GamsJob.run_engine()` method.
- `gams.control.GamsJob.run_engine()` now correctly handles the response status code `429 Too Many Requests` from the server.

Transfer

- Fixed a bug that prevented syncing of records (in jupyter environments) when setting records with a tabular dataframe or int/float data type.

3.4.4 45.4.0 Minor release (November 27, 2023)

3.4.4.1 Acknowledgments

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

3.4.4.2 MPSGE

- Fixed issues solving MPSGE-generated problems with multithreading enabled due to incorrect parallel Jacobian evaluation.

3.4.4.3 Solvers

COPT

- New libraries 7.0.3.

MINOS and SNOPT

- Fixed incorrect global optimal model status for non-convex models with linear constraints.

3.4.4.4 Tools

GAMS Studio

- New version 1.16.3 with bug fixes and minor enhancements:
  - Relaxed boundaries of the GDX Viewer splitter to allow for more resizing flexibility.
  - Reduced allowed SSL warnings when the user accepts self-certified GAMS Engine connection.
  - Fixed that command line changes were lost when switching to another project.
  - Fixed that a double click on a `.gsp`-file did not activate the project.
  - Fixed that the position of the search dialog was not saved when closing with close button in titlebar.

GAMS Tools Library

- Fixed a bug in tool ExcelMerge when accessing the number of sheets in a workbook and improved error messages.

3.4.4.5 APIs

GEV

- Disable multi-threaded Jacobian evaluation in `gevEvalJacLegacyX` for model instances with external equations (aka `=X=` rows): it cannot be assumed that the user-provided evaluation function is thread-safe.

Python

- Fixed a Python 3.12 `SyntaxWarning` in the Control API.

3.4.5 45.5.0 Minor release (December 14, 2023)

3.4.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Rob Davies, Gabriel Homsi, and Sherman Robinson.

3.4.5.2 GAMS System

GAMS Connect

- Fixed a bug in the PandasExcelReader that prevented reading columns containing both numerical values and dates.

3.4.5.3 Solvers

MOSEK

- New libraries 10.1.21.
  - Option MSK_DPAR_CHECK_CONVEXITY_REL_TOL is no longer effective.

SCIP

- New libraries 8.1.0.

3.4.5.4 Tools

GAMS Tools Library

- Fixed a bug in tool ExcelTalk when Excel files are located in (sub)directories.
- Added command `open` and named argument `quit=0|1` to tool ExcelTalk.

3.4.5.5 APIs

GMD

- Fixed memory leak related to not freeing a symbol record iterator when updating a symbol inside a model.

Python

- Fixed a potential memory leak in the control API observed when iterating multiple times over the same symbol. This was also affecting `GAMSModelInstance.solve()`.
- Fixed a warning when building the `gamsapi` wheel using the source distribution.

3.4.6 45.6.0 Minor release (January 04, 2024)

3.4.6.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Simon Wesley Bowen, Rob Davies, and Mart Saluri.

3.4.6.2 GAMS System

GAMS

- Changed $loadDCR to not complain anymore when it should be used to clear a set, that is used as domain, if that set had no data so far.
- Fixed a bug which could have caused a wrong dump file with an empty data statement when dumpOptGDX is set.
- Fixed a bug which could have caused a wrong dump file when $loadM is used while dumpOptGDX is set.
- Fixed a potentially wrong result of a sameAs statement, when it is used in a loop construct with a singleton set which is empty when entering the construct and gets a value in it.

GAMS Connect

- Improved exception messages of several agents when handling symbols that do not exist.

3.4.6.3 Solvers

COPT

- New libraries 7.0.4.

SCIP

- New libraries 8.1 (0dff9f8a3e).

3.4.6.4 Tools

GAMS Studio

- New version 1.16.4 with bug fixes:
  - Fixed that check for update crashed when Studio is opened and immediately closed.
  - Fixed check for update redirect policy (`NoLessSafeRedirectPolicy`) and check for update behavior when disabled.
  - Fixed wrong sorting behavior in the export dialog of the GDX Viewer.

3.4.7 45.7.0 Minor release (January 18, 2024)

3.4.7.1 Acknowledgments

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

3.4.7.2 GAMS System

GAMS

- Fixed MIIMode to implicitly set keep=1 as expected.
- Fixed a problem when loading from a GDX file at compile time which could have caused an incomplete load of data, if the data depends on a domain set that was reloaded before from the same file replacing already existing records.
- Fixed a bug which could have caused a broken dump file with certain data statements.

GMSPython

- Packages `Jinja2` and `urllib3` have been updated to version `3.1.3` and `2.1.0`, respectively, which include fixes for known vulnerabilities.

3.4.7.3 Solvers

COPT

- New libraries 7.0.6.

IPOPT

- New libraries 3.14.14.

MOSEK

- Reduce overhead when calling MOSEK for final solve and avoid its output to stdout when it should have been disabled.

3.5 44 Distribution

3.5.1 44.1.0 Major release (July 20, 2023)

3.5.1.1 Acknowledgments

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

3.5.1.2 GAMS System

GAMS

- Added new function gdxLoad to have a flexible way of loading from GDX files at execution time. The load behavior can be modified using the new options filtered and replace. This new function can be used for both, explicit loading of single symbols but also a bulk load of all symbols in the GDX file that match the declaration in the model.
- The requirements for logic equations have been relaxed to make them more useful in future releases. For example, continuous variables and non-boolean functions/operators can now be present in logic equations. This relaxed use of algebra in logic equations prevents some simple preprocessing steps, e.g. `not not x` is only `x` if `x` is either 0 or 1. Since this is not the case anymore, such preprocessing has been disabled. This might impact some (linear) reformulation of pure boolean logic equations done by LOGMip.
- Added new command line parameter solveOpt to initialize the already existing option `solveOpt`.
- Added new command line parameter MIIMode.
- As announced, the command line parameter logOption was changed. `LogOption=1` was marked as deprecated and became a synonym for the default `logOption=3`, which sends the log output to the standard output.
- PreviousWork now writes a workfile compatible with GAMS 43 and newer.
- Fixed a potential bug with $save and put_utility save.

GAMS Connect

- As announced, the `debug` option of the Options agent has been dropped.
- The CSVReader now supports using the symbolic constant `lastCol` without a `header` or `names`. If no `header` or `names` is provided, `lastCol` will be determined by the first line of data.
- The CSVReader now supports reading multi-row headers for parameters.
- Added LabelManipulator agent to Connect.

3.5.1.3 Installer

- Docker images with a pre-installed GAMS distribution for linux/amd64 are now available at Docker Hub.

3.5.1.4 Licensing

- A GAMS demo license is now included in the GAMS distribution. It is valid for approximately 5 months.

3.5.1.5 Documentation

- The HTML documentation now uses a new style based on Doxygen Awesome.

3.5.1.6 Solvers

BARON

- New libraries 23.6.22.
  - Improved continuous and integer presolve and facilities for quadratic programs.
  - New reformulator.
  - Added options AllowCbc, AllowCplex, AllowHsl, AllowXpress to adjust which solvers can be picked by automatic LP/MIP/QP solver selection.
  - Added support for macOS on ARM64 CPUs. Ipopt and Xpress cannot be used as NLP and LP/MIP/QP solver, respectively, by BARON on this platform so far, that is, options AllowIpopt and AllowXpress are 0 by default.

Gurobi

- New libraries 10.0.2.

HiGHS

- New libraries 1.5.3.
  - New option presolve_reduction_limit.
  - New option ipx_dualize_strategy.
  - Renamed option `simplex_dualise_strategy` to simplex_dualize_strategy.

IPOPT

- New libraries MUMPS 5.6.1.

LINDO/LINDOGLOBAL

- New libraries 14.0.279.

Mosek

- New libraries 10.0.46.

Octeract

- We plan to drop Octeract with one of the following major releases.

SCIP Optimization Suite

- New libraries SCIP 8.0 (bf58b8fcd5).
  - Removed non-default values for option propagating/symmetry/recomputerestart, that is, recomputing symmetry information after a restart can no longer be enabled (as this can produce wrong results).
- New libraries PaPILO 2.1 (2ed99c46).

SELKIE

- As announced, the research solver SELKIE has been dropped from the GAMS distribution. It is expected that it will be made available directly from the authors.

3.5.1.7 Tools

GAMS Studio

- New version 1.15.1.
  - New feature GAMS Debugger: This allows to pause the execution at the beginning of execution statements and review the current data in a temporary GDX file.
  - Improvements for the GDX Viewer:
    - Added search facility to the data view.
    - Added settings to specify default numerical formatting options and show/hide attributes.
    - Disable preferences menu for sets and aliases.
  - Improvements for the GAMS Configuration Editor:
    - Move tab selection from bottom to top.
    - Fixed status in definition section when editing a `Key`.
  - Improvements for the GAMS Configuration Editor / Parameter File Edior / Extended Parameter Editor:
    - Add `group` in parameter/option definition.
    - List all available solvers for `Solver` parameter/option definition.
  - Stability improvements, bug fixes, and minor enhancements, e.g.:
    - Fixed OpenSSL issue for Manjaro, Ubuntu 22.04 LTS and similar Linux distributions.
    - Fixed a bug where `precision=Full` was not restored properly in the GDX Viewer.
    - Fixed path issue for project files: existing projects with a path starting with "." can be fixed manually using "Move Project File..."

GDX2ACCESS

- Fixed gdx2access for systems where both 32-bit Office and 64-bit Access runtime are installed.

MDB2GMS

- Fixed mdb2gms for systems where both 32-bit Office and 64-bit Access runtime are installed.
- Made table browser in MDB2GMS not crash when field sizes for memo or long binary fields cannot be queried (now showing `???` instead).

3.5.1.8 APIs

C

- File `apifiles/C/api/gcmt.c` does no longer need to compiled to use C or Fortran API files. Files `gcmt.c` and `gcmt.h` are now empty and will be removed in a future version.
- File `apifiles/C/api/gclgms.c` does no longer need to compiled to have the arrays and functions from `gclgms.h` defined (they are inlined now). File `gclgms.c` is now empty and will be removed in a future version.
- File `apifiles/C/api/gcdllinit.h` is now empty and will be removed in a future version.
- The definitions `GAMSVERSION`, `GAMSMAJOR`, `GAMSMINOR`, and `GAMSGOLD` in `gclgms.h` are deprecated and will be removed in a future version.

Delphi

- The Delphi API files in `apifiles/Delphi` will be removed with a future major release.

Fortran

- The Fortran API files in `apifiles/Fortran` will be removed with a future major release.

GAMS Transfer Python

- Breaking: Dropped `ConstContainer` and all `Const*` symbols – users should only use `Container` and accompanying symbols.
- Method `setRecords` (and the `records` argument) now supports setting records with `pandas.Series` and `DataFrames` with `Index` or `MultiIndex` axes (must set `uels_on_axes=True`).
- Method `toDense` now requires domain sets to have self-consistent ordering (i.e., record data order and category order must be equal).
- Method `reorderUELs` now reorders UELs to data order (and append unused categories) if `uels` is `None` (the default).
- Fixed bug when attempting to load libraries from different GAMS versions with the `system_directory` argument.

GAMS Transfer R

- New libraries 2.2.0
  - Fixed bug with library unload upon read or write.
  - Performance improvement in setting records for symbols.
- Note: `ConstContainer` and the corresponding symbols will be dropped with the next major GAMS version. Users are advised to use `Container` and accompanying symbols.

GDX

- The dynamic library file on Linux (.so), macOS (.dylib), and Windows (.DLL) has been changed from `(lib)gdxdclib64` to `(lib)gdxcclib64` after porting the GDX codebase from Delphi to C++. While the GDX API remains compatible, the underlying shared library is very different and cannot be renamed back. Therefore, a copy of the old `(lib)gdxdclib64.{so,dylib,DLL}` is still distributed with GAMS for now. Eventually the Delphi library will be dropped, hence it is sensible to compile existing user applications against the new C++-based GDX library and corresponding API files.
- Modified function `gdxDataReadRawFastEx` to allow the callback function to stop the reading after each record. The callback must now return an integer indicating if reading should be continued (>=1) or stopped (=0).

GMD

- Fixed a problem that resulted in a crash when retrieving the domain list with an alias of symbol (`gmdGetDomain`).

GMO

- As announced, the functions `gmoEvalFuncNLCluster` and `gmoEvalFuncNLCluster_MT` have been removed.

Java

- Added method `runEngine` to `GAMSJob` class to run jobs on GAMS Engine.
- Changed minimum requirement of Java SE to compile and run a Java program using the latest GAMS Java API:
  - All classes in `[Path/To/GAMS]/apifiles/Java/api/GAMSJavaAPI.jar` require Java SE 11 or later to run.
  - `GAMSJavaAPI.jar` has an additional dependency on `JSON.simple`, located in `[Path/To/GAMS]/apifiles/Java`
  - `GAMSJavaAPI.jar` and `json-simple-1.1.1.jar` are required to be in the same directory to run.
- Changed GAMS Java API name for a Java program that still requires Java SE 8 to compile and run:
  - All classes that target Java SE 8 are in `[Path/To/GAMS]/apifiles/Java/api/GAMSJavaAPI-8.jar`.
  - New or updated functionalities may not be made available in this version, only corrective maintenance support will be provided in the future.
  - `GAMSJavaAPI-8.jar` has no additional dependency.

Python

- We plan to drop support for Python 3.7 in a future GAMS release.

3.5.1.9 Model Libraries

GAMS Test Library

- Dropped `SELKIE` test models `selkie01` through `selkie19`.
- Added **load18.gms : Test gdxLoad**.
- Added **calabelm.gms : Test Connect agent LabelManipulator**.
- Added **solveopt01.gms : Test solveopt option and command line parameter**.

3.5.1.10 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
ALPHAECP 2.11	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓
BARON	✓	✓	✓	✓
CBC 2.10	✓	✓	✓	✓
CONOPT 3	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓
COPT 6.5	✓	✓	✓	✓
CPLEX 22.1	✓	✓	✓	✓
DECIS	✓	✓	✓	✓
DICOPT 2	✓	✓	✓	✓
GUROBI 10.0	✓	✓	✓	✓
GUSS	✓	✓	✓	✓
IPOPT 3.14	✓	✓	✓	✓
HiGHS 1.5	✓	✓	✓	✓
KESTREL	✓	✓	✓	✓
KNITRO 13.2	✓	✓	✓	✓
LINDO 14.0	✓	✓	✓	✓
LINDOGLOBAL 14.0	✓	✓	✓	✓
MILES	✓	✓	✓	✓
MINOS 5.6	✓	✓	✓	✓
MOSEK 10	✓	✓	✓	✓
NLPEC	✓	✓	✓	✓
OCTERACT 4	✓	✓		
ODHCPLEX 7	✓	✓		
PATH	✓	✓	✓	✓
QUADMINOS 5.6	✓	✓	✓	✓
SBB	✓	✓	✓	✓
SCIP 8.0	✓	✓	✓	✓
SHOT 1.1	✓	✓	✓	✓
SNOPT 7.7	✓	✓	✓	✓
SOPLEX 6.0	✓	✓	✓	✓
XPRESS 41.01	✓	✓	✓	

3.5.2 44.1.1 Maintenance release (August 03, 2023)

3.5.2.1 Acknowledgments

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

3.5.2.2 Solvers

COPT

- New libraries 6.5.7.

HiGHS

- New libraries 1.5.4.

NLPEC

- Fixed that the setting of option NCPBounds was not being honored for doubly-bounded variables.
- Fixed implementation of reformulation type `FB_neg`.
- Fixed implementation of reformulation type `min` for the case of upper-bounded variables and no slack variables.

3.5.2.3 APIs

GAMS Transfer Matlab

- Fixed bug when using more than 255 UELs.

GAMS Transfer R

- New libraries 2.4.0.
  - Fixed read for `Variable` and `Equation` classes where the `lower` and `upper` attributes were interchanged.

3.5.3 44.2.0 Minor release (August 17, 2023)

3.5.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Mogens B. Laursen, Orenzo Porporino, and Hannes Wornig.

3.5.3.2 GAMS System

GAMS

- Fixed errors with $declareAndLoad and $gdxLoadAll which could have caused problems when dealing with aliases.
- Fixed a memory leak with the Multi-Threading Solve Facility (`solveLink=6/7`).

Embedded Python Code Facility

- Fixed a problem when command line parameter `pyMultInst` has been set to 1. Please note that the interrupt signal (e.g., from Ctrl-C) in embedded Python code does not work with `pyMultInst=1`.

GAMS Connect

- Allow the LabelManipulator agent to convert multiple labels to the same target label.

3.5.3.3 Installer

- Fixed a bug of the Windows installer overwriting an existing license file from a previous installation with the demo license.

3.5.3.4 Solvers

SoPlex

- Improved performance for passing the LP to SoPlex.

3.5.3.5 Tools

GAMS Studio

- New version 1.15.3 with some bug fixes and minor enhancements:
  - Fixed missing OpenSSL library on Linux.
  - Fixed missing projects when a project path is missing.
  - Fixed a bug in the GDX Viewer where restoring a symbol state of a partially loaded symbol resulted in a crash.
  - Fixed crash on closing a GDX Viewer instance while symbol data is loading.
  - Fixed GAMS process not terminating when closing Studio.
  - Improved handling of interrupt and stop.

3.5.4 44.3.0 Minor release (September 01, 2023)

3.5.4.1 Acknowledgments

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

3.5.4.2 Solvers

BARON

- Fixed that setting GAMS option threads to a negative value did not enable parallelization in MIP solver.

CBC

- Fixed that solver did not stop on interrupt signal.

CPLEX

- Fixed model status for CPLEX's termination `CPXMIP_OPTIMAL_POPULATED_TOL` and incorrect infeasibility warning for `CPXMIP_OPTIMAL_POPULATED` and `CPXMIP_OPTIMAL_POPULATED_TOL`.

HiGHS

- Fixed issue when running HiGHS more than once in the same process with multiple threads enabled.

NLPEC

- Fixed an incorrect reformulation (required equations were omitted from the reformulated model) that occured when aggregate was set to `partial none`.
- Fixed case where an invalid option combination (aggregate set to `full` but different reformulation type set for single and double) was allowed: this is now shifted to a valid option combination.

SHOT

- Fixed use of multiple threads with CBC as MIP solver.

3.5.4.3 Tools

GAMS Studio

- New version 1.15.4 with some bug fixes and a minor enhancement:
  - Added tab key support for filter widgets in GDX Viewer.
  - Fixed the GDX Viewer not displaying symbol data with too many records by only showing about the first 107 million entries.
  - Fixed path selection for GAMS license installation.

3.5.4.4 Model Libraries

GAMS Test Library

- Updated **casqlr.gms : Test Connect agent SQLReader**.
- Updated **casqlw.gms : Test Connect agent SQLWriter**.
- Added **nlpec03.gms : NLPEC test suite - loop over option combos**.

3.5.5 44.4.0 Minor release (September 19, 2023)

3.5.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Alan Fox, Erwin Kalvelagen, and Nick Sahinidis.

3.5.5.2 GAMS System

GAMS

- Fixed potential problem with $declareAndLoad when loading symbols with name clash for internal symbols or functions.

3.5.5.3 Solvers

CONOPT 3

- New libraries 3.17O.

CONVERT

- Fixed equation type in `=B=` and `=X=` equations in GAMS format.

3.5.5.4 Tools

GAMS Studio

- New version 1.15.5 with a bug fix and a minor enhancement:
  - Fixed crash in Reference File Viewer when searching for a symbol in `FileUsed` tab.
  - Removed the obsolete studio command line option `--help-all`.

3.5.5.5 APIs

GDX

- Fixed a bug causing large GDX files to be read incorrectly due to storing the number of remaining bytes (in the file) in a variable of wrong (not wide enough) type.

3.6 43 Distribution

3.6.1 43.1.0 Major release (April 27, 2023)

3.6.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Pedro Andres Sanchez-Perez, Mogens Bech Laursen, Bruce McCarl, Scott McDonald, and Thomas Rutherford.

3.6.1.2 Platforms

- The GAMS distribution is now also available as a native build for macOS 13 on ARM64 CPUs (Apple M1/M2). Compared to the macOS system for x86_64 CPUs, these are the differences:
  - Third-party software for which no native builds are available (so far):
    - BARON and hence no GAMS/BARON
    - FICO XPRESS and hence no GAMS/XPRESS
    - Matlab and hence no GAMS Transfer Matlab
  - Effects from `Intel's Math Kernel Library` not being available:
    - Pardiso is not available as linear solver in Ipopt.
    - Intel MKL is not available as blasoption for Knitro.
    - `Apple's Accelerate Framework` is used instead by ANTIGONE, CBC, IPOPT, SCIP, and SHOT.
  - The clang compiler frontend is Apple's preferred way to build C/C++ programs on macOS. A GCC variant of the high-level C++ API library has therefore not been included for this platform. GCC users can build the library from `source`.
  - Components marked as deprecated have not been ported:
    - GDXMRW
    - GDXRRW
  - Support for Python 3.7 and macOS versions 11 and 12 are not included.

3.6.1.3 GAMS System

GAMS

- The command line parameter logOption will change with the next major release. Currently, `logOption=1` sends the log output to the console (i.e. `con:` or `/dev/tty`). This feature is planned to be dropped. `LogOption=1` will become a synonym for the default `logOption=3`, which sends the log output to the standard output.
- Added new command line parameter writeOutput, which allows to suppress the creation of the output file completely.
- Added new keyword executeTool and put_utility 'execTool' to invoke tools from the GAMS tools library (see below) at execution time.
- Added new dollar control options $callTool and $hiddenCallTool to invoke tools from the GAMS tools library (see below) at compile time.
- Added new dollar control options $gdxLoadAll and $declareAndLoad, which allow a convenient import of all (even undeclared) symbols from a GDX file at compile time.
- Added new dollar control options $on/offImplicitAssign, which allow to suppress compilation error 141 ("Symbol declared but no values have been assigned."). This is useful in situations where the compiler is unaware that symbols are loaded implicitly (e.g. `execute_loadpoint "solutionfile.gdx"`).
- Added a new section to the end of the reference file that lists all files included.
- Extended some internal limits to allow generation of huge scalar models.
- Renamed compile time constants `platformCode.DEG` and `platformCode.LEG` to `platformCode.DEX` and `platformCode.LEX`, respectively, to be consistent with the system attribute `platform`.
- Minor adjustment to the return code of the function readyCollect: In the past, it returned `0` only if it waited until a job is ready to be collected. Now, it also returns `0` when there is a job ready to be collected without the need to wait for it (that case returned `1` in the past; now the meaning of `1` has changed to "There is neither an active job to wait for nor a job ready to be collected").
- Fixed function platformCode (was not working properly on macOS and Linux).
- Fixed an issue converting strings of length 255, which could cause problems when creating GDX files.
- Fixed a bug where scalars where not updated properly when implicitly loaded from embedded code inside a loop.

Documentation

- The PDF version of the documentation will be dropped in a future release.

Embedded Code Facility

- React on interrupt signal (e.g., from Ctrl-C) when executing embedded code. If an interrupt signal is received, the execution is interrupted and a compilation or execution error gets issued.
- Setting gams.debug only affects the debug behavior of the `GamsWorkspace` since the exception traceback is always enabled now.
- Fixed exception traceback being written to `stdout` instead of GAMS log.
- Fixed wrong line numbers in exception traceback.

GAMS Connect

- Exception traceback has been enabled for unhandled Python exceptions.
- The `debug` option of the Options agent has been deprecated and will be removed in a future version. Use the Options agent with trace > 0 instead to get the exception traceback of Connect errors .
- Fixed exception traceback being written to `stdout` instead of GAMS log.
- Added RawCSVReader agent to Connect.
- Updated long label renaming logic in RawExcelReader so that the suffix number `~n` gets incremented only when necessary.
- With pandas version 1.5.0, the `mad` aggregation method is deprecated and, therefore, will not be available for the Projection agent in a future release.
- The Concatenate agent now supports concatenating the same symbol more than once.
- The SQLReader and SQLWriter agents (former `PandasSQLReader` and `PandasSQLWriter`) have been revised and now support native SQL connections for faster read/write operations. Both agents now connect to MySQL, Postgres, MS-SQL (SQL-Server), SQLite and PyODBC through their respective python package. `pandas.DataFrame` class' I/O API methods `read_sql` and `to_sql` can be used in combination with `connectionType sqlalchemy` and allows to connect to any other database if the relevant drivers are present on the system.
- The SQLWriter agent supports writeAll. If `True`, all symbols (sets and parameters) in the Connect database will be written to the specified database.

GMSPython

- Added the packages `pymssql` and `pymysql`. For macOS on ARM64 CPUs the additional package `freetds` has been added as a dependency for `pymssql`.

3.6.1.4 Solvers

ANTIGONE, CBC, Ipopt, SCIP, SHOT

- New libraries Intel MKL 2023.1 on Linux and macOS for x86_64.

CBC

- New libraries Cbc 2.10.10, Cgl 0.60.7, Clp 1.17.8, CoinUtils 2.11.8.
  - Extensions to symmetry handling.
  - New values `fastish`, `lightweight`, `moreprinting`, `cuts`, and `cutslight` for option OrbitalBranching.
- New libraries nauty 2.8.6.
- As announced, the CBC alias `COINCBC` has been dropped.

CONOPT

- As announced, the CONOPT3 alias `CONOPTD` has been dropped.

CONVERT

- The AMPL .nl writer can now write equation marginals as well.
- Added options AmplNlInitDual and AmplNlInitPrimal to specify which equation marginal and variable level values to write to AMPL .nl file.
- As announced, the CONVERT alias `CONVERTD` has been dropped.

COPT

- New libraries 6.5.2.
  - Enhanced support for special ordered sets and indicator constraints.
- Added possibility to write solve trace files. New options solvetrace, solvetracenodefreq, and solvetracetimefreq.

CPLEX and OSICPLEX

- GAMS/CPLEX can now be used to solve LPs and MIPs without a GAMS/CPLEX or GAMS/CPLEX Link license, but with a OSICPLEX license. It is thus similar to the GAMS/CPLEX Link license, but restricted to (mixed-integer) linear problems. To use ranging, conflict analysis, feasopt, tuning, column generation, solution pool, lazy constraints, quadratic constraints, indicator constraints, linear user cuts, multi-objective optimization, priorities, benders decomposition, or user callbacks, a GAMS/CPLEX or GAMS/CPLEX Link license is still required.
- GAMS/OSICPLEX is now an alias for GAMS/CPLEX. The alias GAMS/OSICPLEX will be dropped with a future major release. Former GAMS/OSICPLEX option files cannot be processed by GAMS/CPLEX.
- As announced, the CPLEX alias `CPLEXD` has been dropped.

DE

- Introduced an experimental and hidden option `deDict` to produce a text file with dictionary type information of the generated deterministic equivalent.

DECIS

- As announced, the EMP-SP solver DECIS has been dropped. This does not effect the stochastic LP solvers DECISC and DECISM. They remain in the system.

Examiner

- Examiner is now solveLink=5 capable. Some cosmetic output changes were made.

GUROBI and OSIGUROBI

- New libraries 10.0.1.
- GAMS/GUROBI can now be used to solve LPs and MIPs without a GAMS/GUROBI or GAMS/GUROBI Link license if a separate GUROBI license is installed on the machine. It is thus similar to the GAMS/GUROBI Link license, but restricted to (mixed-integer) linear problems. To use quadratic constraints, indicator constraints, general or nonlinear constraints, lazy constraints, multi-objective optimization, multiple MIP starts, partitions, MIP stop expression, solution pool, feasopt, iis, sensitivity analysis, or tuning, a GAMS/GUROBI or GAMS/GUROBI Link license is still required.
- GAMS/OSIGUROBI is now an alias for GAMS/GUROBI. The alias GAMS/OSIGUROBI will be dropped with a future major release.

IPOPT

- As announced, the IPOPT alias `COINIPOPT` has been dropped.
- New libraries 3.14.12.
  - Fixed that for a square problem, the dual solution sometimes did not satisfy optimality conditions even though the problem was claimed to be solved to optimality.

KESTREL

- Removed access to discontinued IBM DOcloud.

LINDO/LINDOGLOBAL

- New libraries 14.0.255.

MILES

- As announced, the MILES alias `MILESE` has been dropped.

MINOS

- As announced, the MINOS aliases `MINOS5` and `MINOS55` have been dropped.

MOSEK and OSIMOSEK

- GAMS/MOSEK can now be used to solve LPs and MIPs without a GAMS/MOSEK or GAMS/MOSEK Link license if a separate MOSEK license is installed on the machine. It is thus similar to the GAMS/MOSEK Link license, but restricted to (mixed-integer) linear problems. To use second-order, semidefinite, power, or exponential cones, a GAMS/MOSEK or GAMS/MOSEK Link license is still required.
- GAMS/OSIMOSEK is now an alias for GAMS/MOSEK. The alias GAMS/OSIMOSEK will be dropped with a future major release.
- GAMS/MOSEK should work equivalently to GAMS/OSIMOSEK in most cases. The main differences are:
  - A MIP solve is now followed by an LP solve with all discrete variables fixed by default. Use option SOLVEFINAL to turn this off.
  - Setting GAMS option integer2 does not enable writing the instance to a file anymore. Set option MSK_SPAR_DATA_FILE_NAME instead.

Octeract

- New libraries 4.7.1.

ODHCPLEX

- New libraries 7.07.
  - New option addcuts: Indicator for adding cuts from CPLEX master solve.
  - New option firstfeaslpitlim: Limit on number of relaxed re-solves in first feasible heuristic.
  - New option firstfeasrelaxcrit: Smallest sum of infeasibilites/row where relaxed solution used.
  - New option ignoresetslvrparams: Flag to control whether solver parameters can be dynamically altered by ODH.
  - New option keypartition: Use of solver partition information.
  - New option keysminimum: Minimum number of keys that the automatic decomposition method attempts to find.
  - New option localsearch: Indicator for local search heuristic.
  - New option loosefeastol: Loose feasibility tolerance.
  - New option objthreshold: Threshold for absolute value of objective coefficients.
  - New option sosfind: Automatic detection of special ordered sets (SOSs).
  - New option sosinkey: Assign each SOS to its own sub-model component (key).
  - New option sosmember: Automatically detect SOSs whose variable member names match this pattern.
  - New option sosovar: Automatically detect SOSs whose output variable name matches this pattern.
  - New option sosselect: Select sub-set of SOS members.
  - New option sosselect16: Select sub-set of SOS members for sets with 16 or fewer members only.
  - New option soswvar: Automatically detect SOSs whose input(weight) variable name matches this pattern.
  - New option strictdeterministic: Terminate ODH deterministically when improvement heuristic finishes.
  - New option subcheckfreq: Frequency with which sub-model LPs are interrupted for mutual communication.
  - New option tightenprebounds: Level of bound tightening in ODH presolved model.
  - New option trialbound: Trial bound heuristic.
  - New option trialboundfile: Trial bound file.
  - New option trialboundsetsize: Size adjustment to automatically generated trial bound sets.
  - New option usehistory: Use of past sub-model selections in current selection.
  - New option varcleanlpmethod: Method used to solve variable cleaning LPs.
  - Changed lower bound of option interdiv to 1.
  - Removed lower bound of option seed. Changed default value to 12345.
  - Changed lower bound of option feastol to 0 and upper bound to 1.
  - Changed lower bound of option subnodelimit to -2.
  - Changed lower bound of option divisor to 2 and default value to 4.
  - Changed upper bound of option decompdensity to 1.
  - Changed upper bound of option firstfeascontinue to 2. Changed option type to integer.
  - Changed default value of option firstfeasshift to 0.
  - Changed default value of option recurseminiterlim to 10.
  - Changed default value of option variableclean to 1.
  - Removed option `newcallback`.
  - Removed option `odhpresolve`.
  - Removed option `subpresolve`.

PATH

- The PATH alias `PATHC` has been dropped.

SCIP

- New libraries bliss 0.77.
- As announced, the SCIP alias `COINSCIP` has been dropped.

SOPLEX

- As announced, the SOPLEX alias `OSISOPLEX` has been dropped.

XPRESS and OSIXPRESS

- New libraries 41.01.03.
- GAMS/XPRESS can now be used to solve LPs and MIPs without a GAMS/XPRESS or GAMS/XPRESS Link license if a separate XPRESS license is installed on the machine. It is thus similar to the GAMS/XPRESS Link license, but restricted to (mixed-integer) linear problems. To use quadratic constraints, nonlinear constraints, indicator constraints, priorities, MIP trace, solution pool, a GAMS/XPRESS or GAMS/XPRESS Link license is still required.
- GAMS/OSIXPRESS is now an alias for GAMS/XPRESS. The alias GAMS/OSIXPRESS will be dropped with a future major release.

3.6.1.5 Tools

$LibInclude files

- The libinclude files `gdxservice`, `linalg` and `win32` are deprecated and will be removed in a future release. Please use the corresponding tools of the more flexible and convenient GAMS tools library instead.

CHK4UPD

- As announced, the command line tool `CHK4UPD` has been dropped. The functionality of that tool is still available through GAMS Studio.

CSDP

- As announced, the tool `csdp` has been dropped. GAMS/MOSEK could be used instead.

GAMS Studio

- New version 1.14.2.
  - Improved command line parameter handling for projects:
    - Added editor for parameter file.
    - Added "Add new GAMS Parameter File" to Project Explorer context menu.
    - Added selection of main and parameter file in Project Settings.
  - Added GAMS Engine authorization by token.
  - Added GDX Viewer setting to control decimal separator used when copying data.
  - Added "File Used" section in reference file viewer.

GAMS Tools Library

- The GAMS tools library is a collection of tools to provide an easy access to a complex task. The traditional tool collection will be replaced over time by GAMS Connect agents and tools in the GAMS tools library. For some time both ways will be supported but the executable tools will go away, so when in doubt what tool to pick, select Connect or a tool from the GAMS tools library.
- The GAMS tools library currently consists of the categories `alg` with Rank, `data` with ExcelDump, `gdxservice` with GDXEncoding and GDXRename, `linalg` with Cholesky, Eigenvalue, Eigenvector, Invert, and OLS (Ordinary Least Squares), and `win32` with ExcelMerge, ExcelTalk, MSAppAvail, and ShellExecute.
- From within a GAMS program, tools from the GAMS tools library can be invoked via $callTool (compile time) and executeTool or put_utility 'execTool' (execution time).
- From a shell/command prompt the script gamstool can be used to run tools from the GAMS tools library.

GDXTROLL

- As announced, the tool `gdxtroll` has been dropped.

MPS2GMS

- MPS2GMS can now be instructed to add up or ignore multiple coefficients for the same term in the objective or a constraint of an LP file. See new option `DUPLICATES` in the MPS2GMS documentation.

3.6.1.6 APIs

GAMS Engine API

- Updated for GAMS Engine version 23.03.31.

GAMS Transfer Matlab

- Breaking: `Symbol.domain_labels` now mirrors the column or field names for domains in `Symbol.records`. Changing `Symbol.domain_labels` will change `Symbol.records` and vice versa. `Symbol.domain_labels` now exists in `struct` and `table` format only. Domain fields in records are those fields that are not one of the following:
  - Variables and equations: `level`, `marginal`, `lower`, `upper`, `scale`.
  - Parameters: `value`.
  - Sets: `text`.

  Furthermore, the default labels changed: If `Symbol.domain_names` is a unique list of domain names, then those names are used as domain labels. Otherwise, the previous label strategy "<name>_<dim>" is used. For example, a symbol with domain {`i, j`}, now has domain labels `i` and `j` in records and with a domain {`i, j, i`} it stays `i_1`, `j_2`, `i_3` (unless the table columns or struct fields are named differently by the user).
- Added possibility to modify `Symbol.domain_labels` to any unique list of domain labels. If `Symbol.domain` is modified, domain labels are reset to default label strategy, described above.
- Changed `Symbol.domain_forwarding` to be a vector of length `Symbol.dimension` to enable/disable domain forwarding for each dimension independently.
- Removed `Symbol.getCardinality`, `Symbol.getUELLabels`, `Symbol.initUELs`, and `Container.getUniverseSet`.
- Fixed possibly incorrect order of UELs of symbols in `dense_matrix` or `sparse_matrix` format.
- Fixed `Symbol.setRecords` for cell input and symbols of dimension >= 3.

GAMS Transfer Python

- Breaking: renamed `generateRecords` argument `densities` to `density`.
- Breaking: removed the `getCardinality` method.
- Breaking: renamed $*$ column for `UniverseAlias` symbols to `uni` to be more consistent with other symbol formats.
- Breaking: new dataframe column naming convention. Defaults to `domain_names` if unique. If not unique, `domain_names` will be tagged with a dimension index. Can be customized with new `domain_labels` setter functionality.
- `Container` and `ConstContainer` have been made iterable.
- Exposed `Const`$*$ symbol classes to the user in order to enable better tab completion behavior.
- New symbol method `toValue` (for scalar symbols) – returns symbol records as `float` (underlying Pandas DataFrame is not modified).
- New symbol method `toList` (for all symbols) – returns symbol records as a python `list`.
- New symbol method `toDict` (for only Parameter, Variable and Equation symbol types) – returns symbol records as a python `dict`.
- Allow a `list` of `bool` to be passed to `domain_forwarding` to selectively forward set elements.
- Following GAMS convention, singleton sets can no longer be used to define a symbol domain.
- Small adjustments to the object string representation (`__repr__`) functions.
- Added an `is_scalar` symbol property to (Const)Variable and (Const)Equation symbols.

- Added support for `len` function to `Container` and `ConstContainer` – returns the number of symbols in the container.
- Added support for `PathLike` objects in `load_from` argument.
- Better handling of `str` type relative paths in `load_from` argument.
- Drop unused categories from columns when using `generateRecords`.
- Fixed bug in `getUELs` when there were unused categories in the domain set and the data were out of order from the category order. Impacted behavior of `toDense` and `toSparseCoo` methods.
- Removed the previously deprecated `getUniverseSet` method, use `<container>.getUELs()` instead.
- Better error message with `whereMin`, `whereMax`, `whereMaxAbs` functions if all symbol values are `NaN`.

GAMS Transfer R

- New libraries 1.16.0.
  - Added checks for singleton and multidimensional checks in `Symbol` method `isValid()`.
  - Changed the default domain labels. User-specified domain labels are preserved if unique.
  - Performance improvement in `Container` method `read`.

GDX

- Added new function `gdxDataReadRawFastEx` that behaves like `gdxDataReadRawFast` but works with a callback function that also has the `DimFrst` and `UserMem` argument.

GMO

- Added new function `gmoGetObjL` to get objective activity level. This is useful when reformulating models.
- Functions `gmoEvalFuncNLCluster` and `gmoEvalFuncNLCluster_MT` are declared deprecated and will be removed with the next major release.

Java

- **GAMSModelInstance** of this version is no longer compatible with GAMS 33 or older.

Matlab

- Added option `implicitAssign` to `GAMSOptions` and enumeration class `ImplicitAssign`.

.Net

- `GAMSModelInstance` of this version is not compatible anymore with GAMS 33 or older.

Python

- Added support for Python 3.11.
- **GamsModelInstance** of this version is not compatible anymore with GAMS 33 or older.
- The Python examples located in <sysdir>/api/python/examples have been updated and revised.
  - The examples xp_example[1|2].py have been renamed to core_example[1|2].py.

3.6.1.7 Model Libraries

GAMS EMP Library

- Updated **farmsp.gms : The Farmer's Problem - Stochastic**.

GAMS Model Library

- Updated **schulz.gms : Termination routine to ensure solvers stay with resource limit**.
- **gqapsdp** and **kqkpsdp**: SDPs are now solved with Mosek.

GAMS Test Library

- Updated **carxr.gms : Test Connect agent RawExcelReader**.
- Added **carcr.gms : Test Connect agent RawCSVReader**.
- Added **output01.gms : Test Output related Command Line Parameters**.
- Added **conopt03.gms : CONOPT test suite - different libraries in one process**.
- Added **interrupt.gms : Check Ctrl-C/Interrupt handling of Embedded Code, MIP solver, and GAMS**.
- Added **load17.gms : Test $declareAndLoad and $gdxLoadAll**.
- Added **implassign01.gms : Test implicitAssign**.
- Renamed capdsqlr to **casqlr.gms : Test Connect agent SQLReader** and updated for revised SQLReader.
- Renamed capdsqlw to **casqlw.gms : Test Connect agent SQLWriter** and updated for revised SQLWriter.

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS

3.6.1.8 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
ALPHAECP 2.11	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓
BARON	✓	✓	✓	✓
CBC 2.10	✓	✓	✓	✓
CONOPT 3	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓
COPT 6.5	✓	✓	✓	✓
CPLEX 22.1	✓	✓	✓	✓
DECIS	✓	✓	✓	✓
DICOPT 2	✓	✓	✓	✓
GUROBI 10.0	✓	✓	✓	✓
GUSS	✓	✓	✓	✓
IPOPT 3.14	✓	✓	✓	✓
HiGHS 1.5	✓	✓	✓	✓
KESTREL	✓	✓	✓	✓
KNITRO 13.2	✓	✓	✓	✓
LINDO 14.0	✓	✓	✓	✓
LINDOGLOBAL 14.0	✓	✓	✓	✓
MILES	✓	✓	✓	✓
MINOS 5.6	✓	✓	✓	✓
MOSEK 10	✓	✓	✓	✓
NLPEC	✓	✓	✓	✓
OCTERACT 4	✓	✓		
ODHCPLEX 7	✓	✓		
PATH	✓	✓	✓	✓
QUADMINOS 5.6	✓	✓	✓	✓
SBB	✓	✓	✓	✓
SCIP 8.0	✓	✓	✓	✓
SHOT 1.1	✓	✓	✓	✓
SNOPT 7.7	✓	✓	✓	✓
SOPLEX 6.0	✓	✓	✓	✓
XPRESS 41.01	✓	✓	✓	

3.6.2 43.2.0 Minor release (May 04, 2023)

3.6.2.1 Acknowledgments

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

3.6.2.2 GAMS System

GAMS

- Improved the dollar control options $declareAndLoad to consider domain information and the subtype of a symbol as well (i.e., distinguish between positive and free variables).

3.6.2.3 Solvers

CPLEX, GUROBI, XPRESS

- Fixed that the bare-bone mode to solve LPs or MIPs was not accessible.

GUROBI

- Fixed that models with quadratic equations were rejected when using model type (D/(R)MI)NLP.

3.6.2.4 Tools

GAMS IDE

- Fixed that GAMS did not run anymore.

3.6.2.5 APIs

GAMS Transfer R

- New version 1.18.
  - Added `isScalar` property for the symbols of type `Variable` and `Equation`.
  - Updated symbol method `reorderUELs()`. If the argument `uels` is not passed, UELs are reordered based on the records.
  - Symbol method `isValid` now checks also for scalars with more than one record entries.
  - Breaking: Symbol method `toDense()` now requires domain UELs and domain records to be in the same order and unused UELs in the domain (if any) at the end of the UEL list.
  - Breaking: Container method `read` now preserves the domain type from the source and avoids domain linking by symbol name.
  - Breaking: Symbol method `getCardinality` is removed.
  - Fixed bug in longer symbol `description`.
  - Fixed bug in accessing Container symbols in a case-insensitive manner.
  - Fixed bug in the Container method `describeAliases`.

Jupyter Notebooks

- New optional argument `--system_directory=<path/to/gams>` for `gams_reset` used to explicitly specify a GAMS system directory.

Python

- The pip extras `all` and `connect` now install `psycopg2-binary` instead of `psycopg2` in order to avoid additional dependencies when installing the GAMS Python API.

3.6.2.6 Model Libraries

GAMS Test Library

- Updated **gamsjupyter01.gms : Test GAMS Jupyter Notebooks**.

3.6.3 43.3.0 Minor release (May 18, 2023)

3.6.3.1 Acknowledgments

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

3.6.3.2 GAMS System

GAMS

- Fixed a wrong default value of 255 for the command line parameter putPW (the default is actually 32767).

3.6.3.3 Solvers

COPT

- New libraries 6.5.3.

HIGHS

- Fixed that solver link crashed when interrupted.

3.6.3.4 Tools

GAMS Studio

- New version 1.14.3 with some bug fixes and minor enhancements, for example:
  - Added `abort` and `abort.noError` to highlighter and completer.
  - Fixed connecting to GAMS Engine via HTTPS on macOS.
  - Fixed GAMS Engine failing when using a parameter file.
  - Fixed false detection of `parmFile` command line parameter.
  - Fixed font not immediately updating after changing it in "Settings > Editor & Log".
  - Fixed crash in Connect Editor when opening a file containing unknown schema name.

3.6.3.5 APIs

Jupyter Notebooks

- Fixed a bug where specifying the `--system_directory` argument of `gams_reset` had no effect.

Python

- Fixed a problem with the debug output of Control API destructors.

3.6.4 43.3.1 Maintenance release (June 01, 2023)

3.6.4.1 Acknowledgments

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

3.6.4.2 GAMS System

Installer

- Fixed a problem of the Windows installer that prevented successful installation with a system account that misses a user documents directory.

3.6.4.3 Solvers

HiGHS

- Fixed crash when resolving model instance after modification of constraint coefficients.

3.6.5 43.4.0 Minor release (June 15, 2023)

3.6.5.1 Acknowledgments

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

3.6.5.2 GAMS System

GAMS

- Removed the capability to execute tools via the put_utility 'execTool'.
- Fixed a bug in $declareAndLoad which caused any Alias being loaded as Singleton Set.

3.6.5.3 Solvers

COPT

- New libraries 6.5.4.

3.6.5.4 Tools

GAMS Studio

- New version 1.14.4 with some bug fixes and minor enhancements, for example:
  - Delay license dialog to ensure Studio is already visible.
  - Added information to system log with Studio start arguments.
  - Added warning to system log if setting files could not be created.
  - Fixed issues with parameter file on macOS and Linux.
  - Fixed project setting for parameter file lost or reset on Studio restart.
  - Fixed OpenSSL issue for Manjaro, Ubuntu 22.04 LTS, and similar Linux distributions.

3.6.6 43.4.1 Maintenance release (June 22, 2023)

3.6.6.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Guido M. Bazzani, Hirotaka Isogai and Bruce McCarl.

3.6.6.2 Solvers

CONOPT

- Fixed that when CONOPT reported about variables or equations to the listing file, the wrong variable/equation was refered to (the prior variable/equation was mentioned instead).

CPLEX

- Fixed that a wrong CPLEX algorithm was called for final solve of a MIQP with fixed discrete variables.
- Automatically change the optimality target to look only for a locally optimal solution (value 2) of a fixed MIQP solve.

3.6.6.3 Tools

GAMS Studio

- New version 1.14.5 with some bug fixes:
  - Fixed error link from lst to code not working.
  - Fixed parameters ignored when no pf-file is assigned to the project.
  - Fixed order of parameters: parameter-box can overrule pf-file.
  - Fixed parameters were cleared when project settings were opened.
  - Fixed pf-file set to `-none-` hasn't been stored.
  - Fixed project file name contains suffix like ".gms" when automatically created.

3.7 42 Distribution

3.7.1 42.1.0 Major release (February 01, 2023)

3.7.1.1 Acknowledgments

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

3.7.1.2 Platforms

- As announced, dropped support for macOS 10.15 (Catalina).
- As announced, dropped support for Windows 8.
- Added support for macOS 13 (Ventura).

3.7.1.3 GAMS System

- The End User License Agreement (`eula.pdf` in the GAMS system directory) has been updated to version 01/2023.

GAMS

- Change for the profile option: Entry `Solve Alg` was renamed to `Solver`. The value reported with it matches now the value from model attribute etSolver. In previous versions, `Solve Alg` was reported to be zero when running with solveLink=0. This has been fixed.
- Improved speed in GAMS and GDX for models with many labels.
- In case the overall GAMS program uses some extrinsic functions but the particular model instance does not, then the extrinsic functions are not loaded any longer into the solver link.
- Added dollar control option $save[.keepCode] to create a work file during compilation.

Connect

- Added new Filter agent to reduce symbol data by applying filters on labels and numerical values.
- The PandasExcelReader now enforces sufficient ranges. A range must cover the complete dimension of a symbol including its value (parameter) or text (set). Reading set element text is mandatory and can no longer be skipped. Dropping unwanted text can be achieved using the Projection agent.
- The PandasExcelWriter supports `.xlsx` and `.xlsm` files. `.xls` files are not supported. For other file formats, such as `.ods` files, it might be required to install additional Python packages.
- The PandasExcelReader supports `.xlsx` and `.xlsm` files. For other file formats, such as `.xls` or `.ods` files, it might be required to install additional Python packages.
- Improved exception messages for various Connect agents.
- Fixed bug where writer agents would fail if a symbol was read empty by one of the reader agents.
- Fixed a bug in the PythonCode agent that prevented local variables to be accessible in a subsequent enclosing namespace.

Embedded GAMS Code Facility

- Added GAMS to the family of supported embedded code languages.

Embedded Python Code Facility

- Fixed a crash when using `gams.printLog()` with LogOption being set to `0` (no log output).

GMSPython

- The Python version of GMSPython has been updated to `3.8.16`.

3.7.1.4 Solvers

BARON

- New libraries 23.1.5.
  - Significant improvements for quadratic programs, tree management strategies, and integer presolve.

CBC

- The time reported back to GAMS (e.g., resUsd attribute) is now measured by the clock that is specified by the clocktype option.
- We plan to drop the CBC alias `COINCBC` with the next major release.

CONOPT

- We plan to drop the CONOPT3 alias `CONOPTD` with the next major release.

CONVERT

- Added concurrent mode (value 3) to option qextractalg.
- We plan to drop the CONVERT alias `CONVERTD` with the next major release.

COPT

- New libraries 6.0.4.
  - COPT can now solve convex MIQCPs and MISOCPs. Enabled GAMS model type MIQCP for COPT.
  - A parameter tuning tool has been added. New options Tuning, TuneParams, TuneMethod, TuneMode, TuneMeasure, TunePermutes, TuneOutputLevel, TuneTargetRelGap, TuneTargetTime, TuneTimeLimit.
  - Option Crossover is now an integer option with possible values -1, 0, and 1.
  - Added possible value 5 for option LpMethod: automatic choice of LP method
- Changed type of option iis from boolean to integer. Added value 2 to request an IIS computation without a previous solve.

CPLEX

- New libraries 22.1.1.0.
  - Added option cardls: decides how often to apply the cardinality local search heuristic (CLSH).
- Added concurrent mode (value 3) to option qextractalg.
- Changed default of rerun to `nono`. With this change the solver may not be able to distinguish between unbounded and infeasible models anymore and may just report the model to be infeasible. In order to check for unbounded models, set rerun to `yes` to rerun the optimization with presolve turned off.
- We plan to drop the CPLEX alias `CPLEXD` with the next major release.

DE

- Added option empInfoFile to DE.

DECIS

- We plan to drop the EMP-SP solver DECIS with the next major release. This does not effect the stochastic LP solvers DECISC and DECISM. They will remain in the system.

GUROBI

- New libraries 10.0.0.
  - Added option softmemlimit: Soft memory limit.
  - Added option obbt: Controls aggressiveness of optimality-based bound tightening.
  - Added option networkalg: Network simplex algorithm.
  - Added support of nonlinear constraint `r =e= sigmoid(x)` (or `r =e= 1 / (1 + exp(-x))`), see Gurobi: Nonlinear Programming.
- Added concurrent mode (value 3) to option qextractalg.
- Changed default of rerun to -1 (no). With this change the solver may not be able to distinguish between unbounded and infeasible models anymore and may just report the model to be infeasible. In order to check for unbounded models, set rerun to 1 to rerun the optimization with presolve turned off.
- Fixed multimipstart: Initial guess could have been sent to Gurobi in a wrong order and thus be rejected by Gurobi, e.g., as infeasible.
- Removed requirement of being a MIP to use mipstart, multimipstart, or varhint because Gurobi may switch to "MIP mode" (branch-and-bound) during solve (e.g. for nonconvex QCPs).

HiGHS

- New libraries 1.4.1.
  - Added option mipstart: Initial level values can be passed as starting point to a MIP (if no semicontinuous or semiinteger variables are present). If infeasible, then HiGHS attempts to find a feasible assignment of the continuous variables by solving an LP.
  - Improved reliability of interior-point solver if cross-over is not always run.
  - Option run_crossover changed from boolean to string. New possible value "choose", to run crossover only if the result without crossover is imprecise.
  - Added options to enable and adjust "iCrash": icrash, icrash_approx_iter, icrash_breakpoints, icrash_dualize, icrash_exact, icrash_iterations, icrash_starting_weight, icrash_strategy.
  - Option write_solution_style: New value -1 now specifies "oldraw" format (was value 2 before); value 2 and new value 3 now allow to print solution in GMPL format.
  - New options presolve_rule_logging and presolve_rule_off.

IPOPT

- We plan to drop the IPOPT alias `COINIPOPT` with the next major release.

KNITRO

- New libraries 13.2.0.
  - New option presolveop_redundant: Determine whether or not to enable the Knitro presolve operation to detect and remove redundant constraints.
  - New option mip_gomory: Specifies rules for adding Gomory mixed-integer cuts.
  - New value for option presolveop_tighten: 2 and 3.
  - New value for option mip_clique: -1 (new default). Removed value 3.
  - New value for option mip_knapsack: -1 (new default). Removed value 3.
  - New value for option mip_zerohalf: -1 (new default). Removed value 3.
  - New value for option mip_liftproject: -1 (new default). Changed option type to integer.
  - Changed default value of option mip_cutting_plane: 1.
- Added concurrent mode (value 3) to option qextractalg.

LINDO/LINDOGLOBAL

- New libraries 14.0.204.
- Added option empInfoFile to LINDO.

MILES

- We plan to drop the MILES alias `MILESE` with the next major release.

MINOS

- We plan to drop the MINOS aliases `MINOS5` and `MINOS55` with the next major release.

MOSEK

- Made parameters MSK_IPAR_PTF_WRITE_PARAMETERS and MSK_IPAR_PTF_WRITE_SOLUTIONS available.
- Added concurrent mode (value 3) to option QEXTRACTALG.

OCTERACT

- New libraries 4.6.0.
  - Removed the possibility to use Gurobi or Xpress as MIP solver. Removed GUROBI and XPRESS from possible values for options `MILP_SOLVER` and `MIP_SOLVER`.

ODHCPLEX

- Added concurrent mode (value 3) to option qextractalg.
- Changed default of rerun to `nono`. With this change the solver may not be able to distinguish between unbounded and infeasible models anymore and may just report the model to be infeasible. In order to check for unbounded models, set rerun to `yes` to rerun the optimization with presolve turned off.

SCIP

- We plan to drop the SCIP alias `COINSCIP` with the next major release.

SELKIE

- We plan to drop the research solver `SELKIE` with the next major release.

SHOT

- Added concurrent mode (value 3) to option ModelingSystem.GAMS.QExtractAlg.

SOPLEX

- We plan to drop the SOPLEX alias `OSISOPLEX` with the next major release.

XPRESS

- New libraries 41.01.01.
  - Dynamic adjustment of the reliability threshold for pseudo cost updates.
  - Improved branching strategies on MIP models with highly degenerate objective function.
  - Extended the effect of numericalEmphasis to some bound tightening reductions. This can help to avoid wrong answers in case of numerically challenging problems.
  - New option globalBoundingBox: If a nonlinear problem cannot be solved due to appearing unbounded, it can automatically be regularized by the application of a bounding box on the variables.
  - New option timeLimit: Maximum time in seconds that the Optimizer will run before it terminates, including the problem setup time and solution time.
  - New option solTimeLimit: Maximum time in seconds that the Optimizer will run a MIP solve before it terminates, given that a solution has been found.
  - New option preConfiguration: Determines whether binary rows with only few repeating coefficients should be reformulated.
  - New option primalOps: Allows fine tuning the variable selection in the primal simplex solver.
  - New options cutSelect_gomory and treeCutSelect_gomory: Strong Chvatal-Gomory cuts.
  - New options cutSelect_farkas and treeCutSelect_farkas: Farkas cuts.
  - New option feasibilityJump: Decides if the Feasibility Jump heuristic should be run.
  - New bits in cutSelect, treeCutSelect and barRegularize.
  - New option value (2) for xslp_solver: Use Xpress-Optimizer if possible (convex quadratic problems only).
  - New default for option xslp_iterLimit: 1000.
  - New default for option xslp_mipDefaultAlgorithm: 3.
  - Option xslp_postsolve has been changed from boolean to integer with new default -1: Postsolve if the problem could be solved to optimality/infeasibility.
  - Options maxStallTime and maxCutTime have been changed from integer to double options.
  - Deprecated options `maxTime` (use timeLimit or solTimeLimit instead), `cacheSize`, `l1Cache` and `sleepOnThreadWait`.
  - Removed options `extraPresolve` and `lpThreads`.
- Added concurrent mode (value 3) to option qextractalg.
- New bit options barPresolveOps_standard, barPresolveOps_extra, barPresolveOps_full.

3.7.1.5 Tools

$LibInclude gdxservice

- Added tool GDXEncoding to convert labels in a GDX file from one encoding to another one.

CHK4UPD

- We plan to drop the command line tool `CHK4UPD` with the next major release. The functionality of that tool will still be available through GAMS Studio.

CSDP

- We will remove the tool/solver `csdp` in a future GAMS release. GAMS/MOSEK should be used instead.

GAMS Studio

- New version 1.13.1.
  - New feature: GAMS Connect editor.
  - Project handling / Project Explorer improvements:
    - Projects get now saved to separate files automatically (`GSP` - GAMS Studio Project).
      - Each project is stored in a separate file, the `Project Settings` show a representation of that.
      - File menu and context menu now provide `Open Project`, `New project`, `Move Project File`, and `Copy Project`.
      - `Import project` and `Export project` have been removed.
      - A project name is tied to the name of the project file.
      - Projects in different paths with the same name get a number added. This number is assigned in the "name and path" order, so changing the path of a project may change the assigned number.
    - Fixed that projects were not always saved properly.
  - GDX Viewer improvements:
    - Improved numerical formatting of the GDX Viewer when using `g-format` with `Full` precision.
    - State of GDX files is now persistent even after restarting Studio.
    - Added symbol filter to GDX state and persist it.
    - Added support for label and value filters in GDX Viewer export functionality.
  - Improved Check for Update dialog, including HTML output and text updates.
  - Studio checks now online for a new available GAMS version when started (can be changed in the settings).
  - Stability improvements, bug fixes, and minor enhancements, e.g.:
    - Improved request behavior of GAMS Engine server communication.
    - Adjusted `Replace All` to show search term in clear text instead of internal representation.

GDXCOPY

- Fixed a problem with missing relaxed domain information in the copied GDX file.

GDXTROLL

- We will remove the tool `gdxtroll` in a future GAMS release.

GDX2XLS

- We will remove the tool gdx2xls in a future GAMS release. GAMS Connect agent PandasExcelWriter should be used instead.

MPS2GMS

- New libraries HiGHS 1.4.1 for LP/MPS reading.

XLS2GMS

- We will remove the tool xls2gms in a future GAMS release. GAMS Connect agent PandasExcelReader should be used instead.

XLSDUMP

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

3.7.1.6 APIs

.Net

- Changed the `TargetFramework` of the GAMS .Net API from 4.0 to 4.5.
- Added method `RunEngine` to `GAMSJob` class to run jobs on GAMS Engine, rather than locally.

Matlab

- Added option SuffixAlgebraVars.
- Added option values RoundedFloatingDec (3) and ForEFloatingDec (4) to option PutNR.

GAMS Transfer Matlab

- Improved performance of `Container.hasSymbols`. Among others, this has a significant effect when adding many symbols.
- Added support of partial write.
- Added parameter `symbols` to `Container.write`, `Container.getDomainViolations`, `Container.resolveDomainVio` and `Container.isValid`.
- Added parameter `allow_merge` to `Container.renameUELs` and `Symbol.renameUELs` in order support merging UELs while renaming (renaming a UEL to an already existing UEL).

GAMS Transfer Python

- New `generateRecords` method to automatically generate records from domain information.
- New `pivot` convenience method to pivot symbol records into various shapes.
- Removed possible `dict` type for `rtol` and `atol` in `equals` method.

GAMS Transfer R

- `removeSymbols` removes symbol links in other symbols.
- Added `symbols` argument to Container methods `renameUELs` and `removeUELs`.
- `findDuplicateRecords` now returns a data frame instead of row indices.
- **Breaking**: `Container` `data` field is now an ordered dictionary from `collections` package instead of named list. Instead of `m$data$<symbolname>`, use `m[<symbolname>]`.
- Performance improvement to `Container` method `hasSymbols`. This results in significant speed-ups when adding multiple symbols to the container.
- Added `symbols` argument to `Container` methods `getDomainViolations`, `hasDomainViolations`, `countDomainViolations`, `dropDomainViolations`, `hasDuplicateRecords`, `countDuplicateRecords`, `dropDuplicateRecords`, `isValid`, and `write` for partial operation.
- `Container` method `getSymbols` now always returns a list.
- Added `equals` method to `Symbols` to check if symbols are equal.
- Bugfix in `SpecialValues$isNA`.
- Bugfix in `getUELs` for scalar symbols.
- Bugfix in `Symbol` method `isValid` for symbols containing only `NA`.
- Bugfix in `Variable` and `Equation` set records for numeric inputs.
- Added `equals` method for `Container` and `ConstContainer`.

Jupyter Notebooks

- Removed methods `gams.pivot` and `gams.pivot2d`. This functionality is now available from GAMS Transfer Python via the pivot method.
- As announced, the package `gams_magic_legacy` has been removed, use `gams.magic` instead.

Python

- The GAMS Python API structure has been revised. Detailed information about the new structure can be found in the Python API documentation:

  Attention

  - Due to the changed structure of the API, `import` statements in existing code might not work anymore - please refer to Migrate import statements for details on how to migrate.
  - Replaced the use of `distutils` by `setuptools`. Installation/uninstallation of the Python API is now done using `pip`.
  - All examples have been moved to `<sysdir>/api/python/examples` and are organized by sub packages.
  - Documentation has been restructured and contains information on the `GAMS numpy API` (previously `gams2numpy`) now.

3.7.1.7 Model Libraries

GAMS Model Library

- Added **binpacking.gms : Bin packing problem with different ways to estimate number of bins**.
- **maxcut.gms : Goemans/Williamson Randomized Approximation Algorithm for Max-Cut** : SDP is now solved with Mosek if called with `--SDPSOLVER=MOSEK`.
- **tablelayout.gms : Configuring text layout in table cells to minimize table height** : Uses now embedded code GAMS.

GAMS Test Library

- Added **scensol10.gms : MCP GUSS Test**.
- Added **cafilter.gms : Test Connect agent Filter**.
- Added **gdxencoding1.gms : Simple gdxencoding test**.
- Added **save3.gms : Test DCO save**.
- Added **embgms01.gms : Test for embedded code facility**.
- Added **embgms02.gms : Test for embedded code facility**.
- Added **embgms03.gms : Test projection operator when loading data from embedded code**.
- Added **embgms04.gms : Test continuation of embedded code blocks**.
- Added **embgms05.gms : Test merge/replace when loading data from embedded code**.
- Added **embgms06.gms : Test domain check/filtered when loading data from embedded code**.
- Added **embgms08.gms : Test filtered load from Embedded Code**.
- Added **embgms09.gms : Test Embedded Code after restart**.
- Added **gdxcopy6.gms : Test gdxcopy with relaxed domain information**.

API Library

- Added **gdxperf.gms : Test various GDX APIs and report run times**.
- Added **generate.gms : Generate some random but structured GDX files**.

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS

3.7.1.8 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.11	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
COPT 6.0	✓	✓	✓
CPLEX 22.1	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GUROBI 10.0	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.14	✓	✓	✓
HiGHS 1.4	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO 13.2	✓	✓	✓
LINDO 14.0	✓	✓	✓
LINDOGLOBAL 14.0	✓	✓	✓
MILES	✓	✓	✓
MINOS 5.6	✓	✓	✓
MOSEK 10	✓	✓	✓
NLPEC	✓	✓	✓
OCTERACT 4	✓	✓	
ODHCPLEX 6	✓	✓	
PATH	✓	✓	✓
QUADMINOS 5.6	✓	✓	✓
SBB	✓	✓	✓
SCIP 8.0	✓	✓	✓
SHOT 1.1	✓	✓	✓
SNOPT 7.7	✓	✓	✓
SOPLEX 6.0	✓	✓	✓
XPRESS 41.01	✓	✓	✓

3.7.2 42.2.0 Minor release (February 16, 2023)

3.7.2.1 Acknowledgments

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

3.7.2.2 GAMS System

GAMS

- Fixed a potential problem that did not allow to execute an embedded code block as first statement in a repeat loop.
- Improved the way, the Grid Facility is executed to resolve a problem when running a GAMS model with solveLink=3 under GAMS MIRO.

3.7.2.3 Solvers

CONOPT

- New libraries 4.30.
  - Fixes an error (stop without termination message) for models with dummy objective function.

3.7.2.4 Tools

MPS2GMS

- Fixed that parameter `COLUMNINTVARSAREBINARY` could not be set.
- Fixed that objective function was lost if MPS reading falled back to the reader for the fixed MPS format.

3.7.2.5 APIs

GAMS Transfer R

- Significant performance improvement when reading from a GDX file.
- Partial domain forwarding is now allowed by passing a logical vector as the `domainForwarding` argument.
- Added `generateRecords` method for `Symbols` to automatically generate records.

3.7.3 42.3.0 Minor release (March 02, 2023)

3.7.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Yuzar Aryadi, Wolfgang Britz, Geoffroy Duparc-Portier, Scott McDonald, and Tom Rutherford.

3.7.3.2 GAMS System

GAMS

- Fixed a bug which could have caused a crash when option clear/kill was used inside a loop on symbol attributes.
- Fixed a bug which caused a potentially wrong dump file when Singleton Sets are loaded.
- Fixed a bug that could have caused an unwanted error in rare cases when relaxed punctuation was used to declare a list of symbols without separating them by comma.

GAMS Connect

- Fixed a problem with labels containing non-printable characters as well as wrong sheet size calculations in RawExcelReader.
- Fixed a problem in RawExcelReader with workbooks that did not have any cells with numerical data.
- Fixed bug in the CSVReader agent where `indexSubstitutions` in a stacked dimension would lead to `NaN`.
- Fixed bug in the CSVReader agent where `indexSubstitutions` would also substitute in the value column of parameters.
- Fixed bug in the CSVReader agent where index and text columns were read as numeric.
- Fixed bug where the CSVWriter agent converts the text column of a set to numeric.
- Fixed bug where the Concatenate agent converts GAMS `NA` into GAMS `UNDF` for a symbol that solely has GAMS `NA` und `UNDF` records.

3.7.3.3 Solvers

CPLEX

- Fix for the case that solvefinal failed: GAMS/CPLEX will now return the solution from the preceeding full solve.

MOSEK

- Added workaround for crash when GAMS/Mosek link terminates after running Mosek with multiple threads.

3.7.3.4 Tools

GAMS Studio

- New version 1.13.3 with some bug fixes and minor enhancements, for example:
  - Added a special group `-GAMS-System-` on project level to bundle files like `Changelog` and `gamsconfig.yaml`.
  - Added support for macro symbol in Reference File Viewer.
  - Enabled controls for empty symbols in GDX Viewer as well.
  - Fixed crash when reloading an empty GDX symbol in table view mode.
  - Fixed several memory leaks.

3.7.4 42.4.0 Minor release (March 16, 2023)

3.7.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Erwin Kalvelagen and Pedro Andres Sanchez-Perez.

3.7.4.2 GAMS System

GAMS

- Fixed duplicate prints of some lines in the output file when $onDollar is set.

GAMS Connect

- Fixed bug where the Projection agent unintentionally alters source symbols in the Connect database.

3.7.4.3 Solvers

BARON

- New libraries 23.3.11.
  - Fixed that a wrong dual bound was reported for a maximization problem when interrupted by the user (`SIGINT`).

CONOPT

- Worked around some library symbol mix-ups when using both CONOPT 3 and CONOPT 4 in the same GAMS process with solvelink 5 or higher on Linux.

COPT

- New libraries 6.0.5.

CPLEX

- Fixed hanging program after fulfilled or ignored mipstopexpr.

HiGHS

- New libraries 1.5.1.
  - New value 4 for option write_solution_style.
  - Changed default for option presolve_rule_logging from 1 to 0.

MOSEK

- New libraries 10.0.38.

SCIP Optimization Suite

- New libraries SCIP 8.0 (1870b6ada8).
- New libraries PaPILO 2.1 (cf0c6277).

SHOT

- New libraries 1.1 (3ba57397).

3.7.4.4 Tools

GAMS Studio

- New version 1.13.4 with some bug fixes and minor enhancements, for example:
  - Added support for Connect filter agent to Studio connect editor.
  - Fixed crash on creating an `.opt` file via context menu in the Project Explorer.
  - Fixed wrong `.opt` file extension which prevented saving.
  - Fixed value edit in GAMS Configuration Editor and allowed to delete `minVersion` and `maxVersion` values.
  - Fixed GdxDiffDialog crash after opening an input file on macOS.

3.7.4.5 APIs

GAMS Transfer Python

- Fixed a bug associated with long symbol descriptions

Python

- Fixed a bug where **GamsWorkspace.add_job_from_file** did not raise an exception in case of a non-existing file.
- Fixed a bug where **GamsModelInstance.copy_modelinstance** did fail if the source instance was instantiated with **GamsModifiers** for parameters.
- Fixed the `==` operator of **_GamsSymbol** and its subclasses that resulted in `True` even for different symbols.
- Fixed **GamsWorkspace.add_database_from_gdx** not raising an exception in case of argument `gdx_file_name` being `None` or the empty string.
- Changed return type of the `==` operator of **_GamsSymbolRecord** and its subclasses from `int` to `bool`.

3.7.5 42.5.0 Minor release (March 30, 2023)

3.7.5.1 Acknowledgments

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

3.7.5.2 GAMS System

GAMS Connect

- Fixed the Projection agent failing on empty symbols.

Installer

- Fixed a problem of the Windows installer that prevented successful installation with a system account (e.g. when using SCCM).

3.7.5.3 Solvers

PATH and PATHNLP

- New libraries 5.0.07.

CONVERT

- Fixed writing `eps` instead of 0 for linear coefficients in GAMS format.

3.7.5.4 Tools

subsubsection g4250_STUDIO GAMS Studio

- New version 1.13.5 with some bug fixes and minor enhancements, for example:
  - Added automatic opening of project editor after creating a new project.
  - Fixed duplicated tab in reference file viewer when reloading a reference file.
  - Fixed crash on macOS when a GAMS related file is opened via double-click in Finder.
  - Fixed broken compilation error link to `.gms` file in `.lst` file.

3.7.5.5 APIs

GAMS Transfer R

- Significant performance improvements when writing a GDX file from a Container.
- Fixed bug in Container read when reading a `Symbol` with unused UELs.
- Fixed bug in `todense` method for `Symbol`.
- Fixed bug in writing empty container.
- Added `Symbol` method `copy` to copy symbol from one `Container` or `ConstContainer` to another `Container`.
- Added `Container` and `ConstContainer` method `copy` to copy symbols to another Container.
- Fixed bug in using ConstContainer Alias methods when the aliased parent set is absent.

3.8 41 Distribution

3.8.1 41.1.0 Major release (October 28, 2022)

3.8.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Sebastian Braun, Wolfgang Britz, Arne Stolbjerg Drud, Michael Ferris, Mitch Phillipson, and Zhang Tao.

3.8.1.2 Platforms

- We plan to drop support for macOS 10.15 with the next major release.
- We plan to drop support for Windows 8.1 with the next major release.

3.8.1.3 GAMS System

Connect

- Symbol names in Connect are now case insensitive. The global attribute `caseSensitiveLookup` has been removed.
- Added new Concatenate agent that allows concatenating multiple symbols into a single symbol.
- The `CSVReader`, `PandasExcelReader`, and `PandasSQLReader` now read relaxed domain information.
- The `PandasSQLReader` now supports pyodbc connections. The connection type can be configured via the new option connectionType.
- Options `readAll` and `writeAll` available in some agents allow and default to value `auto`. So if the `symbols` section is missing, `auto` becomes `True`, otherwise `False`.
- Fixed bug where `PandasExcelWriter` and `PandasSQLWriter` would not properly close the file/database connection in case of an exception.
- Fixed bug where `GAMSWriter` and `GDXWriter` would fail when writing an empty symbol with relaxed domains.
- Fixed bug where `PandasExcelReader` and `CSVReader` would generated `NaN` entries for labels with trailing spaces.

GAMS

- As announced we have removed language elements to access some commercial and licensing information:
  - System suffixes `components`, `clipCodes`, `clipComponentMap`, `componentSolverMap`, `gamsLicenses`, `gamsLicenseTypes`, `lice1`, `lice2`, `licenseDateEvalN`, `licenseDateEvalS`, `licenseDateMaintN`, `licenseDateMaintS`, `licenseDateN`, `licenseDateS`, `licenseDateSysN`, `licenseDateSysS`, `licenseDaysEval`, `licenseDaysMaint`, `licenseDC`, `licenseEval`, `licenseID`, `licenseInstitution`, `licenseLevelText`, `licenseLicensee`, `licenseMudText`, `licensePlatform`, `licensePlatformText`, `licenseStatus`, `licenseStatusText`, `licenseType`, `licenseVendor`, and `vendors`.
  - Predefined sets `componentEDate`, `componentMDate`, and `componentLicenseOptions`.
  - Many of these elements were used in the GAMS Model Library model `licememo`, which we also dropped.
- Added new intrinsic functions lseMax, lseMaxSc, lseMin, and lseMinSc.
- Added new dollar control options $on/offSuffixAlgebraVars and related command line parameter SuffixAlgebraVars to disallow the use of variable suffixes (aka. attributes) in model algebra. This option allows to easily detect unintentional use of variables with suffixes in equation algebra.
- Improved message when exceeding the limit of the GAMS Community License or Demo License.
- Patched up the Windows memory reporting for showOSMemory and procTreeMemMonitor, and added clarifying documentation.

GMSPython

- Added the package `psycopg2`.
- Removed the package `pyyaml-include`.

Embedded Python Code Facility

- On Windows, the directory with the Python DLL is automatically added to `PATH`, so some DLLs are properly found.

3.8.1.4 Solvers

ANTIGONE, CBC, Ipopt, SCIP, SHOT

- Updated MKL to 2022.2.0 on Linux and macOS.

BARON

- New libraries 22.9.30.
  - Better treatment of convex and quadratic problems, including improved relaxations for binary quadratic programs.
  - Improved cutting, range reduction, and branching strategies.
  - Improved continuous and integer presolve algorithms.
  - Improved interfaces to CPLEX, CBC, and IPOPT.

CONOPT

- New libraries 3.17N.
- New libraries 4.29.

CONVERT

- Added option AmplNL: Generates AMPL `.nl` file.
- Added option AmplNLBin: Enables to write `.nl` file in binary format.

COPT

- New libraries 5.0.5.
- Value 1 for option Presolve has been changed to mean "fast presolve". Additional values 2 and 3 have been added to specify normal and aggressive presolving levels.

CPLEX

- Changed iis from boolean to integer option and added option value 2: Conflict analysis without previous solve.
- The time reported in model attribute resUsd now corresponds to the chosen clock type. That is, the CPU time spend by CPLEX is reported if `clocktype` is changed to 1.

DICOPT

- Added option dumpsubprob to activate writing of MIP and NLP subproblems as GAMS files.

GUROBI

- Added support for more nonlinear constraints: 1-norm, 2-norm, and inf-norm specified as nonlinear expressions.
- Deprecated dot-option `.GenConstrType`: Please use nonlinear expressions instead.
- Changed iis from boolean to integer option and added option value 2: Conflict analysis without previous solve.

Ipopt

- New libraries 3.14.10.
- Small performance improvement for problems with i) linear objective function, ii) quadratic objective and linear constraints, iii) only linear equality constraints, or iv) only linear inequality constraints.

Lindo/LindoGlobal

- New libraries 14.0.162:
  - LP Solver Improvements:
    - Improved algorithms for searching alternative optima.
  - MIP Solver Improvements:
    - Improved selection of search parameters when solving difficult instances.
  - Linearization:
    - Recognition of more expressions than previous versions.
    - Advanced linearization of QP and conic models.
  - Nonlinear Solver Improvements:
    - Recognition of fractional linear programs and algorithmic improvements for this class.
  - Global Solver Improvements:
    - Introduce auxiliary variables to optimize in hyperspace for improved performance.
    - Improved warning messages for possible sub-optimal solutions.
    - Improved bound-tightening process.
- Added support for new intrinsic function lseMax.

MOSEK

- New libraries 10.0.25.
  - Improved presolve for conic problems.
  - Improved performance of interior-point optimizer for large-scale LPs and when running on recent AMD CPUs.
  - Improved cutting plane separation and presolve for mixed-integer programs.
    - Default of option MSK_IPAR_MIO_CUT_IMPLIED_BOUND changed from off to on.
    - New option MSK_IPAR_MIO_CUT_LIPRO to enable lift-and-project cuts.
    - New option MSK_IPAR_MIO_PRESOLVE_AGGREGATOR_USE to control use of aggregator.
  - Added symmetry detection and exploitation for mixed-integer programs. New option MSK_IPAR_MIO_SYMMETRY_LEVEL to control level of symmetry detection and handling.
  - Introduced reformulations for MIQCPs, which allows solution of more non-convex MIQCPs that can be reformulated into convex ones. New option MSK_IPAR_MIO_QCQO_REFORMULATION_METHOD to set reformulation method.
  - Added possibilities to enable memory saving and higher numerical emphasis for mixed-integer programs by use of new options MSK_IPAR_MIO_MEMORY_EMPHASIS_LEVEL and MSK_IPAR_MIO_NUMERICAL_EMPHASIS_LEVEL, respectively.
  - Added possibility to solve continuous problem obtained from fixing all integer variables to initial level values by enabling new option MSK_IPAR_MIO_CONSTRUCT_SOL.
  - Additional option changes:
    - New option MSK_DPAR_PRESOLVE_TOL_PRIMAL_INFEAS_PERTURBATION to specify amount by which Mosek can perturb problem in presolve to avoid infeasibility.
    - Removed option `MSK_IPAR_INTPNT_MULTI_THREAD`.
    - Removed moderate and aggressive scaling options from MSK_IPAR_INTPNT_SCALING and MSK_IPAR_SIM_SCALING.
    - New option MSK_IPAR_MIO_DATA_PERMUTATION_METHOD to enable permutation of mixed-integer problems.
    - New option MSK_IPAR_WRITE_JSON_INDENTATION to control whether json files are written with indentation.
    - Removed options `MSK_IPAR_WRITE_LP_QUOTED_NAMES`, `MSK_IPAR_WRITE_LP_STRICT_FORMAT`, `MSK_IPAR_WRITE_LP_TERMS_PER_LINE`, and `MSK_IPAR_WRITE_PRECISION`.

Octeract

- New libraries 4.5.1.

ODHCPLEX

- The time reported in model attribute resUsd now corresponds to the chosen clock type. That is, the CPU time spend by ODHCPLEX is reported if `clocktype` is changed to 1.

SCIP

- New libraries 8.0.2.

XA

- As announced, dropped XA.

XPRESS

- New libraries 40.01.03.

3.8.1.5 Tools

GAMS Studio

- New version 1.12.1.
  - New feature: Navigator (`Ctrl+k`)
    - Please note: this feature is currently in beta and does not have mouse support yet!
    - Filter files by typing a search term, wildcards are also supported (e.g.: `*.gms`).
    - Press `Enter` to jump to a selected file.
    - Type `?` to show help about all available prefixes.
    - The `GoTo`- and `TabBrowser`-dialogs are now obsolete and will be removed in the future.
  - GDX Viewer improvements:
    - New feature: Added first version of an export dialog, currently supporting Excel output only.
    - New feature: Added quick selection to GDX Viewer filters.
    - Improved auto-resizing behavior.
    - Improved symbol table colors especially when the theme has changed.
  - Search and Replace improvements:
    - Added highlighting to matches in `Search Result View`.
    - Added capture group support for regular expressions in `Search Dialog`.
  - Added filter to MIRO deploy dialog and EFI editor.
  - Added function to open directly in Pin-View from Project Explorer by `Ctrl+double-click` (for horizontal pin) or `Ctrl+Shift+double-click` (for vertical pin).
  - Added icons to the main tabs (except for macOS).

- Added editor settings to skip syntax highlighting for long lines (this can improve performance significantly).
- Stability improvements, bug fixes, and minor enhancements, e.g.:
  * Changed GAMS version mismatch error message to a warning message.
  * Fixed GDX Viewer crash when reloading a file with an alias selected.
  * Fixed crash when switching tabs on macOS.
  * Fixed project changes on search over multiple files where a file is linked to more than one project.
  * Fixed unexpected behavior when adding the main file to an EFI file.
  * Fixed wrong context menu for new entry in Project Explorer after using `Save as...`.
  * Fixed that models with names containing spaces failed to be executed on NEOS and Engine.

GDXRRW

- GDXRRW is now deprecated and will be removed in a future release. Please use GAMS Transfer R instead. If you encounter issues with the transition to GAMS Transfer R or if you have any feature requests for GAMS Transfer R or concerns about the transition, do not hesitate to contact GAMS support.

GDXXRW

- When the limit on number of supported of rows or columns is reached when reading from a file, a warning is now given that subsequent entries might have been skipped.

$libInclude win32

- Added `XLSMerge` functionality to <`sysdir`>\`inclib`\`win32.gms` to merge two Excel workbooks.

3.8.1.6 APIs

gams2numpy

- Non-UTF-8 characters do not result in an exception anymore. Instead, such characters are transferred to an alternative representation by using the Python error handler `backslashreplace`. This change affects `gams2numpy` and all its clients – `GAMS Transfer`, `Connect`, and `gams_magic` (Jupyter notebooks).
- The binaries are now build with `numpy 1.21.6` for Python 3.7 and `numpy 1.23.3` for Python 3.8, 3.9, and 3.10.

GAMS Transfer Matlab

- Breaking: Symbol name uniqueness is now checked case insensitively. For example, it is not possible anymore to have three different symbols named `symbol`, `Symbol`, and `SYMBOL`.
- Breaking: Changed `Symbol.addUELs` signature from `addUELs(dim, uels)` to `addUELs(uels, dim)`. `dim` is now allowed to accept a vector of dimensions.
- Breaking: Changed `Symbol.setUELs` signature from `setUELs(dim, uels)` to `setUELs(uels, dim)` and `setUELS(_, "rename", true/false)`. Setting `rename` to `true` triggers the old `Symbol.initUELs`. `dim` is now allowed to accept a vector of dimensions.
- Breaking: Changed `Symbol.removeUELs` signature from `removeUELs(dim, uels)` to `removeUELs()`, `removeUELs(uels)`, and `removeUELs(uels, dim)`. `dim` is now allowed to accept a vector of dimensions.
- Breaking: Changed `Symbol.renameUELs` signature from `renameUELs(dim, olduels, newuels)` to `renameUELs(uels)` and `renameUELs(uels, dim)`. `uels` can now be `cellstr`, `struct`, or `containers.Map`. `dim` is now allowed to accept a vector of dimensions.
- Added method `Container.getSymbolNames` to return the original symbol names for a list of symbol names of any case.
- Added method `Container.hasSymbols` to check if symbol name (case insensitive) exists.
- Added method `Container.getUELs` to get UELs from all symbols.
- Added method `Container.removeUELs` to remove UELs from all symbols.
- Added method `Container.renameUELs` to rename UELs in all symbols.
- Added method `Symbol.reorderUELs` to reorder UELs without changing the meaning of records.
- Added flags `Container.modified` and `Symbol.modified` to indicate if a container and/or symbol has been modified since last reset.
- Added possibility to filter UEL codes in `Symbol.getUELs`.
- Added possibility to pass a vector of dimensions to `Symbol.getUELs`.
- Added possibility to overwrite symbols with `Container.add*` if main symbol definition (e.g. type, domain) is equal.
- Changed `Container.getSymbols`, `Container.removeSymbol`, `Container.renameSymbol`, `Container.describe*`, and others that use `Container.getSymbols` to accept symbol names case insensitively.
- Changed behaviour of default records: Default records do not get written to GDX anymore if the records format is `dense_matrix` and either the container is in indexed mode or if the symbol has a regular domain.
- Changed behaviour of `getDomainViolations`: As in GDX, different character case does not lead to a domain violation.
- Categoricals for record domain labels are now created with `Ordinal` set to `true`, but ordinal categoricals are not enforced, i.e., users may pass categoricals with `Ordinal` set to `false`.
- Changed symbol read order when reading a subset of symbols: Symbol order is defined by source order (e.g., symbol order in GDX file) rather than user supplied order. To establish a custom order after the read, use `reorderSymbols`.
- Deprecated `Symbol.getUELLabels`. Use `Symbol.getUELs` instead.
- Deprecated `Symbol.initUELs`. Use `Symbol.setUELs` instead.
- Deprecated `Container.getUniverseSet`. Use `Container.getUELs` instead.

- Fixed failing symbol constructors when using `domain_forwarding`, but none of the optional arguments.
- Fixed `Symbol.transformRecords` (table-like to matrix-like formats) in case the domain set records and UELs differ.
- Fixed write of sets defined over sets.

GAMS Transfer Python

- The API has been redesigned to more closely mimic GAMS behavior (i.e., GAMS is case preserving, not case sensitive).
- The `.data` dict has been replaced with a custom `dict` that preserves the case of symbol names. This mimics the convention in GAMS. Symbol lookup from this custom `dict` is case insensitive.
- Added new methods to customize dimension, symbol, and container UELs: `getUELs`, `setUELs`, `renameUELs`, `removeUELs`, `addUELs`, and `reorderUELs`
- New Symbol method `equals` can be used to compare symbol objects.
- Deprecated the container method `getUniverseSet`. This method will be removed in a future major release. Use `<Container>.getUELs()` instead.
- Container methods `addSet`, `addAlias`, `addParameter`, `addVariable`, and `addEquation` now allow for symbol overwriting. Previously, an error was raised and the symbol had to be removed before recreation.
- New class `UniverseAlias` and new Container method `addUniverseAlias` to allows creation of a symbol which is an alias to the universe (fixes bug).
- Container no longer link symbol categoricals together when setting records with `setRecords`; data that results in domain violations is not immediately lost and the user has greater flexibility to modify inplace to the `Container`.
- Updated behavior of `removeSymbols`: the domain of dependent symbols is relaxed to "$*$" if a set/alias is removed. If a parent set of an alias is removed, the alias symbol will also be removed automatically.
- Implemented a new class called `DomainViolation` to organize where domain violations occur in data.
- New method `getDomainViolations` to return a list of `DomainViolation` objects, should they exist.
- Methods $*$`DomainViolations` locate domain violations by referencing all appropriate domain sets. `findDomainViolations` now returns a view of the `records` DataFrame that contains all domain violations (previously returned only an `Index` where domain violations occurred).
- Methods $*$`DuplicateRecords` locate duplicate records on a case insensitive basis (consistent with GAMS behavior).
- Removed tests for duplicate records and domain violations from the `<Symbol>.isValid()` method. `<Symbol>.isValid()` validates the structural aspects of the data, not the quality of the data. This improves performance by not looping over all records.
- Created a Symbol `modified` flag that tracks if any changes have been made to a symbol. These flags can be set/reset by setting `<Container>.modified=True/False`.
- Enabled the use of the Python `in` operator to test if a symbol is `in` a container with the syntax `<symbol_object> in <container_object>` or `<symbol_string_name> in <container_object>`, where `<symbol_string_name>` is case insensitive.

- The Container is now subscriptable, i.e., `m[<symbol_name>]` returns the symbol object. Symbol objects are still accessible through the underlying dictionary, i.e., `m.data[<symbol_name>]`.
- `setRecords` now maintains user specified categoricals when passing a DataFrame.
- `setRecords`, `getUELs`, `setUELs`, `renameUELs`, and `addUELs` methods now strip any trailing whitespace from UELs.
- Improved Container behavior if reading from `ConstContainer`/`Container`: lists passed to the symbols argument `read(<source>, symbols)` are no longer considered to be ordered – previously, if an alias was listed before its parent set, an exception would occur.
- Fixed bug when attempting to read in a symbol named `all`, but the Container read in all symbols.
- Exposed (previously hidden) `ConstContainer` methods `getCardinality()` and `getSparsity()`.
- An exception is now raised if attempting to read a symbol that does not exist in the data source. Previously, such symbol names were silently ignored.
- Improved clarity of some exception messages and other error handling.
- Fixed bug that prevented `dict_keys` and `dict_items` (returns from dict methods `.keys()` and `.items()`, respectively) objects from being passed directly into `setRecords`.
- Fixed bug that resulted in a `NameError` when calling `.whereMin()` or `.whereMax()` on variables or equations.
- New tests within `<Container>.isValid()` method to detect broken container references in symbols and inconsistent symbol naming between the `<Container>.data` dict and the symbol objects. These issues can occur when creating new symbols in a container with a copy/deepcopy operation.
- Restructured `ConstContainer` symbol classes to improve read-only behavior.

GAMS Transfer R

- Breaking: Symbol name uniqueness is now checked case insensitively. For example, it is not possible anymore to have three different symbols named `symbol`, `Symbol`, and `SYMBOL`.
- Breaking: Symbol records and symbol names are treated in a case insensitive manner. Symbols domain is not checked automatically.
- Added a new class `ConstContainer` for efficient data transfer.
- Allow read from `Container`/`ConstContainer` into another `Container`.
- Added a method `Container$getSymbolNames` to return the original symbol names for a list of symbol names of any case.
- Added a method `Container$hasSymbols` to check if symbol name (case insensitive) exists.
- Changed `Container$getSymbols`, `Container$removeSymbols`, `Container$renameSymbol`, `Container$describe*`, and others that use `Container$getSymbols` to accept symbol names case insensitively.
- New Symbol methods `hasDuplicateRecords`, `countDuplicateRecords`, `findDuplicateRecords`, and `dropDuplicateRecords` to help debug and resolve errors with duplicate records.
- New Container methods `hasDuplicateRecords`, `countDuplicateRecords`, and `dropDuplicateRecords` to help find symbols that contain duplicate records that cause the Symbol to be invalid.
- New tests within `Container$isValid()` method to detect broken container references in symbols and inconsistent symbol naming between the `<Container>$data` field and the symbol objects.
- Method `isValid` does not check for symbol record domain columns being factors, for duplicates, and for domain violations.

- New Symbol methods `hasDomainViolations`, `countDomainViolations`, `findDomainViolations`, `dropDomainViolations`, and `getDomainViolations`.
- New Container methods `hasDomainViolations`, `countDomainViolations`, and `dropDomainViolations`.
- Allowed symbol overwriting with `addSet/addParameter/addVariable/addEquation/addAlias` methods when everything other than records and description is unchanged.
- Added methods `SpecialValues$isNA`, `SpecialValues$isEps`, `SpecialValues$isUndef`, `SpecialValues$isPosInf`, `SpcialValues$isNegInf` to test for special values.
- The argument for `where*` methods changed from `columns` to `column`.
- Trailing whitespaces is now removed automatically from UELs in `setRecords` and `*UEL` methods.

GMO

- Added a concurrent mode for QP extraction. In this mode, independent extraction methods are run in parallel threads. Extraction terminates when the first method completes. This is done row-wise: the concurrent threads work on each row in turn, using the result of the first thread to finish for that row and moving on together to the next row. Setting the `gmoQExtractAlg` property to 3 selects this method.
- Added new function `gmoGetQMakerStats` to get statistics about QP extraction.
- Added new function `gmoFillMatches` to complete the row/column matching for MCP models.
- Added new function `gmoLoadDataLegacyEx` to API: this function differs from the existing `gmoLoadDataLegacy` function in that the new function includes a flag `fillMatches` to toggle the completion of matching information for MCP models during the load.
- Changed the behavior of `gmoLoadDataLegacy` vis-a-vis matching information for MCP. Previously, `gmoLoadDataLegacy` behaved like `gmoLoadDataLegacyEx(fillMatches=true)`. With this release, it behaves like `gmoLoadDataLegacyEx(fillMatches=false)`. Users of this function that require the old behavior must switch to calling `gmoLoadDataLegacyEx(fillMatches=true)` or call `gmoFillMatches` explicitly.

Jupyter Notebooks

- The data synchronization between Python and GAMS relies now entirely on GAMS Transfer.
- *Environments* allow working with different GAMS instances.
- Instead of `gams_push` and `gams_pull` with various Python data formats, GAMS lines and cells automatically synchronize symbols between GAMS and Python.
- GAMS Symbols can now be declared in the Transfer container in Python rather than in the GAMS cells. Please inspect the Magic (Jupyter Notebooks) Magic "GAMS Jupyter API description" for details.
- The older version of GAMS magic commands is still available under `apifiles/Python/gams/gams_magic_legacy`. This will be removed with one of the next major versions.

Python

- Added example `transport_gt.py` that demonstrates how to combine the GAMS Python high level API with a GAMS Transfer Container, essentially replacing the `GAMSDatabase` class.
- Added method `run_engine` to `GAMSJob` class to run jobs on GAMS Engine, rather than locally.

3.8.1.7 Model Libraries

GAMS Data Library

- Added **connect05.gms : Simple Connect Example for Excel**.

GAMS Model Library

- Dropped `licememo`.

GAMS Test Library

- Added **fnlse.gms : Rough correctness test for LSE max/min intrinsics**.
- Added **fnlsemax.gms : Test correctness of LSEMax intrinsic**.
- Added **fnlsemaxsc.gms : Test correctness of LSEMaxSc intrinsic**.
- Added **fnlsemin.gms : Test correctness of LSEMin intrinsic**.
- Added **fnlseminsc.gms : Test correctness of LSEMinSc intrinsic**.
- Added **fnlseslv.gms : Rough solver correctness test for LSEMax intrinsics**.
- Added **caconcat.gms : Test Connect agent Concatenate**.
- Added **suffix02.gms : Test CLP suffixAlgebraVars**.

FIN Library

- Fixed **ThreeStageSPDA : A three stage stochastic programming model for SPDA**.

PSOPT Library

- Fixed **DED : Dynamic Economic Load Dispatch**.
- Fixed **DED-PB : Price based Dynamic Economic Load Dispatch**.
- Fixed **DED-wind : Dynamic Economic Load Dispatch considering Wind generation**.
- Fixed **DEDESS : Cost based Dynamic Economic Dispatch integrated with Energy Storage**.
- Fixed **DEDESSwind : Cost based Dynamic Economic Dispatch integrated with Energy Storage and Wind**.
- Fixed **ESSDCOPFwind : DC-OPF integrated with Energy Storage and Wind**.
- Fixed **MultiperiodDCOPF24bus : Multi-period DC-OPF for IEEE 24-bus network considering wind and load shedding**.
- Fixed **PBUC : Price based Unit commitment**.

3.8.1.8 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.11	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
COPT 5.0	✓	✓	✓
CPLEX 22.1	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GUROBI 9.5	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.14	✓	✓	✓
HiGHS 1.2	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO 13.1	✓	✓	✓
LINDO 14.0	✓	✓	✓
LINDOGLOBAL 14.0	✓	✓	✓
MILES	✓	✓	✓
MINOS 5.6	✓	✓	✓
MOSEK 10	✓	✓	✓
NLPEC	✓	✓	✓
OCTERACT 4	✓	✓	
ODHCPLEX 6	✓	✓	
PATH	✓	✓	✓
QUADMINOS 5.6	✓	✓	✓
SBB	✓	✓	✓
SCIP 8.0	✓	✓	✓
SHOT 1.1	✓	✓	✓
SNOPT 7.7	✓	✓	✓
SOPLEX 6.0	✓	✓	✓
XPRESS 40.01	✓	✓	✓

3.8.2 41.2.0 Minor release (November 14, 2022)

3.8.2.1 Acknowledgments

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

3.8.2.2 GAMS System

GAMS

- Improved the dollar control options $abort[.noError], $error, and $log, so that the text given as an optional argument is not limited to 255 characters anymore.
- Fixed the creation of a potentially wrong dump file with dumpOptGDX.

3.8.2.3 APIs

GAMS Transfer Python

- Enabled writing of partial `Containers` to GDX with `write_symbols` argument.
- Changed `Container.write()` argument `write_symbols` to `symbols` to better harmonize with Matlab and R versions of GAMS Transfer.
- Fixed bug with `equals` symbol method when comparing `UniverseAliases`.

Jupyter Notebooks

- Fixed a bug in `%gams` line magic that prevented code containing dashes preceded by a space from being executed properly.

3.8.3 41.3.0 Minor release (November 28, 2022)

3.8.3.1 Acknowledgments

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

3.8.3.2 GAMS System

GAMS

- Improved performance when using the command line parameter GDX for programs with many labels.
- Fixed a potential problem using a platform specific license with the pLicense parameter.

3.8.3.3 Solvers

COPT

- Fixed that option AbsGap was not available.

Lindo/LindoGlobal

- New libraries 14.0.191.

3.8.3.4 Tools

GAMS Studio

- New version 1.12.2 with some bug fixes and minor enhancements, for example:
  - Added support for trailing `:NUMBER` when choosing files in Navigator.
  - Fixed crash when moving line in text file.
  - Fixed crash when reloading an invalid GDX file.

3.8.3.5 APIs

GMO

- Fixed function `gmoNameInput`: in case old control files are read, the result string was uninitialized.

3.8.4 41.4.0 Minor release (December 14, 2022)

3.8.4.1 Acknowledgments

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

3.8.4.2 Solvers

GUSS

- Fixed a bug that could have caused unwanted skipping of scenarios when the scenario data is sparse.

MOSEK

- New libraries 10.0.31.

SCIP Optimization Suite

- New libraries SCIP 8.0.3.
- New libraries SoPlex 6.0.3.
- New libraries PaPILO 2.1.2.

3.8.4.3 Tools

GAMS Studio

- New version 1.12.3 with some bug fixes and minor enhancements, for example:
  - Added `x` prefix to Navigator to access Quick Actions.
  - Unified Navigator input field with other search fields, adding regex and exact match modifiers.
  - Fixed behavior of `Squeeze Trailing Zeroes` check box in GDX Viewer when using `g-format` or `e-format` with `Full` precision.
  - Fixed help view being empty on some Linux distributions.

3.8.4.4 APIs

GAMS Transfer Python

- Fixed `setRecords` throwing an exception in case of labels that differ in trailing spaces only.
- Fixed a bug when reading data from multiple sources with a single `Container` instance.
- Fixed a bug when writing a GDX file with a symbol list.
- Fixed a bug that prevented writing when the first symbol in the `Container` is a `UniverseAlias`.

Jupyter Notebooks

- Utilize partial GDX writing of GAMS Transfer containers in GAMS magic. Only symbols that are new or modified will be written to GDX. This improves the data exchange performance between Python and GAMS.

3.8.5 41.5.0 Minor release (January 03, 2023)

3.8.5.1 Acknowledgments

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

3.8.5.2 Solvers

CONVERT

- Fixed error messages to be thrown incorrectly when using external functions (files were written correctly).

GUSS

- Fixed a problem with MCP models and variable fixings as scenario data.

Mosek

- New libraries 10.0.33.

3.8.5.3 APIs

gams2numpy

- Fixed a memory leak in `gdxWriteSymbol[Raw|Str]` and `gmdFillSymbol[Raw|Str]`.
- Fixed a problem with uel and string numbers that do not point to a label or element text.
- Improve performance converting GDX/GMD strings to Python strings.
- Allow to specify an optional encoding string for methods `[gdx|gmd]GetUelList` and `[gdx|gmd]ReadSymbol[Raw|Str]`.

3.9 40 Distribution

3.9.1 40.1.0 Major release (August 01, 2022)

3.9.1.1 Acknowledgments

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

3.9.1.2 GAMS System

GAMS

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

- Do not allow to load a symbol implicitly from $offEmbeddedCode under $onExternalInput, when it was not declared as `external input` symbol (the same check was so far done for explicit loading only).
- Changed the interaction of external input and $gdxIn to allow the continued use of a `$gdxIn` file over multiple external input sections. So, the following code works now, but threw an error in the past:

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

* Do something but not opening another GDX file

$onExternalInput
Set j(*) markets ;
$load j
$offExternalInput
```

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

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

- Also changed the behavior when encountering a symbol with unknown dimension with a compile time embedded code statement. In the past, this resulted in an error, now the dimension of that symbol is set to the dimension defined in the embedded code block.
- Make sure that we do not change the loop control index by implicit loading from endEmbeddedCode (the same check was so far done for explicit loading only).
- Added an explicit error check to ensure that the command line parameter restartNamed is used together with the command line parameter restart only.
- Changed the put utility facility to ignore the extra double quotes added when the print control is set to `4`, `5`, or `6`. In the past that lead to an error since the commands were not recognized when they are quoted.
- We plan to remove language elements to access some commercial and licensing information with the next major release:
  - System suffixes: `components`, `clipCodes`, `clipComponentMap`, `componentSolverMap`, `gamsLicenses`, `gamsLicenseTypes`, `lice1`, `lice2`, `licenseDateEvalN`, `licenseDateEvalS`, `licenseDateMaintN`, `licenseDateMaintS`, `licenseDateN`, `licenseDateS`, `licenseDateSysN`, `licenseDateSysS`, `licenseDaysEval`, `licenseDaysMaint`, `licenseDC`, `licenseEval`, `licenseID`, `licenseInstitution`, `licenseLevel`, `licenseLevelText`, `licenseLicensee`, `licenseMudText`, `licensePlatform`, `licensePlatformText`, `licenseStatus`, `licenseStatusText`, `licenseType`, `licenseVendor`, and `vendors`.
  - Predefined sets `componentEDate`, `componentMDate`, and `componentLicenseOptions`.
  - Many of these are used in the GAMS Model Library model `licememo`, which we also plan to remove.
- Fixed that exceeding the internal limit of 2.1 billion instructions did not raise an error.

- Fixed a rare problem which stopped GAMS from terminating if a solver invoked in an asynchronous thread did not return.
- Fixed an error causing missing domain information, when implicit sets are used while creating a dump file.

GMSPython

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

Connect

- Domain information is copied into the Connect database when using agents GAMSReader and GDXReader. Agent GDXWriter exports the domain information to the GDX file.
- Added attribute `duplicateRecords` to GAMSWriter and GDXWriter to control the way to deal with duplicate records in a Connect container symbol when writing to GAMS or GDX. Note that both agents currently deal with duplicate records in a case sensitive way.
- Added new DomainWriter agent to modify domain information of Connect symbols.
- Added new PandasSQLReader and PandasSQLWriter agents to read from and write to SQL databases.
- Updated Projection agent:
  - Changed boolean attribute `dropText` to a string attribute text.
  - Added boolean attribute asParameter to turn variables into parameters with an extra index to store the suffixes.
  - Allow a list of symbol names for attribute name to combine multiple scalar symbols into a single one-dimensional symbol.
- Symbol names in Connect are now case sensitive by default. So casing in the YAML file is important and needs to match e.g. when reading from GDX or GAMS (which use case in-sensitive symbol name lookup) or across agents. There is now a global attribute caseSensitiveLookup in agent Options to control the case-(in)sensitive symbol lookup.
- Added new attribute `debug` in agent Options to provide traceback information in case of an exception.
- Dropped support for writing variables and equations with CSVWriter and PandasExcelWriter. Variables and equations can be turned into parameters using the Projection agent.
- Added attribute `trace` to GAMSReader, GAMSWriter, GDXReader, and GDXWriter.
- Fixed bug where PandasExcelWriter failed when writing no data.

Embedded Python Code Facility

- Allow to specify source code encoding by providing a comment of format `# coding=<encoding name>` as first line in an embedded Python code section.
- If `gams._debug` is set to an integer greater than 0 a traceback is printed to `stdout` in case of an exception.

Installer

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

3.9.1.3 Solvers

BARON

- New libraries 22.7.23.
  - Improved presolve.

Bonmin and BonminH

- As announced, the solvers Bonmin and BonminH have been removed. For the time being, libraries of GAMS/Bonmin are available at the `COIN-OR GAMSlinks project`. Note, that GAMS does not test or offer support for these libraries.

CBC, SCIP, SHOT

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

CONVERT

- Export prettier formulas (`x` for `x**1`, `power(x,1)`, `vcpower(x,1)` or `rpower(x,1)`).

COPT

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

GLOMIQO

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

GUROBI

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

HiGHS

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

Ipopt

- New libraries Ipopt 3.14.9.
  - Added options ma27_print_level, ma57_print_level, and mumps_print_level.
  - Fixed that variable names for variable bounds in detailed log output were wrong for problems with fixed variables.
- New libraries Mumps 5.5.0.

KNITRO

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

LGO

- As announced, dropped LGO.

MPSGE

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

MSNLP

- As announced, dropped MSNLP.

Octeract

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

SCIP

- Added support for logical functions `bool_and`, `bool_eqv`, `bool_imp`, `bool_not`, `bool_or`, `bool_xor`. Only constants, binary variable, and logical functions are allowed as arguments for these operators. The GAMS/SCIP link introduces additional variables and constraints of types and, or, and xor to handle these functions.
- New libraries SCIP 8.0 (504f5f2749).
  - Improved and extended cut selection statistics.
  - Added options presolving/milp/maxbadgesizepar and presolving/milp/maxbadgesizeseq.
- New libraries PaPILO 2.1 (79da073).

XPRESS

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

3.9.1.4 Tools

GAMS IDE

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

GAMS Studio

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

GDX2ACCESS

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

GDX2SQLITE

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

GDXVIEWER

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

MDB2GMS

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

MPS2GMS

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

SQL2GMS

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

3.9.1.5 APIs

GAMS Transfer Python

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

GAMS Transfer R

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

Python

- As announced, dropped support for Python 3.6.

3.9.1.6 Model Libraries

GAMS Data Library

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

GAMS Test Library

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

3.9.1.7 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.11	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
COPT 5.0	✓	✓	✓
CPLEX 22.1	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GUROBI 9.5	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.14	✓	✓	✓
HiGHS 1.2	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO 13.1	✓	✓	✓
LINDO 13.0	✓	✓	✓
LINDOGLOBAL 13.0	✓	✓	✓
MILES	✓	✓	✓
MINOS 5.6	✓	✓	✓
MOSEK 9	✓	✓	✓
NLPEC	✓	✓	✓
OCTERACT 4	✓	✓	
ODHCPLEX 6	✓	✓	
PATH	✓	✓	✓

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
QUADMINOS 5.6	✓	✓	✓
SBB	✓	✓	✓
SCIP 8.0	✓	✓	✓
SHOT 1.1	✓	✓	✓
SNOPT 7.7	✓	✓	✓
SOPLEX 6.0	✓	✓	✓
XA	✓	✓	
XPRESS 40.01	✓	✓	✓

3.9.2 40.1.1 Maintenance release (August 16, 2022)

3.9.2.1 Acknowledgments

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

3.9.2.2 GAMS System

GAMS

- Fixed a bug that cause restart files containing sums that add set elements to execute incorrectly. Restart files created with GAMS 40.1.0 carry this bug and should not be used but recreated.

3.9.2.3 Solvers

COPT

- New libraries 5.0.3.

MOSEK

- New libraries 9.3.21.

OCTERACT

- The options documentation provided on the Octeract webpage was apparently not uptodate when their last releases were made. Therefore, for the GAMS/Octeract link, the available options and their documentation also did not match the solvers libraries. We have updated these now and provide the list of changes to available options here:
  - Changed default for `BOUND_VIOLATION_TOLERANCE` from 1e-8 to 1e-6.
  - Added values `MOST_FRACTIONAL_VARIABLE` and `MODIFIED_MOST_NONCONVEX_VARIABLE` for option `BRANCHING_STRATEGY`.
  - New option `STRONG_BRANCHING_DEPTH`.
  - Removed `CBC_LB_MAX_NODES`. Added option `MILP_LB_MAX_NODES`.

  - Removed CBC_PRESOLVE.
  - Added values GUROBI and XPRESS for MIP_SOLVER (see also docu).
  - New options HEUR_INEQUALITY and HEUR_SUPREME.
  - Changed default for HEUR_NL_FEASIBILITY_PUMP from 0 to 1.
  - Replaced IS_PURGE_CUTS_ENABLED by new option PURGE_CUTS.
  - Added option LOCAL_SEARCH.
  - New option PRESOLVE.
  - New options REDUCE_LINEAR_CONSTRAINTS and STRENGTHEN_LINEAR_CONSTRAINTS.
  - Changed type of option REFORMULATE_INTEGERS_IN_MIQCQP to boolean.
  - New option USE_NONLINEAR_RELAXATION.
  - Changed default for OUTPUT_FREQUENCY from ∞ to 1.
- Fixed default for CONVERGENCE_TOLERANCE: Since Octeract uses this value as both absolute and relative gap tolerance, the default is now min(GAMS optcr, GAMS optca) to avoid too early termination on instances with objective value in (-1,1).
- Added workaround for wrong dual bound reported by Octeract if problem is reformulated into MILP and the MILP solver does not solve to optimality (e.g., stops due to a timelimit). This resulted in wrongly declaring solutions as global optimal.

SCIP

- New libraries 8.0 (0e5ff0a6b5).
  - Added options estimation/showstats, heuristics/alns/shownbstats, misc/showdivingstats to reenable some statistics that are now disabled.

3.9.3 40.2.0 Minor release (September 01, 2022)

3.9.3.1 Acknowledgments

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

3.9.3.2 GAMS System

GAMS

- Fixed the creation of a potentially wrong dump file with dumpOptGDX.

3.9.3.3 Solvers

COPT

- New libraries 5.0.4.

IPOPT

- New libraries MUMPS 5.5.1.

SCIP

- New libraries 8.0 (f0a1490e3b).
- Added parameter gams/infbound to instruct the GAMS/SCIP link to replace all missing variable bounds by a given value (off by default).

SHOT

- Fixed use of Gurobi for solving relaxations.

3.9.3.4 APIs

Python

- Fixed `setup.py` not copying schema files for Connect agents.

GAMS Transfer Matlab

- New version 0.2.2:
  - Fixed read of equation with unknown subtype (recast as `=e=`).
  - Fixed partial read of symbols in indexed container.
  - Fixed `equals` method in indexed container.
  - Fixed possible segfault when reading a subset of symbols as dense matrix with at least one scalar symbol.

3.9.4 40.3.0 Minor release (September 16, 2022)

3.9.4.1 Acknowledgments

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

3.9.4.2 GAMS System

- Fixed an error with command line parameter dumpOpt, that ignored the `SemiInt` modifier for a variable.

3.9.4.3 Solvers

CPLEX

- Fixed reporting of final (fixed) solve objective value instead of MIP solve objective value if final solve is enabled.

3.9.4.4 Tools

MODEL2TEX

- Added support for more (singleton) set attributes.

GAMS Studio

- New version 1.11.2 with some bug fixes, for example:
  - Updated tooltip of a tab after filename changed with "save as...".
  - Removed link to outdated introduction video.
  - Fixed occasional crash when opening search dialog after closing all files.
  - Fixed drag'n'drop creating group with invalid working directory.
  - Fixed GDX Viewer showing wrong data in table view because of a falsely restored symbol which dimension or type has changed.
  - Fixed GDX Viewer crash occurring when a file is reloaded during a drag'n'drop operation in table view mode.
  - Fixed disabled "Create" button in MIRO dialog on missing assembly file.

3.9.5 40.4.0 Minor release (October 03, 2022)

3.9.5.1 Acknowledgments

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

3.9.5.2 GAMS System

GAMS

- Fixed wrong result for set attributes `.tlen` and `.tval` when used with Singleton Sets.
- Fixed an unexpected compilation error, when an embedded code section starts right after an `else` statement.

3.9.5.3 Solvers

CONVERT

- Fix model statement in GAMS format for MCP models with incomplete matching information.

3.9.5.4 Tools

GAMS Studio

- New version 1.11.3 with some bug fixes and minor enhancements, for example:
  - Added shortcut `Ctrl + G` to also close the `GoTo` dialog.
  - Added `Ctrl + C` shortcut to system log.
  - Added `Clear Log` action to the system log context menu.
  - Added shortcut descriptions for various actions in context menu of editor.
  - Fixed drag'n'drop not working in project explorer on macOS.
  - Fixed wrong dialog title for "New Project".

3.10 39 Distribution

3.10.1 39.1.0 Major release (May 03, 2022)

3.10.1.1 Acknowledgments

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

3.10.1.2 GAMS System

GAMS

- Added basic model statistics (range of absolute non-zero values for the bounds, right hand sides and matrix coefficients) to the output file, and log.
- Added model attributes to access the values of the basic model statistics also programmatically: rngBndMax, rngBndMin, rngMatMax, rngMatMin, rngRhsMax, rngRhsMin.
- Added new command line parameter multi to initialize the state of $on/offMuli and $on/offMultiR.
- Extended the syntax of $(on|off)Echo[S|V], $(on|off)Put[S|V], $(on|off)EmbeddedCode[S|V], and (end|pause)EmbeddedCode[S|V] with an optional suffix to allow nested use. Note that this might trigger an error for code that was OK before. In the following example, the suffix at `$onEcho` was ignored in the past. Now, this will result in an error, since there is no `$offEcho` with matching suffix:

```
$onEcho.abc > x.txt
abc
$offEcho
```

- In embedded code Python, the Python interpreter stays alive even when the embedded code section ends with $offEmbeddedCode and endEmbeddedCode. So one has access to Python objects across embedded Python code sections. Nevertheless, the `gams` object gets newly initialized (with the exception of continueEmbeddedCode). Hence old references to `gams` or derived objects, e.g. `sym = gams.get("sym")` still exist in Python but the connected external resources are no longer available. Accessing these objects results in a crash as demonstrated in the following example:

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

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

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

- Added new command line parameter and option ECImplicitLoad, as well as related dollar control options $on/offECImplicitLoad.
- Added new command line parameters ConnectIn and ConnectOut to instruct GAMS to pass the Connect YAML files provided by these option to the Connect framework at the beginning of a GAMS job (`ConnectIn`) and at the end (`ConnectOut`) of the GAMS job.
- Added new put_utility command `ECArguments` to extend the embedded code *arguments* with a dynamic text.
- Added explicit range checking for some Put File Attributes. In the past, unexpected values for some attributes (e.g. `file.nr = 8;`) could have led to unwanted output. Now, unacceptable values get projected to the nearest acceptable value.
- Updated use of decimals (file.nd) value in floating-point put output. Previously, the decimals value was treated as a specification, with putfile error messages resulting if the output contained fewer decimals. We treat the decimals value as a limit now: it's not an error if fewer digits are shown because of the width limit or for other reasons.
- Removed the GDX file `gmspfmap.gdx` that contained some partial mapping of files to components from the system directory.
- There was a hidden option `work`. This cannot be used anymore. The command line parameter or model attribute workSpace should be used instead.
- Allow holdFixed=1 for EMP, but not EMPSP models. The partly reverts the change of holdFixed for EMP models from GAMS 38.
- Fixed a bug where the command line parameters holdFixed, nodLim, workFactor and workSpace took precedence over the related model attributes.
- Fixed an error with constant evaluation in data statements for scalar symbols in a parameter statement.
- Fixed output of very large negative numbers (e.g. `-DBL_MAX`) to trace files.
- Fixed minor rounding and display issues with double-precision numbers. This is relevant to values displayed in the listing file, in put files, and less commonly in the GAMS log. Ties are now resolved using the **round-to-even** rule instead of **round-towards-zero**. This new tie-breaking rule is consistent with the display routines in C++, Python, Julia, and R. For example:
  - with `option decimals=5`, `scalar p / 9999998.4 /` was displayed as `1.000000E+7` – now as `9.999998E+6`

  - with `option dispwidth=9, decimals=5, scalar p / -8.123441235 /` was displayed as `-0.8123E+1` – now as `-8.1234412`
  - very small values (`1e-250` or less) were displayed as 0 but are now displayed using E-format in the usual way
  - displaying 1.5 or 2.5 with zero decimals both now show 2, while previously 1.5 would display as 1
  - similarly, 1.75 with `decimals=1` now displays as 1.8 instead of 1.7 as it did previously

Connect

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

Embedded Code

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

Embedded Connect Code Facility

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

Embedded Python Code Facility

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

Extrinsic Function libraries

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

Windows Installer

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

3.10.1.3 Solvers

CONOPT

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

COPT

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

CPLEX

- New libraries 22.1.0.
  - New options:
    - lowerobjstop: in a minimization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value lower than or equal to the specified value.
    - upperobjstop: in a maximization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value greater than or equal to the specified value.
  - Removed CPLEX Remote Object Server and Distributed MIP.

  - Removed options `computeserver`, `polishtime`, `rampupdettimelimit`, `rampupduration` and `rampuptimelimit`.

- New option exactkappa: Report exact condition number in quality report. Note: Previously, the exact condition number has always been reported. Now, default is to not do it due to possibly high computation times.
- Add previously hidden options:
  - iafile: secondary option file to be read in interactive mode triggered by iatriggerfile.
  - iatriggerfile: file that triggers the reading of a secondary option file in interactive mode.
  - iatriggertime: time interval in seconds the link looks for the trigger file in interactive mode.
  - ltol: basis identification primal tolerance.
  - mtol: basis identification dual tolerance.
  - readparams: read CPLEX parameter file.

DE

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

EMPSP

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

GLOMIQO

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

GUSS/ScenarioSolver

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

Ipopt

- New libraries 3.14.6.

KNITRO

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

LGO

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

Lindo/LindoGlobal

- New libraries 13.0.340.

MOSEK

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

MSNLP

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

Octeract

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

ODHCPLEX

- New libraries 6.11.

SCIP Optimization Suite

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

XA

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

XPRESS

- New libraries 39.01.06.

3.10.1.4 Tools

GAMS Studio

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

MPS2GMS

- The tool has been reimplemented and now uses the readers of the `HiGHS solver`.
- If the MPS file appears to be in fixed form (row or character names with spaces and at most 8 characters), then parsing in fixed form is attempted.
- GDX and GMS files are now also written if no GDX and GMS filenames are provided explicitly.
- Dropped support for writing an MPS file.
- Dropped support for `OBJNAME` section.

- Sections SOS and SETS are now assumed to follow the format used by CPLEX, GUROBI, and FICO XPRESS.
- Dropped parameters NAME, N, RHS, RANGES, BOUNDS, MPSOut, Trace.
- Added parameter COLUMNINTVARSAREBINARY to specify how to handle integer variables that appear first in COLUMNS section.
- Quadratic coefficient matrices are no longer written in symmetric form. Zero coefficients (eps) that were written to GDX in certain situations are now omitted.
- See also the MPS2GMS documentation.

3.10.1.5 APIs

GAMS Transfer Matlab

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

GAMS Transfer Python

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

GMD

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

GMO

- Added properties gmoHessNZ64, gmoHessMaxNZ64, and gmoHessLagNZ64 and functions gmoHessStruct64 and gmoHessValue64 to get Hessian element counts as 64-bit integer.
- Added properties gmoQNZ64 and gmoGNLNZ64: all properties returning 32-bit Jacobian nonzero counts now have 64-bit counterparts.
- Fixed a memory leak in the Hessian evaluation: the leak occurred during speculative Hessian evaluations, i.e., evaluations with a tight memory limit intended to see how memory-intensive Hessian evaluation is and if Hessians should be used by the solver.

High-level APIs

- `GamsModelInstance.solve()` falls back to a warm start in case a hot start fails.
- Added read property for `GAMSSet.SetType` (.NET, Python) and `GAMSSet.getSetType` (Java) to distinguish `multi` and `singleton` sets.
- New optional argument `setType=SetType.multi` in `GAMSDatabase.AddSet()` (.NET), `GamsDatabase.add_set()` and `GamsDatabase.add_set_dc()` (Python), and `GAMSDatabase.addSet` (Java) to set type of set.
- Fixed defined enumerated value for options **FDOpt** and **DumpOpt** in Java interface.

3.10.1.6 Model Libraries

GAMS Model Library

- Dropped `deploy`.
- Dropped `dplytst`.
- Updated **cta.gms : Controlled Tabular Adjustments**.
- Updated **dicegrid.gms : MIP Decomposition and Parallel Grid Submission - DICE Example**.
- Updated **herves.gms : Herves (Transposable Element) Activity Calculations**.
- Updated **qmeanvar.gms : Financial Optimization: Risk Management using MIQCP**.
- Updated **sddp.gms : Multi-stage Stochastic Water Reservoir Model solved with SDDP**.
- Updated **seders.gms : ERS Data Manipulations with SED**.
- Updated **tsp5.gms : TSP solution with Miller et al subtour elimination**.
- Added **asyncincbi.gms : Asynchronous processing of incumbents reported by GAMS/CPLEX**.
- Added **springchain.gms : Equilibrium of System with Piecewise Linear Springs**.
- Added **trussm.gms : Truss Toplogy Design with Multiple Loads**.

GAMS Data Library

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

GAMS Test Library

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

FIN Library

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

PSOPT Library

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

3.10.1.7 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.11	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
BONMIN 1.8	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
COPT 4.0	✓	✓	✓
CPLEX 22.1	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓
GUROBI 9.5	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.14	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO 13.0	✓	✓	✓
LGO	✓	✓	✓
LINDO 13.0	✓	✓	✓
LINDOGLOBAL 13.0	✓	✓	✓
MILES	✓	✓	✓
MINOS 5.6	✓	✓	✓
MOSEK 9	✓	✓	✓
MSNLP	✓	✓	✓

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
NLPEC	✓	✓	✓
OCTERACT 4	✓	✓	
ODHCPLEX 6	✓	✓	
PATH	✓	✓	✓
QUADMINOS 5.6	✓	✓	✓
SBB	✓	✓	✓
SCIP 8.0	✓	✓	✓
SHOT 1.1	✓	✓	✓
SNOPT 7.7	✓	✓	✓
SOPLEX 6.0	✓	✓	✓
XA	✓	✓	
XPRESS 39.01	✓	✓	✓

3.10.2 39.1.1 Maintenance release (May 18, 2022)

3.10.2.1 Acknowledgments

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

3.10.2.2 Solvers

CPLEX

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

KNITRO

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

3.10.2.3 Tools

GAMSCONNECT

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

GAMS Studio

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

GDXDIFF

- Fixed wrong differences of parameters with option `fldOnly`.

MODEL2TEX

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

3.10.3 39.2.0 Minor release (June 02, 2022)

3.10.3.1 Acknowledgments

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

3.10.3.2 GAMS System

GAMS

- Fixed potential wrong dump file with implicit sets.

Connect

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

Embedded Python Code Facility

- Fixed a problem with ignoring `gams.ws.my_eps`.

3.10.3.3 Solvers

CONOPT

- New libraries 4.28.

3.10.3.4 Tools

GAMS Studio

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

3.10.3.5 APIs

GAMS Transfer Matlab

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

GAMS Transfer Python

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

GMD

- Fixed a problem with aliases to the universe.

3.10.4 39.2.1 Maintenance release (June 21, 2022)

3.10.4.1 Acknowledgments

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

3.10.4.2 GAMS System

GAMS

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

Connect

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

3.10.4.3 Solvers

CPLEX

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

SHOT

- New libraries 1.1 (c9bc78df).

3.10.5 39.3.0 Minor release (July 07, 2022)

3.10.5.1 Acknowledgments

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

3.10.5.2 GAMS System

GAMS

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

3.10.5.3 Solvers

SCIP Optimization Suite

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

SHOT

- New libraries 1.1 (033622c6).

3.10.5.4 Tools

GAMS Studio

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

3.10.5.5 APIs

GMO

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

3.11 38 Distribution

3.11.1 38.1.0 Major release (January 31, 2022)

3.11.1.1 Acknowledgments

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

3.11.1.2 Platforms

- As announced, dropped support for macOS 10.14 (Mojave).
- Added support for macOS 12 (Monterey).

3.11.1.3 GAMS System

GAMS

- Added new command line parameter dumpOptGDX to load the data needed in dumpOpt files from GDX instead of data statements.
- Added new command line parameter ReferenceLineNo to control the line numbers written to a reference file.
- We plan to drop the command line parameter `freeEmbeddedPython` with the next major release.
- Extended the Solve Summary written with asyncSolLst=1 to include information about controlling loop indices during job submission, name of the solver called, source line number of the solve statement, resource and iteration limit.
- Make handling of execution errors during model generation (e.g. division by zero in an equation) more consistent across different settings for the solveLink option. In the past, the error count was set to 0 automatically when returning from a solve with `solveLink=0` only (which is equivalent to setting execError=0 after the solve statement). This is not done anymore.
- Write additional information about execution errors during model generation (e.g. division by zero error) to the lst file, e.g., the equation name and index causing the problem.
- Improve recognition of '/' as path delimiter on Windows.
- Pause automated time dependent log update while executing embedded code sections.
- Check for certain byte order marks (BOMs) in GAMS input files. Files with UTF8 BOM can be processed now, other BOMs generate a compilation error.

- Added support for (mixed-integer) quadratic models whose Q matrices contain more than 2^{31} = 2,147,483,648 nonzeros in their upper triangle. At this time, only the solver links for CPLEX and ODHCPLEX are capable of handling such large model instances.
- Changed system suffix `isAlfaBeta` to `isAlphaBeta`.
- Fixed a problem for EMP models with holdFixed model attribute.
- Fixed a problem where IDCGenerateGDXInput, IDCGenerateGDXOutput, IDCGDXOutput , and IDCGenerateGDX did not write a GDX file if the execution was stopped with abort[.noError].

Extended Mathematical Programming (EMP)

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

Embedded Python Code Facility

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

Extrinsic Function libraries

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

3.11.1.4 Solvers

BARON

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

CONOPT

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

CONVERT

- The symbol `We` in DumpGDX is now stored sparse.
- Changed option default of GDXHessian and GDXQuadratic to 0.

COPT

- Added new parallel LP and MIP solver COPT 3.0.5 by Cardinal Operations (`https://www.shanshu.ai/copt`).
- For GAMS demo and community licenses, model size limitations as for other solvers apply. For an unlimited GAMS/COPT or GAMS/COPT-Link license, contact `sales@gams.com`.

CPLEX

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

DICOPT

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

KNITRO

- New libraries 13.0.
  - Introduces an updated, parallel, branch-and-bound solver for mixed-integer optimization. This updated solver is able to exploit parallelism and uses improved heuristics to solve mixed-integer problems much faster than before.
  - Offers new initial point strategies for non-convex quadratic programs (QPs) and quadratically constrained quadratic programs (QCQPs). These new initial point strategies improve the likelihood of finding the global solution, and can be used in conjunction with the Knitro multi-start procedure to search for global or better local solutions for non-convex QP and QCQP models.
  - Offers significant robustness and speed improvements on difficult nonlinear optimization problems.
  - Offers improved performance when using the SQP and MISQP algorithms on larger models.
  - Added options:
    - ncvx_qcqp_init: Specifies the initialization strategy used for non-convex QPs and QCQPs.
    - mip_cutoff: This value specifies the objective cutoff value for MIP.

    * mip_heuristic_lns: Specifies whether or not to enable the MIP large neighborhood search (LNS) heuristics.
    * mip_liftproject: Specifies rules for adding lift and project cuts.
    * mip_multistart: Use to enable MIP multi-start at the branch-and-bound level.
    * mip_numthreads: Specify the number of threads to use for MIP branch and bound.
  - Renamed options (old option names still available as synonyms but deprecated):
    * `bar_pencons` to bar_penaltycons.
    * `bar_penrule` to bar_penaltyrule.
    * `mip_integral_gap_abs` to mip_opt_gap_abs.
    * `mip_integral_gap_rel` to mip_opt_gap_rel.
    * `par_msnumthreads` to ms_numthreads.
    * `par_blasnumthreads` to blas_numthreads.
    * `par_lsnumthreads` to linsolver_numthreads.
    * `tuner_maxtimecpu` to tuner_maxtime_cpu.
    * `tuner_maxtimereal` to tuner_maxtime_real.
  - Added option values:
    * blasoption: 3.
    * mip_heuristic_diving: bits 1 - 4.
  - Changed option defaults:
    * restarts: -1 (auto).
    * mip_strong_candlim: 128.

- Fixed possible error of incorrect Jacobian for QCP models.

Lindo/LindoGlobal

- New libraries 13.0.319.

LocalSolver

- As announced, dropped LocalSolver and LocalSolver70.

MOSEK

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

Octeract

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

ODHCPLEX

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

SCIP Optimization Suite

- New libraries PaPILO 2.0.0.
  - Added dual postsolve functionality for LPs.
  - Immediate application of a presolvers reductions if run sequentially.
  - Improvements in performance and functionality of many presolvers.
  - For more details, see [22] and the `PaPILO CHANGELOG`.
  - Updated TBB to version 2021.5.0.
- New libraries SoPlex 6.0.0.
  - PaPILO can now be enabled as presolver by setting int:simplifier to the new value 2.
  - SoPlex can now be interrupted by Ctrl+C/SIGINT.
  - Changed default for option int:simplifier from 1 to new value 3. The value 1 now means "auto".
  - New option real:simplifier_modifyrowfac.
  - For more details, see [22] and the `SoPlex CHANGELOG`.
- New libraries SCIP 8.0.0.
  - Symmetry handling:
    - New symmetry handling inequalities based on the Schreier Sims table, which are able to handle symmetries of arbitrary kinds of variables.
    - The symmetry code can now heuristically search for orbitopes that do not completely handle a symmetry component and add certain Schreier Sims cuts to such components.
    - In-tree restarts due to tree size estimation have been made compatible with orbital fixing.
    - Improved performance of upgrades to packing/partitioning orbitopes, symresack cover separation, and propagation for orbisack and symresack.
    - Symmetry handling has been extended to detect also symmetries between variables that also appear in nonlinear constraints.
  - Cutting planes:
    - New separator mixing to generate mixing cuts.
    - New separator rlt to compute cuts via the reformulation-linearization technique (RLT).
    - New separator minor to compute cuts from 2x2 minors of a violated semidefiniteness constraint that is implied by the extended formulation for bilinear products.
    - New separator interminor to compute intersection cuts from 2x2 (not only principle) minors of a violated rank-one constraint that is implied by the extended formulation for bilinear products (currently disabled by default).
    - Separator "aggregation" now uses the objective cutoff row as base row and separates lifted cover cuts based on newer lifting function of Letchford and Souli (2019).
    - Separators "strongcg" and "gomory" now share the same computed basis.
  - Primal Heuristics:
    - New construction heuristic DPS which additionally needs a user-provided decomposition and splits the problem into several sub-SCIPs according to this decomposition. The sub-SCIPs are solved and updated repeatedly until a feasible solution of the original problem is found.
    - PADM can now reoptimize found solution with original objective function.
    - The RENS neighborhood in the heuristic ALNS now fixes fractional variables if necessary for reaching its target fixing rate.
    - Revised and improved heuristic subNLP, in particular choice of NLP iteration limit and starting condition and reuse of subSCIP.
  - The handling of algebraic expressions and nonlinear constraints has been rewritten:
    - Support for trigonometric functions sine and cosine and for the entropy function has been added.

  - Improved recognition of common subexpressions, in particular square and bilinear terms.
  - All type of nonlinear constraints, including quadratic and second-order-cones, are now handled by the constraint handler for nonlinear constraints. The specialized constraint handlers have been removed. However, methods that work on a particular nonlinear structure are now implemented via nonlinear handlers (nlhdlr).
  - Added nonlinear handler for quadratic, bilinear, convex, concave, and quotient expressions, second-order cone constraints, and perspective tightening for expressions in semi-continuous variables.
  - As before, an extended formulation for nonlinear constraints is constructed. However, this formulation is now handled implicitly and used to construct the LP relaxation only, while the original formulation is used to check feasibility, propagate variable domains, etc. This difference should drastically improved the likelihood that a solution that satisfies the nonlinear constraints in the presolved problem is also feasible for the original problem.
  - Products in nonlinear constraints are no longer disaggregated for the extended formulation.
  - Improved analysis on which nonlinear variables in nonlinear constraints can be increased or decreased without harming feasibility by taking monotonicity into account.
  - Improved linearization of (sums of) products of binary variables, e.g., clique information is now taken into account.
  - When a variable appears in only one concave less-or-equal constraint, it will be fixed to one of its bounds in more cases now.
  - New branching rule for variables in nonlinear constraints that scores candidates based on constraint violation, variable type, and pseudo-costs simultaneously. Variables that are added for the extended formulation of a nonlinear constraint are no longer branched on by default.
  - New strategies to deal with weak estimators for nonlinear functions and small violations of nonlinear constraints.
  - Improved under/overestimation of multidimensional vertex-polyhedral (e.g., multilinear or concave) functions by use of scaling and keeping the cut-generating LP around.
  - Added tables to print statistics of nonlinear constraint handler, nonlinear handlers, and handlers for all supported functions.
- Revised and improved interfaces to Ipopt and CppAD, e.g.:
  - Optimization of taped expressions and sparse Hessian evaluation of CppAD are used now.
  - Added table to print statistics on NLP solvers.
- The interfaces to Gurobi and Mosek are thread-safe now.
- Parameters removed:
  - `gams/resolvenlp`
  - `constraints/abspower/&lowast;`, `constraints/bivariate/&lowast;`, `constraints/quadratic/&lowa` `constraints/soc/&lowast;`
  - `constraints/nonlinear/cutmaxrange`, `constraints/nonlinear/linfeasshift`, `constraints/nonlinear/reformulate`, `constraints/nonlinear/sepanlpmincont`, `constraints/nonlinear/enfocutsremovable`, `constraints/nonlinear/maxexpansionexponent`, `constraints/nonlinear/upgrade/abspower`, `constraints/nonlinear/upgrade/and`, `constraints/nonlinear/upgrade/bivariate`, `constraints/nonlinear/upgrade/quadratic`
  - `constraints/orbitope/usedynamicprop`
  - `heuristics/multistart/nlpminimpr`
  - `heuristics/subnlp/iteroffset`, `heuristics/subnlp/iterquotient`, `heuristics/subnlp/nlpiterlim` `heuristics/subnlp/nlptimelimit`, `heuristics/subnlp/runalways`, `heuristics/subnlp/minimprove`, `heuristics/subnlp/nlpoptfile`, `heuristics/subnlp/resolvefromscratch`, `heuristics/subnlp/reso`
  - `propagating/symmetry/disableofrestart`
  - `separating/convexproj/nlptimelimit`, `separating/gauge/nlptimelimit`
  - `separating/strongcg/dynamiccuts`, `separating/strongcg/maxrounds`, `separating/strongcg/maxrou` `separating/strongcg/maxsepacuts`, `separating/strongcg/maxsepacutsroot`
- Parameters added:

  - branching/inference/conflictprio, branching/inference/cutoffprio
  - constraints/nonlinear/bilinmaxnauxexprs, constraints/nonlinear/linearizeheursol, constraints/nonlinear/re constraints/nonlinear/reformbinprodsand, constraints/nonlinear/reformbinprodsfac, constraints/nonlinear/checkvarlocks, constraints/nonlinear/conssiderelaxamount, constraints/nonlinear/en constraints/nonlinear/forbidmultaggrnlvar, constraints/nonlinear/forcestrongcut, constraints/nonlinear/pr constraints/nonlinear/propinenforce, constraints/nonlinear/rownotremovable, constraints/nonlinear/stron constraints/nonlinear/strongcutmaxcoef, constraints/nonlinear/tightenlpfeastol, constraints/nonlinear/var constraints/nonlinear/varboundrelaxamount, constraints/nonlinear/violscale, constraints/nonlinear/vpadj constraints/nonlinear/vpdualsimplex, constraints/nonlinear/vpmaxperturb, constraints/nonlinear/weakcu constraints/nonlinear/weakcutthreshold, constraints/nonlinear/upgrade/setppc, constraints/nonlinear/bra
  - cutselection/hybrid/∗
  - decomposition/disablemeasures
  - expr/log/minzerodistance, expr/pow/minzerodistance
  - heuristics/alns/nodesquotmin, heuristics/alns/initduringroot, heuristics/alns/maxcallssamesol
  - heuristics/dps/∗
  - heuristics/padm/reoptimize
  - heuristics/subnlp/expectinfeas, heuristics/subnlp/feastolfactor, heuristics/subnlp/iterinit, heuristics/subnlp/ninitsolves, heuristics/subnlp/nodesfactor, heuristics/subnlp/nodesoffset, heuristics/subnlp/presolveemphasis, heuristics/subnlp/setcutoff, heuristics/subnlp/successrateexp, heuristics/subnlp/opttol,
  - misc/avoidmemout
  - nlhdlr/bilinear/∗, nlhdlr/concave/∗, nlhdlr/convex/∗, nlhdlr/default/∗, nlhdlr/perspective/∗, nlhdlr/quadratic/∗, nlhdlr/quotient/∗, nlhdlr/soc/∗
  - nlpi/ipopt/∗
  - presolving/milp/probfilename
  - propagating/obbt/createlincons,
  - propagating/symmetry/addconflictcuts, propagating/symmetry/addstrongsbcs, propagating/symmetry/ad propagating/symmetry/detectsubgroups, propagating/symmetry/maxnconsssubgroup, propagating/symmetry/onlybinarysymmetry, propagating/symmetry/preferlessrows, propagating/symmetry/sstaddcuts propagating/symmetry/sstleaderrule, propagating/symmetry/sstleader propagating/symmetry/sstmixedcomponents, propagating/symmetry/ssttiebreakrule, propagating/symmetry/usedynamicprop
  - separating/maxcoefratiofacrowprep, separating/filtercutpoolrel
  - separating/gomory/genbothgomscg, separating/gomory/trystrongcg separating/gomorymi
  - separating/interminor/∗
  - separating/knapsackcover
  - separating/minor/∗
  - separating/mixing/∗
  - separating/rlt/∗
  - table/cons_nonlinear/active, table/cutsel/active, table/exprhdlr/active, table/nlhdlr/active, table/nlhdlr_bilinear/active, table/nlhdlr_quadratic/active, table/nlpi/active
  - timing/nlpieval

– Parameters changed:

  - default of constraints/nonlinear/maxproprounds changed from 1 to 10
  - default of display/nexternbranchcands/active changed from 2 to 1 also for nonlinear instances
  - default of display/nfrac/active changed from 2 to 1 also if discrete variables are present
  - default of heuristics/dualval/priority changed from 0 to -10
  - default of heuristics/lpface/priority changed from -1104000 to -1104010
  - default of heuristics/mutation/priority changed from -1103000 to -1103010
  - default of heuristics/nlpdiving/priority changed from -1003000 to -1003010
  - default of heuristics/repair/priority changed from 0 to -20
  - default of heuristics/simplerounding/priority from 0 to -30

    - default of heuristics/subnlp/forbidfixings from 1 to 0
    - default of heuristics/subnlp/itermin changed from 300 to 20
    - default of SCIPheuristics_subnlp_priority heuristics/subnlp/priority changed from -2000000 to -2000010
    - default of heuristics/trivialnegation/priority changed from 40000 to 39990
    - default of heuristics/trustregion/priority changed from -1102000 to -1102010
    - default of heuristics/trysol/priority changed from -3000000 to -3000010
    - default of misc/usesymmetry changed from 3 to 7 and range changed from {0,...,3} to {0,...,7}
    - default of presolving/qpkktref/maxrounds changed from -1 to 0
    - default of propagating/obbt/onlynonconvexvars changed from 0 to 1
    - default of propagating/symmetry/addsymresacks changed from 1 to 0
    - type of propagating/symmetry/recomputerestart changed from boolean to int with range {0,...,2}
    - default of separating/strongcg/maxbounddist changed from 1 to 0
    - default of separating/strongcg/priority changed from -2000 to -100000
    - replaced `separating/minortho` by cutselection/hybrid/minortho, replaced `separating/minorthoroot` by cutselection/hybrid/minorthoroot, replaced `separating/dircutoffdistfac` by cutselection/hybrid/dircutoffdistweight, replaced `separating/efficacyfac` by cutselection/hybrid/efficacyweight, replaced `separating/intsupportfac` by cutselection/hybrid/intsuppo replaced `separating/objparalfac` by cutselection/hybrid/objparalweight
  - For more details, see [22] and the `SCIP release notes`.

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

SHOT

- New libraries 1.1 (e225a002).
  - Added parameter Model.Reformulation.Quadratics.EigenValueDecomposition.Tolerance.

XPRESS

- New libraries 39.01.03.
  - Improved MIP performance:
    - The Optimizer now uses an ML module to decide whether to use in-tree cutting. See autoCutting.
    - Improved separation of Mixed Integer Rounding (MIR) cuts.
    - Improved cut activation and deactivation strategies.
    - Improved application of strong branching during the tree search.
    - More information carried through a MIP restart.
    - The 'R' local search heuristic will now be called more frequently in the early phase of a MIP solve.
  - Improved crossover performance after a Barrier solve through automatic objective perturbation. See barObjPerturb.

  - Improved Barrier performance for large problems on CPUs that support AVX2 (requires setting cpuPlatform to -2).
  - New options:
    * autoCutting: Automatically decide whether to generate cutting planes at local nodes in the tree.
    * barObjPerturb: Defines how the barrier perturbs the objective.
    * ioTimeout: Maximum number of seconds to wait for an I/O operation before it is cancelled.
    * maxStallTime: Maximum time in seconds that the Optimizer will continue to search for improving solution after finding a new incumbent.
    * preCliqueStrategy: Determines how much effort to spend on clique covers in presolve.
    * siftPresolveOps: Determines the presolve operations for solving the subproblems during the sifting algorithm.
  - Changed option default of xslp_mipDefaultAlgorithm: 1.
- Added option fixoptfile: name of option file which is read just before solving the fixed problem.
- Changed default value of option mipAddCutoff and mipRelCutoff to 0.
- Fixed unnecessary solve of fixed MIP (see mipCleanup) when solving a linear program as MIP.

3.11.1.5 Tools

BIB2GMS

- As announced, dropped the tool `bib2gms`.

CSV2GDX

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

GAMS Studio

- New version 1.9.4.
  - Added Project Options editor.
  - Added possibility to create an empty project.
  - Adjusted loading from a model library to add files to active project working directory if "Open in existing project" is checked.
  - Added selection of user instances on `GAMS Engine SaaS`.
  - Added selection of namespaces on GAMS Engine Server.
  - Stability improvements, bug fixes, and minor enhancements, e.g.:
    * Expanded MIRO installation location on macOS if AppBundle has been selected.
    * Adjusted check if the MIRO data contract is available before the MIRO app is deployed.
    * Changed mouse wheel behavior in tab bars:
      · `Mouse Wheel Up/Down`: moves active tab
      · `Ctrl + Mouse Wheel Up/Down`: changes active tab
    * Added touch pad support for tab bars.
    * Added highlight in `LST` file for current section header of `LXI` viewer.

* Changed behavior of search in selection: the search selection is now only set once until the user resets it.
* Changed "Clear" button to remove search selection on first click when present.
* Fixed non-overridden recurrence of immediate options on command line parameters.
* Fixed jump behavior of search when using "This File" and navigating different files.
* Fixed `Find Next` broken for results across multiple files.
* Fixed crash on showing "Project Options" for a project without a runnable `gms` file.
* Fixed search not updating cache when document has changed.
* Fixed `Clear Selection` button not working as intended, when search scope was set to anything but "Selection".
* Fixed random crash in `Search and Replace` dialog.
* Fixed that checks for file type did not ignore casing (`gms`, `Gms`, `GMS` are all valid).
* Fixed that the default file filter in search dialog expected files to include a '.' character on non-windows platforms.
* Fixed file filter in search dialog not being ignored for scopes "This File" and "Selection".
* Fixed search jump behavior when current file is excluded by file filter.
* Fixed completer and syntax help activation for user defined GAMS source files.

GDXDIFF

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

GDXMRW

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

HEXDUMP

- As announced, dropped the tool `hexdump`.

3.11.1.6 APIs

C++ high-level API

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

GAMS Transfer Matlab

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

GAMS Transfer Python

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

GDX

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

GMO

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

High-level APIs

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

PAL

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

Python

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

3.11.1.7 Model Libraries

GAMS Data Library

New model:

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

GAMS Test Library

New models:

- **dumpopt2.gms : Test dumpOpt with dumpOptGDX**
- **encoding01.gms : Test processing of source files with BOM**

3.11.1.8 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.11	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
BONMIN 1.8	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
COPT 3.0	✓	✓	✓
CPLEX 20.1	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓
GUROBI 9.5	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.14	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO 13.0	✓	✓	✓
LGO	✓	✓	✓

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
LINDO 13.0	✓	✓	✓
LINDOGLOBAL 13.0	✓	✓	✓
MILES	✓	✓	✓
MINOS 5.6	✓	✓	✓
MOSEK 9	✓	✓	✓
MSNLP	✓	✓	✓
NLPEC	✓	✓	✓
OCTERACT 4	✓	✓	
ODHCPLEX 6	✓	✓	
PATH	✓	✓	✓
QUADMINOS 5.6	✓	✓	✓
SBB	✓	✓	✓
SCIP 8.0	✓	✓	✓
SHOT 1.1	✓	✓	✓
SNOPT 7.7	✓	✓	✓
SOPLEX 6.0	✓	✓	✓
XA	✓	✓	
XPRESS 39.01	✓	✓	✓

3.11.2 38.2.0 Minor release (February 17, 2022)

3.11.2.1 Acknowledgments

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

3.11.2.2 GAMS System

GAMS

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

3.11.2.3 Solvers

BARON

- New libraries 22.2.3.

CONOPT

- New libraries 4.26.
  - Removed option "Flg_Crash_Basis".
  - Changed default of option Lim_Start_Degen from 10 to 100.

CONVERT

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

COPT

- Added option readparams to instruct COPT to read an additional parameters file.

CPLEX

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

GAMSCHK

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

Ipopt

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

XPRESS

- New libraries 39.01.04.

3.11.2.4 APIs

High-level APIs

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

GAMS Transfer Python

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

GMO

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

3.11.3 38.2.1 Maintenance release (February 19, 2022)

3.11.3.1 Solvers

CONOPT

- Fixed warning about unknown option name "FLG_CRASH_BASIS" (introduced with GAMS 38.2.0).

Convert

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

Octeract

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

3.11.4 38.3.0 Minor release (April 05, 2022)

3.11.4.1 Acknowledgments

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

3.11.4.2 GAMS System

GAMS

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

3.11.4.3 Solvers

BARON

- New libraries 22.3.21.

CONVERT

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

COPT

- Fixed that indicators from solver options file were ignored.

CPLEX

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

GUROBI

- New libraries 9.5.1.

IPOPT

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

KNITRO

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

Mosek

- New libraries 9.3.18.

Octeract

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

SCIP Optimization Suite

- New libraries SCIP 8.0 (189beeaf30).
  - Clique lifting for setpartitioning/packing/covering constraints is now only disabled if it has been applied.
- New libraries PaPILO 2.0 (g166dea4).

XPRESS

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

3.11.4.4 Tools

MODEL2TEX

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

MPS2GMS

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

GAMS Studio

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

3.11.4.5 APIs

GAMS Transfer Python

- Fixed memory leak associated with `gams2numpy`.

3.12 37 Distribution

3.12.1 37.1.0 Major release (November 11, 2021)

3.12.1.1 Acknowledgments

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

3.12.1.2 Platforms

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

3.12.1.3 GAMS System

GAMS

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

Embedded Python Code Facility

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

Extended Mathematical Programming (EMP)

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

Windows Installer

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

3.12.1.4 Solvers

ANTIGONE, IPOPT, SHOT, SCIP

- Updated Intel MKL to version 2021.2.0 on Windows.

CONVERT

- Fix default behavior if reading option file fails.

CPLEX

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

GUROBI

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

GUSS

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

KNITRO

- Added support for MCP model type.

Lindo/LindoGlobal

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

LocalSolver

- New libraries 10.5 (20211014).

Mosek

- New libraries 9.3.7.

Octeract

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

SCIP

- New libraries 7.0 (b2afa5403b).

SHOT

- New libraries 1.1.0 (11fda1ec).
  - New option Model.Reformulation.Quadratics.EigenValueDecomposition.Formulation.
  - New option Model.Reformulation.Quadratics.EigenValueDecomposition.Method.
  - Renamed option `Model.Reformulation.Quadratics.UseEigenValueDecomposition` to Model.Reformulation.Quadratics.EigenValueDecomposition.Use.
  - Default value for option Subsolver.Gurobi.NumericFocus changed from 2 to 1.

SOPLEX

- New libraries 5.0 (ad7592b9).

XPRESS

- New libraries 38.01.05.

3.12.1.5 Tools

BIB2GMS

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

FINDTHISGAMS

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

GAMS Studio

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

GDX2HAR/HAR2GDX

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

HEXDUMP

- We will remove the tool `hexdump` in a future GAMS release. A similar functionality is available on Unix systems via the `od` utility with parameters `od -A x -t x1z -v`. On Windows `od` is distributed as part of the POSIX utilities.

SCENRED

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

3.12.1.6 APIs

C++ high-level API

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

gams2numpy

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

GAMS Transfer

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

GMD

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

GMO

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

PAL

- Added routines `palLicenseSolverCheckSizes64` and `palLicenseCheck64` for solvers that use 64-bit integers for non-zero counts.

Python high-level API

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

3.12.1.7 Model Libraries

GAMS Model Library

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

New Models:

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

GAMS Test Library

New models:

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

3.12.1.8 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.11	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
BONMIN 1.8	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
CPLEX 20.1	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓
GUROBI 9.5	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.14	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO 12.4	✓	✓	✓
LGO	✓	✓	✓
LINDO 13.0	✓	✓	✓
LINDOGLOBAL 13.0	✓	✓	✓
LOCALSOLVER 10.5	✓	✓	✓
MILES	✓	✓	✓
MINOS	✓	✓	✓
MOSEK 9	✓	✓	✓
MSNLP	✓	✓	✓
NLPEC	✓	✓	✓
OCTERACT 3.5	✓	✓	
ODHCPLEX 6	✓	✓	
PATH	✓	✓	✓
SBB	✓	✓	✓
SCIP 7.0	✓	✓	✓
SHOT 1.1	✓	✓	✓
SNOPT 7.7	✓	✓	✓
SOPLEX 5.0	✓	✓	✓
XA	✓	✓	
XPRESS 38.01	✓	✓	✓

3.13 36 Distribution

3.13.1 36.1.0 Major release (August 02, 2021)

3.13.1.1 Acknowledgments

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

3.13.1.2 GAMS System

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

GAMS

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

  ```
  ejk(j,k)..  sum(i, x(i,j,k)) =e= 42;
  ```

  Added new command line parameter sys19 to deactivate this performance optimization.
- Added new function platformCode and new compile time constants to be used with it.
- Added information about available CPU sockets, cores, and threads to the log.
- Improve error message for invalid values of the `.scale` attribute.
- Allow to read environment variables and command line parameters at execution time in put context.
- Added Put_Utility commands `setEnv` and `dropEnv` to allow environment variable modification at execution time.
- Added Python function `gams.get_env` to EmbeddedCode Python to access modified environment variables after Python module `os` has been initialized.
- Fixed a problem causing an unexpected error when beginning execution time embedded code as first statement after starting from a restart file.
- Fixed a potential warning using execution time embedded code after starting from a restart file which was created with action=c.

Windows Installer

- Windows installer is now built with `Inno Setup 6`.
- Command line parameter `/allUsers=yes|no` has been deprecated and will be removed in a future GAMS release. Use the new parameter `/installAllUsers=yes|no` instead.
- New command line flags `/allUsers` and `/currentUser` can be used to explicitly control if the installation is performed in administrative or non-administrative installation mode.

GMSPython

- The Python version has been updated to `3.8.10`.

3.13.1.3 Solvers

Bonmin

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

CONVERT

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

CPLEX

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

GUROBI

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

GUSS

- Added support of option savepoint for scenario solves.

Ipopt and IpoptH

- New libraries 3.14.1.

  Attention
  - Value `pardiso` of option linear_solver to choose Pardiso from Intel MKL has been **renamed to** `pardisomkl`. Renamed `pardiso_*` options to pardisomkl_*.
  - Added value `pardiso` for option linear_solver to choose to use Pardiso from the `Pardiso Solver Project`. This solver is **not included in the GAMS distribution**. A user needs to provide a Pardiso library. Added option pardisolib to specify name and path of Pardiso library. Added pardiso_* options.
  - Added option hsllib to specify name and path of HSL library to be loaded at runtime (for users of GAMS/Ipopt or users that want to use HSL_MA77).
  - Changed default of honor_original_bounds to `no`. In addition, Ipopt now ensures that, in addition to option bound_relax_factor, variable bounds are relaxed by at most constr_viol_tol. The Ipopt log now shows the violation of variable bounds in the solution.
  - Fixed that marginals for fixed variables were not computed if fixed_variable_treatment is set to `make_parameter` (the default). Added value `make_parameter_nodual` for this option to disable calculation of marginals for fixed variables (which can cost an extra evaluation of the Jacobian).
  - Added options max_wall_time, print_options_mode, print_advanced_options, and timing_statistics.
  - Ipopt should now be threadsafe also when used with MUMPS as linear solver (a mutex ensures that only one thread executes MUMPS at a time).
  - For more information, including bugfixes, see the `Ipopt 3.14.0 release notes`.
- Changed default of constr_viol_tol to 1e-6.
- GAMS ResLim now sets the default for Ipopt option max_wall_time instead of max_cpu_time.

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

JAMS

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

KNITRO

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

Lindo/LindoGlobal

- New libraries 13.0.291.

LocalSolver

- New libraries 10.5 (20210624).

MOSEK

- Added option QEXTRACTALG.
- New libraries 9.2.47.

ODHCPLEX

- New libraries 6.0.4.

PATH

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

SCIP

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

SHOT

- New libraries 1.1 (c898dd84).
  - SHOT can now call itself to solve fixed NLP problems. Activated with new value 2 for option Primal.FixedInteger.Solver. For nonconvex problems it is still recommended to use an external NLP solver.
    Added also options Subsolver.SHOT.ReuseHyperplanes.Fraction, Subsolver.SHOT.ReuseHyperplanes.Use, Subsolver.SHOT.UseFBBT.
  - Partition convex nonseparable quadratic functions as separate constraints using an eigenvalue decomposition-based reformulation. Activated with new option `Model.Reformulation.Quadratics.UseEigenV`
  - Support for performing an initial polyhedral approximation of the nonlinear feasible set before feasibility-based bound tightening. Activated with new option Model.BoundTightening.InitialPOA.Use.
    Added also options Model.BoundTightening.InitialPOA.CutStrategy, Model.BoundTightening.InitialPOA.Itera
    Model.BoundTightening.InitialPOA.StagnationIterationLimit, Model.BoundTightening.InitialPOA.Constraint
    Model.BoundTightening.InitialPOA.ObjectiveConstraintTolerance, Model.BoundTightening.InitialPOA.Objec
    Model.BoundTightening.InitialPOA.ObjectiveGapRelative, Model.BoundTightening.InitialPOA.StagnationCo
    Model.BoundTightening.InitialPOA.TimeLimit.
  - Added support for semi-continuous and semi-integer variables.
  - Added support for special ordered sets.
  - Improved support for generating supporting hyperplanes for the entire nonlinear feasible set instead of the feasible sets for the individual constraint functions. Activated with new option Dual.ESH.Rootsearch.UseMaxFunction.

  - Improved support for passing nonconvex quadratic functions directly to the MIP solver (if supported).
  - Added option ModelingSystem.GAMS.QExtractAlg.
  - Added option Output.GAMS.AlternateSolutionsFile.
  - Added option Dual.HyperplaneCuts.SaveHyperplanePoints.
  - Added option Dual.MIP.InfeasibilityRepair.Use.
  - Added option Dual.ReductionCut.Use.
  - Changed default of option Subsolver.Cbc.Scaling from 0 to 4.
  - Changed default of option Subsolver.Cplex.NumericalEmphasis from 0 to 1.
  - Changed default of option Subsolver.Gurobi.PoolSolution from 1 to 10.
  - Changed default of option Model.BoundTightening.FeasibilityBased.TimeLimit from 5 to 2.
  - Changed default of option Termination.DualStagnation.IterationLimit from 50 to ∞.
  - Changed meaning of values 1 and 2 for option Model.Reformulation.Bilinear.IntegerFormulation.
- Updated Ipopt to Ipopt 3.14.1, see also above.

XPRESS

- New libraries 38.01.03.
  - Xpress Optimizer Release Notes:
    * Improved MIP performance.
    * It is now possible to activate a special heuristic solve mode for MIPs, which emphasizes finding new solutions.
    * Improved sifting algorithm when used with Barrier.
    * Improved numerical stability of the Barrier algorithm.
  - Xpress Nonlinear Release Notes:
    * Significant MINLP performance improvements due to improved nonlinear presolver.
    * Improved performance on instances with very large separable nonlinear expressions due to imroved automatic differentiation.
    * Multistart now uses the nonlinear primal integral for tie breaking.
  - Added options:
    * barPerturb: In numerically challenging cases it is often advantageous to apply perturbations on the KKT system to improve its numerical properties.
    * barRefIter: After terminating the barrier algorithm, further refinement steps can be performed.
    * clamping: Allows for the adjustment of returned solution values such that they are always within bounds.
    * clamping_dual: Adjust primal slack values to always be within constraint bounds.
    * clamping_primal: Adjust primal solution to always be within primal bounds.
    * clamping_rdj: Adjust reduced costs to always be within dual bounds implied by the primal solution.
    * clamping_slacks: Adjust dual solution to always be within the dual bounds implied by the slacks.
    * genConsAbsTransformation: Specifies the reformulation method for absolute value general constraints at the beginning of the search.
    * heurEmphasis: Specifies an emphasis for the search w.r.t. primal heuristics and other procedures that affect the speed of convergence of the primal-dual gap.
    * inputTol: Tolerance on input values elements.

    - mipComponents: Determines whether disconnected components in a MIP should be solved as separate MIPs.
    - mipConcurrentNodes: Sets the node limit for when a winning solve is selected when concurrent MIP solves are enabled.
    - mipConcurrentSolves: Selects the number of concurrent solves to start for a MIP.
    - mipRestartFactor: Fine tune initial conditions to trigger an in-tree restart.
    - mipRestartGapThreshold: Initial gap threshold to delay in-tree restart.
    - netStallLimit: Limit the number of degenerate pivots of the network simplex algorithm, before switching to either primal or dual simplex.
    - nodeProbingEffort: Adjusts the overall level of node probing.
    - outputControls: Toggles the printing of all control settings at the beginning of the search.
    - ppFactor: Partial pricing candidate list sizing parameter.
    - preFolding: Determines if a folding procedure should be used to aggregate continuous columns in an equitable partition.
    - siftPasses: Determines how quickly we allow to grow the worker problems during the sifting algorithm.
    - siftSwitch: Determines which algorithm to use for solving the subproblems during sifting.
    - xslp_presolveOps_noLinear: Avoid performing linear reductions at the NLP level.
    - xslp_presolveOps_noSimplifier: Avoid simplifying nonlinear expressions.
  - Added option values:
    - deterministic: 2.
    - mipRestart: 2.
  - Changed option defaults:
    - mipPresolve: -257 to -1.
    - mipPresolve_symmetryReductions: 0 to 1.
  - Deprecated options:
    - `heurStrategy`. Use heurEmphasis instead.
    - `xslp_matrixTol`. Use matrixTol instead.
  - Removed options:
    - `treePresolve_keepBasis`.
    - `treePresolve`.
    - `treePresolveOps`.
    - `xslp_analyze_autosavePool`.
    - `xslp_analyze_recordLinearization`.
    - `xslp_analyze_recordLinesearch`.
    - `xslp_presolvePassLimit`.
    - `xslp_timePrint`.
- Added option algAfterNetwork: Algorithm to be used for the clean up step after the network simplex solver.
- Added option qextractalg: quadratic extraction algorithm in GAMS interface.
- Removed options that were marked deprecated in GAMS 35.

3.13.1.4 Tools

Cholesky, Eigenvalue, Eigenvector, Invert

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

GAMS Studio

- New version 1.7.2.
  - Dropped support for Ubuntu 16.04 LTS and openSUSE Leap 15.2 because the updated AppImage now requires glibc 2.27. Dropped support for Debian 9.
  - New feature: Allow specifying location of User Model Libraries in Settings dialog.
  - Improved visual distinction between current and other groups in Project Explorer.
  - Added a warning message before printing large files.
  - Added a list of solvers for each model type to be selected in Extended Parameter Editor and Configuration Editor.
  - Added shortcut `Alt+I` (macOS: `Option+I`) to fold blocks of dollar control options.
  - Added setting to automatically collapse dollar control option blocks when opening a file (default: off).
  - Changed shortcuts to show/hide Extended Parameter Editor
    - Windows/Linux: From `Ctrl-Alt-3` to `Shift+Ctrl+3`.
    - macOS: From `Shift-Command-3` to `Control-Option-3`.
  - Improvements for syntax highlighter and code completer, e.g.:
    - Added support for system attributes and compile-time constants.
    - Added support of quoted elements in syntax highlighter.
    - Disabled code completer for certain code blocks like `$onEmbeddedCode` or `$onPut`.
  - Improvements for remote execution support, e.g.:
    - Added support for GAMS Engine server with self-signed certificate (`https`).
    - Added support for GAMS Engine server with `http` protocol.
    - Added option to remember login credentials for GAMS Engine in Settings.
    - Renamed "MIRO" Settings page to "Remote" to better reflect new content.
    - Improved GAMS Engine dialog.
    - Added file type `EFI` to send additional files to a GAMS Engine server.
  - Stability improvements and minor bug fixes, e.g.:
    - Fixed duplicate file entries in the Project Explorer when having two references with different case to a single file
    - Fixed Studio title bar appearing outside of screen in some cases.
    - Fixed wrong content of "File" tab in Reference File Viewer.
    - Fixed missing check of some data locations for GAMS license file.
    - Fixed `ESC` key not behaving like close button in Settings dialog.
    - Fixed crash when reopening a closed log tab.
    - Fixed that MIRO menu entries were not enabled when MIRO was automatically discovered.
    - Fixed that theme could be broken in Settings dialog when using cancel button.

GDXRank, GDXRename

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

MCFilter

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

MSAppAvail, Shellexecute, XLSTalk

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

3.13.1.5 APIs

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

CFG

- The function `cfgAlgAllowsModifyProblem` now reports correctly for all solvers if they provide the `xyzModifyProblem` functionality instead of a only for a fixed (and incorrect) list of solvers.

C++ high-level API

- Removed dependency on Qt. Now requiring C++17.
- Added support for Microsoft Visual Studio 2019.
- As announced we dropped support for Microsoft Visual Studio 2015.
- As announced, the Clang compiled version of `libgamscpp.dylib` does not work on macOS 10.14 or older anymore. It has been moved to the subfolder `apifiles/C++/lib/Clang/`.
  The new `libgamscpp.dylib` in `apifiles/C++/lib/` is compiled with GCC.

gams_engine

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

gams2numpy

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

GAMSX

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

GEV

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

GMO

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

Solverlink library interface

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

3.13.1.6 Model Libraries

GAMS API Library

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

GAMS Data Library

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

GAMS Test Library

New models:

- **platform01.gms : Test platform function**
- **qcp12.gms : Test different GMO Q extraction algorithms**
- **asyncfix01.gms : Test asynchronous solves with holdFixed**
- **savep2.gms : Test savepoint with async solves**
- **convert16.gms : CONVERT test suite - Basic test of convert output for quadratic model**
- **embpy09.gms : Test Embedded Code after restart**
- **scensol8.gms : Check if solver with modify problem use hotstart in GUSS**
- **gdxrename1.gms : Simple gdxrename test**
- **gdxrename2.gms : Tests some gdxrename stuff**
- **scensol9.gms : Test option savepoint in GUSS**
- **embpy10.gms : Test Embedded Code for Environment Variables**
- **etsuf01.gms : Test Execution Time Suffixes**
- **emp33.gms : JAMS: matches from model statement may be ignored**

Models trilib01, trilib02, trilib03, parlib01, cpplib00:

- Removed jinja2 from associated source archive. It is now required to have the Python package `jinja2` installed to run the `ql.py` script.

3.13.1.7 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.11	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
BONMIN 1.8	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
CPLEX 20.1	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓
GUROBI 9.1	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.14	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO 12.4	✓	✓	✓
LGO	✓	✓	✓

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
LINDO 13.0	✓	✓	✓
LINDOGLOBAL 13.0	✓	✓	✓
LOCALSOLVER 10.5	✓	✓	✓
MILES	✓	✓	✓
MINOS	✓	✓	✓
MOSEK 9	✓	✓	✓
MSNLP	✓	✓	✓
NLPEC	✓	✓	✓
ODHCPLEX 6	✓	✓	
PATH	✓	✓	✓
SBB	✓	✓	✓
SCIP 7.0	✓	✓	✓
SHOT 1.1	✓	✓	✓
SNOPT 7.7	✓	✓	✓
SOPLEX 5.0	✓	✓	✓
XA	✓	✓	
XPRESS 38.01	✓	✓	✓

3.13.2 36.2.0 Minor release (September 03, 2021)

3.13.2.1 Acknowledgments

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

3.13.2.2 GAMS System

GAMS

- Added new command line parameter holdFixedAsync to allow the use of holdFixed for models solved asynchronously. Activating this option can lead to inconsistent solutions being reported.
- Fixed a potential crash introduced with GAMS 36.1.0 related to the improved performance of model generation.

3.13.2.3 Solvers

CONVERT

- Fixed MCP mapping for DumpGDX.

GAMSCHK

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

GUSS

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

IPOPT

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

MOSEK

- New libraries 9.3.4.

SHOT

- New libraries 1.1 (9a8893e7).

3.13.2.4 Tools

GAMS Studio

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

3.14 35 Distribution

3.14.1 35.1.0 Major release (April 29, 2021)

3.14.1.1 Acknowledgments

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

3.14.1.2 Platforms

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

3.14.1.3 GAMS System

GAMS

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

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

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

3.14.1.4 Solvers

AlphaECP

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

ANTIGONE, IPOPT, SHOT, SCIP

- Updated Intel MKL to version 2021.2.0 on Linux and macOS.

BARON

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

BONMIN

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

CONOPT

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

CPLEX

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

GAMSCHK

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

GUROBI

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

GUSS/Scenario Solver

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

KNITRO

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

- linesearch: Indicates which linesearch strategy to use for the Interior/Direct or SQP algorithm to search for a new acceptable iterate.
- newpoint: Specifies additional action to take after every iteration in a solve of a continuous problem.
- presolve_level: Set the level of presolve operations to enable through the Knitro presolver.
- presolve_initpt: Control whether the Knitro presolver can shift a user-supplied initial point.
- presolve_passes: Set a maximum limit on the number of passes through the Knitro presolve operations.
- presolveop_tighten: Determine whether or not to enable the Knitro presolve operation to tighten variable bounds.
- restarts: Specifies whether or not to enable automatic restarts in Knitro.
- restarts_maxit: When restarts are enabled, this option can be used to specify a maximum number of iterations before enforcing a restart.
- strat_warm_start: Specifies whether or not to invoke a warm-start strategy.
- findiff_relstepsize: Specifies the relative stepsize used for finite-difference gradients.
- findiff_terminate: Specifies the termination criteria when using finite-difference gradients.
- infeastol_iters: The optimization process will terminate if the relative change in the feasibility error is less than infeastol for infeastol_iters consecutive infeasible iterations.
- xtol_iters: The optimization process will terminate if the relative change in all components of the solution point estimate is less than xtol for xtol_iters.
- bar_conic_enable: Enable special treatments for conic constraints when using the Interior/Direct algorithm (has no affect when using the Interior/CG algorithm).
- bar_initpi_mpec: Specifies the initial value for the MPEC penalty parameter π used when solving problems with complementarity constraints using the barrier algorithms.
- bar_linsys: Indicates which linear system form is used inside the Interior/Direct algorithm for computing primal-dual steps.
- bar_maxcorrectors: Specifies the maximum number of corrector steps allowed for primal-dual steps.
- bar_slackboundpush: Specifies the amount by which the barrier slack variables are initially pushed inside the bounds.
- bar_switchobj: Indicates which objective function to use when the barrier algorithms switch to a pure feasibility phase.
- act_lpfeastol: Specifies the feasibility tolerance used for linear programming subproblems solved when using the Active Set or SQP algorithms.
- act_lppenalty: Indicates whether to use a penalty formulation for linear programming subproblems in the Knitro Active Set or SQP algorithms.
- act_lppresolve: Indicates whether to apply a presolve for linear programming subproblems in the Knitro Active Set or SQP algorithms.
- act_parametric: Indicates whether to use a parametric approach when solving linear programming (LP) subproblems when using the Knitro Active Set or SQP algorithms.
- act_qppenalty: Indicates whether to use a penalty formulation for quadratic programming subproblems in the Knitro SQP algorithm.
- mip_clique: Specifies rules for adding clique cuts.
- mip_cutfactor: This value specifies a limit on the number of cuts added to a node subproblem.
- mip_heuristic_terminate: Specifies the condition for terminating the MIP heuristic.
- mip_intvar_strategy: Specifies how to handle integer variables.
- mip_mir: Specifies rules for adding mixed integer rounding cuts.
- mip_selectdir: Specifies the MIP node selection direction rule (for tiebreakers) for choosing the next node in the branch and bound tree.

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

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

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

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

- Removed formerly deprecated options:
  - maxcrossit

  - mu
  - feasible
  - feasmodetol
  - barrule
- Removed unused option `reform`.
- Fixed an issue with options not being forwarded to Knitro.

Lindo/LindoGlobal

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

LocalSolver

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

MOSEK

- New libraries 9.2.41.

MPSGE

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

ODHCPLEX

- New libraries 5.34.

SBB

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

SCIP

- New libraries 7.0 (c53330372e).

SNOPT

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

XPRESS

- Added support for choosing a different algorithm than primal simplex for solving the fixed MIP (see mipCleanup). The algorithm can be chosen with defaultAlg and lpFlags.
- Deprecated the following options:
  - `algorithm`: Use defaultAlg or lpFlags instead.
  - `extraPresolve`
  - `lpThreads`
- Added the following options:
  - algAfterCrossover: Algorithm to be used for the final clean up step after the crossover.
  - autoPerturb: Simplex: Indicates whether automatic perturbation is performed.
  - autoScaling: Whether the Optimizer should automatically select between different scaling algorithms.
  - backtrackTie: Branch and Bound: Specifies how to break ties when selecting the next node to work on when a full backtrack is performed.
  - barCores: If set to a positive integer it determines the number of physical CPU cores assumed to be present in the system by the barrier algorithm.
  - barFailIterLimit: Newton barrier: Maximum number of consecutive iterations that fail to improve the solution in the barrier algorithm.
  - barFreeScale: Defines how the barrier algorithm scales free variables.
  - barGapTarget: Newton barrier: Target tolerance for the relative duality gap.

- barKernel: Newton barrier: Defines how centrality is weighted in the barrier algorithm.
- barObjScale: Defines how the barrier scales the objective.
- barOrderThreads: If set to a positive integer it determines the number of concurrent threads for the sparse matrix ordering algorithm in the Newton barrier method.
- barPresolveOps: Controls the Newton barrier specific presolve operations.
- barRegularize: Determines how the barrier algorithm applies regularization on the KKT system.
- barRhsScale: Defines how the barrier scales the right hand side.
- branchChoice: Once a global entity has been selected for branching, this control determines which of the branches is solved first.
- branchDisj: Branch and Bound: Determines whether the optimizer should attempt to branch on general split disjunctions during the branch and bound search.
- branchStructural: Branch and Bound: Determines whether the optimizer should search for special structure in the problem to branch on during the branch and bound search.
- `cacheSize`: Newton barrier: L2 or L3 (see notes) cache size in kB (kilobytes) of the CPU.
- choleskyAlg: Newton barrier: Type of Cholesky factorization used.
- choleskyTol: Newton barrier: Tolerance for pivot elements in the Cholesky decomposition of the normal equations coefficient matrix, computed at each iteration of the barrier algorithm.
- conflictCuts: Branch and Bound: Specifies how cautious or aggressive the optimizer should be when searching for and applying conflict cuts.
- coresPerCpu: Used to override the detected value of the number of cores on a CPU.
- cpuTime: How time should be measured when timings are reported in the log and when checking against time limits.
- crossoverAccuracyTol: Newton barrier: Determines how crossover adjusts the default relative pivot tolerance.
- crossoverIterLimit: Newton barrier: Maximum number of iterations that will be performed in the crossover procedure before the optimization process terminates.
- crossoverOps: Newton barrier: Bit vector for adjusting the behavior of the crossover procedure.
- cutFactor: Limit on the number of cuts and cut coefficients the optimizer is allowed to add to the matrix during global search.
- cutSelect: Bit vector providing detailed control of the cuts created for the root node of a global solve.
- dualGradient: Simplex: Dual simplex pricing method.
- dualize: Whether presolve should form the dual of the problem.
- dualizeOps: Bit-vector control for adjusting the behavior when a problem is dualized.
- dualPerturb: Factor by which the problem will be perturbed prior to optimization by dual simplex.
- dualStrategy: Bit-vector control specifies the dual simplex strategy.
- elimFillin: Amount of fill-in allowed when performing an elimination in presolve.
- elimTol: Markowitz tolerance for the elimination phase of the presolve.
- feasibilityPump: Branch and Bound: Decides if the Feasibility Pump heuristic should be run at the top node.
- feasTolPerturb: Determines how much a feasible primal basic solution is allowed to be perturbed when performing basis changes.
- feasTolTarget: Target feasibility tolerance for the solution refiner.
- forceParallelDual: Dual simplex: Specifies whether the dual simplex solver should always use the parallel simplex algorithm.
- genConsDualReductions: Parameter specifies whether dual reductions should be applied to reduce the number of columns and rows added when transforming general constraints to MIP structs.

- heurBeforeLp: Branch and Bound: Determines whether primal heuristics should be run before the initial LP relaxation has been solved.
- heurDiveIterLimit: Branch and Bound: Simplex iteration limit for reoptimizing during the diving heuristic.
- heurDiveRandomize: Level of randomization to apply in the diving heuristic.
- heurDiveSoftRounding: Enables a more cautious strategy for the diving heuristic.
- heurDiveSpeedUp: Branch and Bound: Changes the emphasis of the diving heuristic from solution quality to diving speed.
- heurDiveStrategy: Branch and Bound: Chooses the strategy for the diving heuristic.
- heurForceSpecialObj: Branch and Bound: Whether local search heuristics without objective or with an auxiliary objective should always be used, despite the automatic selection of the Optimizer.
- heurFreq: Branch and Bound: Frequency at which heuristics are used in the tree search.
- heurSearchEffort: Adjusts the overall level of the local search heuristics.
- heurSearchFreq: Branch and Bound: How often the local search heuristic should be run in the tree.
- heurSearchRootCutFreq: How frequently to run the local search heuristic during root cutting.
- heurSearchRootSelect: Bit vector control for selecting which local search heuristics to apply on the root node of a global solve.
- heurSearchTreeSelect: Bit vector control for selecting which local search heuristics to apply during the tree search of a global solve.
- `heurStrategy`: Branch and Bound: Heuristic strategy.
- historyCosts: Branch and Bound: How to update the pseudo cost for a global entity when a strong branch or a regular branch is applied.
- indLinBigM: Indicator constraints can be internally converted to regular rows (i.e. linearized) using a BigM coefficient whenever the BigM coefficient is smaller or equal to this value.
- indPreLinBigM: During presolve, indicator constraints can be internally replaced with regular rows (i.e. linearized) using a BigM coefficient whenever the BigM coefficient is smaller or equal to this value.
- `l1Cache`: Newton barrier: L1 cache size in kB (kilo bytes) of the CPU.
- lnpBest: Number of infeasible global entities to create lift-and-project cuts for during each round of Gomory cuts at the top node (see gomCuts).
- lnpIterLimit: Number of iterations to perform in improving each lift-and-project cut.
- localChoice: Controls when to perform a local backtrack between the two child nodes during a dive in the branch and bound tree.
- lpFlags: Bit-vector control which defines the algorithm for solving an LP problem or the initial LP relaxation of a MIP problem.
- lpFolding: Simplex and barrier: Whether to fold an LP problem before solving it.
- lpLogDelay: Time interval between two LP log lines.
- lpLogStyle: Simplex: Style of the simplex log.
- lpRefineIterLimit: Simplex iteration limit the solution refiner can spend.
- markowitzTol: Markowitz tolerance used for the factorization of the basis matrix.
- maxCutTime: Maximum amount of time allowed for generation of cutting planes and reoptimization.
- maxImpliedBound: Presolve: When tighter bounds are calculated during MIP preprocessing, only bounds whose absolute value are smaller than maxImpliedBound will be applied to the problem.
- maxLocalBacktrack: Branch-and-Bound: How far back up the current dive path the optimizer is allowed to look for a local backtrack candidate node.

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

- preConvertSeparable: Presolve: Reformulate problem with non-diagonal quadratic objective and/or constraints as diagonal quadratic or second-order conic constraints.
- preDomCol: Presolve: Determines the level of dominated column removal reductions to perform when presolving a mixed integer problem.
- preDomRow: Presolve: Determines the level of dominated row removal reductions to perform when presolving a problem.
- preDupRow: Presolve: Determines the type of duplicate rows to look for and eliminate when presolving a problem.
- preElimQuad: Presolve: Allows for elimination of quadratic variables via doubleton rows.
- preImplications: Presolve: Determines whether to use implication structures to remove redundant rows.
- preLinDep: Presolve: Determines whether to check for and remove linearly dependent equality constraints when presolving a problem.
- preObjCutDetect: Presolve: Determines whether to check for constraints that are parallel or near parallel to a linear objective function, and which can safely be removed.
- presolveMaxGrow: Limit on how much the number of non-zero coefficients is allowed to grow during presolve, specified as a ratio of the number of non-zero coefficients in the original problem.
- presolveOps: Specifies the operations which are performed during the presolve.
- presolvePasses: Number of reduction rounds to be performed in presolve.
- primalPerturb: Factor by which the problem will be perturbed prior to optimization by primal simplex.
- primalUnshift: Determines whether primal is allowed to call dual to unshift.
- qcCuts: Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for the root node, when solving a mixed integer quadratically constrained or mixed integer second order conic problem with outer approximation.
- qcRootAlg: Determines which algorithm is to be used to solve the root of a mixed integer quadratically constrained or mixed integer second order cone problem, when outer approximation is used.
- qSimplexOps: Controls the behavior of the quadratic simplex solvers.
- quadraticUnshift: Determines whether an extra solution purification step is called after a solution found by the quadratic simplex (either primal or dual).
- randomSeed: Sets the initial seed to use for the pseudo-random number generator in the Optimizer.
- refineOps: Specifies when the solution refiner should be executed to reduce solution infeasibilities.
- relaxTreeMemoryLimit: When the memory used by the branch and bound search tree exceeds the target specified by the treeMemoryLimit control, the optimizer will try to reduce this by writing nodes to the global file.
- repairIndefInitEq: Controls if the optimizer should make indefinite quadratic matrices positive definite when it is possible.
- resourceStrategy: Controls whether the optimizer is allowed to make nondeterministic decisions if memory is running low in an effort to preserve memory and finish the solve.
- rootPresolve: Determines if presolving should be performed on the problem after the global search has finished with root cutting and heuristics.
- sbBest: Number of infeasible global entities to initialize pseudo costs on each node.
- sbEffort: Adjusts the overall amount of effort when using strong branching to select an infeasible global entity to branch on.
- sbEstimate: Branch and Bound: How to calculate pseudo costs from the local node when selecting an infeasible global entity to branch on.

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

- Added options to conveniently set bits of XPRESS bitmap options:
  - New bits of refineOps:
    * refineOps_lpOptimal
    * refineOps_mipSolution
    * refineOps_mipNodeLp
    * refineOps_lpPresolve
    * refineOps_iterativeRefiner
    * refineOps_refinerPrecision
    * refineOps_refinerUsePrimal
    * refineOps_refinerUseDual
    * refineOps_mipFixGlobals
    * refineOps_mipFixGlobalsTarget
  - New bits of mipPresolve:
    * mipPresolve_reducedCostFixing
    * mipPresolve_logicPreprocessing
    * mipPresolve_allowChangeBounds
    * mipPresolve_dualReductions
    * mipPresolve_globalCoefTightening
    * mipPresolve_objBasedReductions
    * mipPresolve_allowTreeRestart
    * mipPresolve_symmetryReductions
  - New bits of presolveOps:
    * presolveOps_singletonColRemoval
    * presolveOps_singletonRowRemoval

    * presolveOps_forcingRowRemoval
    * presolveOps_dualReductions
    * presolveOps_redundantRowRemoval
    * presolveOps_duplicateColRemoval
    * presolveOps_duplicateRowRemoval
    * presolveOps_strongDualReductions
    * presolveOps_variableEliminations
    * presolveOps_noIpReductions
    * presolveOps_noGlobalDomainChange
    * presolveOps_noAdvIpReductions
    * presolveOps_linDependRowRemoval
    * presolveOps_noIntVarEliminations
    * presolveOps_noIntVarAndSosDetect
  - New bits of scaling:
    * scaling_rowScaling
    * scaling_colScaling
    * scaling_rowScalingAgain
    * scaling_maximum
    * scaling_curtisReid
    * scaling_byMaxElemNotGeoMean
    * scaling_bigM
    * scaling_simplexObjScaling
    * scaling_ignoreQuadRowPart
    * scaling_beforePresolve
    * scaling_noScalingRowsUp
    * scaling_noScalingColsDown
    * scaling_disableGlobalObjScaling
    * scaling_rhsScaling
    * scaling_noAggressiveQScaling
    * scaling_slackScaling
  - New bits of lpFlags:
    * lpFlags_dual
    * lpFlags_primal
    * lpFlags_barrier
    * lpFlags_network
  - New bits of cutSelect:
    * cutSelect_clique
    * cutSelect_mir
    * cutSelect_cover
    * cutSelect_mirRowAggregation
    * cutSelect_flowpath
    * cutSelect_implication
    * cutSelect_liftAndProject
    * cutSelect_disableCutRows
    * cutSelect_gubCover
    * cutSelect_zeroHalf
    * cutSelect_indicator
  - New bits of treeCutSelect:
    * treeCutSelect_clique
    * treeCutSelect_mir

    * treeCutSelect_cover
    * treeCutSelect_mirRowAggregation
    * treeCutSelect_flowpath
    * treeCutSelect_implication
    * treeCutSelect_liftAndProject
    * treeCutSelect_disableCutRows
    * treeCutSelect_gubCover
    * treeCutSelect_zeroHalf
    * treeCutSelect_indicator

- Renamed the following options to match the original XPRESS options names. Old option names currently serve as alias, but are deprecated and will be removed in an upcoming release:
  - `treePresolveKeepBasis` to `treePresolve_keepBasis`.
  - `slpKnitroOptFile` to knitroOptFile.
  - All XPRESS Nonlinear (SLP) options, e.g., `slpSolver` to xslp_solver.
- Fixed default values of single bit options for XPRESS bitmap options with negative default values.

3.14.1.5 Tools

GAMS Studio

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

* Fixed wrong sort order after resetting GDX Viewer.
* Fixed GAMS engine not appending `/api` to the URL on macOS when clicking `OK` directly after editing the URL.
* Fixed `eolCom` not working in some dollar control option lines (like `$include`).
* Fixed jump to next search result if the result is in a different file and the current one has no results.
* Fixed jump to search result occasionally not working.
* Fixed `Find Next/Prev` behavior for `.opt` files.

GDXDUMP

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

GDXMERGE

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

GDXXRW

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

3.14.1.6 APIs

CFG

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

DCT

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

gams2numpy

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

GMO

- Updated to better handle cases where quadratic structure is explicitly detected (i.e. `useQ` is set) and some *but not all* of the nonlinear equations are quadratic. Several new functions have been added. In addition, some existing functions have been renamed - the new names are more descriptive and/or more consistent with new and existing function names. The old names are still available as deprecated synonyms - we expect to remove these synonyms in the near future and potentially without warning.
  - Updated API version to 20 - compatible with some older API versions.
  - New read-only properties:
    - `gmoObjLNZ`: Number of linear nonzeros in objective gradient.
    - `gmoObjNLNZEx`: Number of GMOORDER_NL nonzeros in objective gradient.
    - `gmoObjQNZEx`: Number of GMOORDER_Q nonzeros in objective gradient.
    - `gmoObjCVecNZ`: Number of nonzeros in c vector of objective (-1 if Q information not used).
    - `gmoObjConstEx`: Objective constant - this is independent of `useQ`.
    - `gmoObjQConst`: Constant in solvers quadratic objective.
  - New functions:
    - `gmoGetObjCVec`: Get c vector of quadratic objective.
    - `gmoGetObjSparseEx`: Get information for gradient of objective function (sparse).
    - `gmoGetRowStatEx`: Get Jacobian row nonzero counts: total and by GMOORDER_XX.
    - `gmoGetRowCVecNZOne`: Number of nonzeros in c vector of row si (-1 if Q information not used).
    - `gmoGetRhsOneEx`: Get individual equation RHS - independent of `useQ`.
    - `gmoGetRowSparseEx`: Get info for one row of Jacobian (sparse).
    - `gmoGetRowCVec`: Get c vector of the quadratic form for one row.
    - `gmoGetRowQConst`: Get the constant of the quadratic form for one row.
  - New read-write properties:
    - `gmoHessInclQRows`: if `useQ` is true, this boolean property toggles the inclusion of GMOORDER_Q rows in the Hessian.
      Attention

      The default was originally `true` but has accidentally been changed to `false` with this release (35.1.0). That is, if `useQ` is set, then the Hessian does not include entries from quadratic equations anymore. The default is changed back to true in GAMS 38.2.0.
  - Renamed properties/functions:
    - `gmoObjQNZ` → `gmoObjQMatNZ`: Number of nonzeros in lower triangle of Q matrix of objective (-1 if `useQ` is false).
    - `gmoGetObjQ` → `gmoGetObjQMat`: Get lower triangle of Q matrix of objective.
    - `gmoGetRowQ` → `gmoGetRowQMat`: Get lower triangle of Q matrix of one row.

High-Level APIs

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

Low-Level APIs

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

PAL

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

3.14.1.7 Model Libraries

GAMS Model Library

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

GAMS Test Library

New models:

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

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS

3.14.1.8 Solver/Platform availability matrix

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

3.14.2 35.2.0 Minor release (June 02, 2021)

3.14.2.1 Acknowledgments

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

3.14.2.2 GAMS System

GAMS

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

EMP

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

GMSPython

- Removed package `tqdm`.

3.14.2.3 Solvers

AlphaECP

- New library with some minor bug fixes.

CONVERT

- Fixed exporting of variable scale attribute in DumpGDX format.

CPLEX

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

KNITRO

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

3.14.2.4 Tools

GAMS Studio

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

3.14.2.5 APIs

Python

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

DCT

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

3.14.2.6 Model Libraries

GAMS Model Library

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

3.15 34 Distribution

3.15.1 34.1.0 Major release (January 29, 2021)

3.15.1.1 Acknowledgments

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

3.15.1.2 Platforms

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

3.15.1.3 GAMS System

GAMS

- Improved performance of *dense* operations with out-of-order index positions, as in the following code snippet, where the assignment to `d1` was significantly slower than the one to `d2` in the past:

```
d1(j,i) = sum(ij(i,j), exp(pij(i,j)));
d2(j,i) = sum(ij(i,j), abs(pij(i,j)));
```

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

GMSPython

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

3.15.1.4 Solvers

AMPL

- As announced, dropped the meta-solver AMPL.

BARON

- New libraries 21.1.13.
  - Improved interface to IPOPT for local search.
  - Changed default for option LPSol to new value -1, which makes BARON use CPLEX for LP or MIP relaxations, if licensed, and CLP/CBC otherwise.
  - Further performance improvements and bugfixes.
- Added option BarName to specify name of file into which to write problem in BARON format.

BDMLP

- As announced, dropped the solver BDMLP.

BENCH

- As announced, dropped the tool BENCH.

CONOPT

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

CONVERT

- The two solver links CONVERT and CONVERTD were merged under the name CONVERT. Features that were formerly available in only one of the two solver links are now all usable with the new CONVERT link (except for the dropped features, see below). CONVERTD is an alias for CONVERT now.
- JuMP format:
  - Added constraint names.
  - Using `@NLconstraint` and `@NLobjective` instead of `add_NL_constraint` and `set_NL_objective`, respectively.
  - Changed default of option JuMP to `jump.jl`.
- Pyomo format:
  - Fix export of SOS1 and SOS2 variable types.
  - Changed default of option Pyomo to `pyomo.py`.
- DumpGDX format (formerly Hessian or Jacobian):
  - The current behaviour of disabling variable and equation names or UELs when setting the option to "NOVENAMES.gdx" or "NOUELS.gdx" has no special effect anymore. Use GDXNames and GDXUELs instead.
- New options:
  - Width
  - DumpGDX (replaces options Hessian and Jacobian)
  - GDXNames (instead of setting the former options Hessian or Jacobian to "NOVENAMES.gdx")
  - GDXUELs (instead of setting the former options Hessian or Jacobian to "NOUELS.gdx")
  - GDXHessian (enables hessian information for DumpGDX)
  - GDXQuadratic (enables quadratic information for DumpGDX)
- Options PermuteEqus and PermuteVars are now available for all target languages.
- Removed the following formats:
  - AlphaECP
  - AmplNLC
  - Analyze
  - AnalyzeS
  - Baron (GAMS/Baron link provides the option BarName to export this format)
  - Lgo
  - LindoMPI (GAMS/Lindo link provides the option WRITEMPI to export this format)
  - LocalSolver (GAMS/Localsolver link provides the option `writelsp` to export this format)
  - LSPSol
  - Memo
  - Minopt
  - SFS
  - ViennaDag
- Removed the following experimental formats:
  - AmplNLCG
  - CHull

  - CppAD
  - Lago
  - NLP2MCPE
  - PDCO
  - Qmaker (use DumpGDX with GDXQuadratic enabled instead)
  - UnitBC

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

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

CPLEX

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

- Fixed error messages for CPLEX error 5002 (CPXERR_Q_NOT_POS_DEF).

- Fixed an error that caused a crash for special option file input.

- Fixed an error that wrote additional quotes in GDX files of ranging and column generation information.

- Fixed miptrace timing info.

Gurobi

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

  - csport
  - cspriority
  - csrouter
  - cstimeoutfrac
  - cstlsinsecure
  - icsecretkey
  - icpool
  - icpriority
  - instantcloud

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

Ipopt

- Updated MUMPS to version 5.3.5.

KESTREL

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

KNITRO

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

LINGO

- As announced, dropped the meta-solver LINGO.

LS

As announced, dropped the tool LS.

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

For example, the GAMS code

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

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

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

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

Mosek

- New libraries 9.2.35.

MPECDUMP

- As announced, dropped the tool MPECDUMP.

NLPEC

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

ODHCPLEX

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

PYOMO

- As announced, dropped the meta-solver PYOMO.

SCIP

- New libraries 7.0 (9a11615b47).

SoPlex

- New libraries 5.0.2.

XPRESS

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

3.15.1.5 Tools

CSV2GDX

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

GAMS Studio

- New version 1.5.2.
  - New feature: Allow to edit and store custom color themes in the settings dialog.
  - New feature: Allow to define custom sections in the code, that can be folded using `$onFold` / `$offFold`.
  - Improved MIRO deploy dialog.
  - Added e-mail field for NEOS submissions in NEOS dialog. This can also be initialized from environment variable `NEOS_EMAIL` (which can be set permanently in gamsconfig.yaml).
  - Changed "Autosave on Run" setting to save ALL modified files, not just files of the current group.
  - Changed default light theme comment color to grey for better distinction from declarations.
  - Disabled GAMS syntax highlighting between `$onPut/$offPut` and `$onEcho/$offEcho`.
  - Added "select entry" on clicking timestamp in log.
  - Pressing `Enter` in the LXI tree scrolls to the corresponding position in the LST view.
  - Adjusted list of files in Reference File Viewer to be sorted by order in which they have been used.
  - Stability improvements and minor bug fixes, e.g.:
    - Fixed rare problem with undetected external file changes.

* Fixed broken include file links in combination with end of line comments.
* Fixed missing request for new search cache after switching file.
* Fixed rare crash related to LOG output.
* Fixed endlessly recurring message box for invalid GDX file.
* Fixed missing update of parameter history for a file assigned to more than one group.
* Fixed search results not being highlighted in `.log` files.
* Fixed search performance problem after reset search parameters.
* Fixed behavior of `ESC` key when current widget has a selection.
* Fixed syntax highlighting of `table` statements not detecting row headers.
* Fixed print dialog shortcut not working directly after Studio launch.

GDX2VEDA

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

GDXDUMP

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

GDXMRW

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

3.15.1.6 APIs

C++

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

Python

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

Matlab

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

GMO

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

3.15.1.7 Model Libraries

GAMS API Library

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

GAMS Data Library

New models:

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

GAMS Test Library

New models:

- **show01.gms : Test $show feature**
- **dmpsym01.gms : Test dmpSym and dmpUserSym feature**
- **ide01.gms : Basic test for IDE mode**
- **cplex08.gms : CPLEX test suite - conflict analysis** (841)
- **convert13.gms : CONVERT test suite - apply dict to gams format** (842)

- **convert14.gms : CONVERT test suite - nonlinear expression reformulations** (843)
- **convert15.gms : CONVERT test suite - support of nonlinear functions** (844)
- **unload12.gms : Make sure that sameAs and diag cannot be exported to GDX** (845)
- **unload13.gms : Unload with relaxed and full domain** (846)
- **unload14.gms : Check that unload happens at GDX file closing time** (847)
- **dumpopt1.gms : Test verbatim dumpOpt** (848)

Removed models:

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

3.15.1.8 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.10	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
BONMIN 1.8	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
CPLEX 20.1	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓
GUROBI 9.1	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.13	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO	✓	✓	✓
LGO	✓	✓	✓
LINDO 12.0	✓	✓	✓
LINDOGLOBAL 12.0	✓	✓	✓
LOCALSOLVER 9.5	✓	✓	✓
MILES	✓	✓	✓
MINOS	✓	✓	✓
MOSEK 9	✓	✓	✓
MSNLP	✓	✓	✓
NLPEC	✓	✓	✓
ODHCPLEX 5	✓	✓	
PATH	✓	✓	✓
SBB	✓	✓	✓
SCIP 7.0	✓	✓	✓

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
SHOT 1.0	✓	✓	✓
SNOPT	✓	✓	✓
SOPLEX 5.0	✓	✓	✓
XA	✓	✓	
XPRESS 36.01	✓	✓	✓

3.15.2 34.2.0 Minor release (February 05, 2021)

3.15.2.1 Acknowledgments

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

3.15.2.2 Solvers

CONOPT

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

CONVERT

- Fix variable coefficients in JuMP format.

CPLEX

- Fixed issue of non-working option userincbicall.

GAMSCHK

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

3.15.3 34.3.0 Minor release (February 25, 2021)

3.15.3.1 Acknowledgments

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

3.15.3.2 GAMS System

GAMS

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

3.15.3.3 Solvers

CONOPT

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

CONVERT

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

GAMSCHK

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

MOSEK

- New libraries 9.2.38.

ODHCPLEX

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

OsiMosek

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

3.15.3.4 Tools

GAMS Studio

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

GDXDUMP

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

GDXXRW

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

3.15.3.5 APIs

3.15.3.6 Expert Level APIs

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

Matlab

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

3.16 33 Distribution

3.16.1 33.1.0 Major release (November 01, 2020)

3.16.1.1 Acknowledgments

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

3.16.1.2 Platforms

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

3.16.1.3 GAMS System

GAMS

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

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

  generates this log:

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

- Fixed a problem using captureModelInstance with solveLink=5/6/7; if `captureModelInstance=1` and `solveLink=6/7`, `solveLink` will be reset to `3/4` automatically.
- Fixed a problem with file names longer than 128 characters for option IDCGenerateJSON.

GMSPython

- Added the packages `urllib3`, `certifi`, and their dependencies.

3.16.1.4 Solvers

AMPL, LINGO, MPECDUMP, PYOMO

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

ANTIGONE, IPOPT, SHOT, SCIP

- Updated Intel MKL to version 2020.2 on Linux and macOS.

BARON

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

BDMLP

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

BENCH

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

CONOPT

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

Convert and ConvertD

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

Couenne

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

CPLEX

- The two solver links CPLEX and CPLEXD were merged under the name CPLEX. Features that were formerly available in only one of the two solver links are now all usable with the new CPLEX link. Those features include:
  - Formerly only CPLEX:
    - SOS1 and SOS2 sets
    - Branching priorities
    - Indicator constraints
    - Conflict refiner
    - Ranging / sensitivity analysis
    - Solution pool for mixed-integer (quadratic) problems
  - Formerly only CPLEXD:
    - Better handling of quadratic constraints (e.g. providing duals)
    - Hot start capability in GUSS
    - Solving multiple instances of `GAMSModelInstance` in parallel using GAMS APIs
    - `Solvelink=solvelink.asyncThreads`
    - Cplex remote object and distributed MIP
    - Benders decomposition
- New features:
  - Partial MIP starts: allow to only include variable levels to the MIP start that are integer (within the tolerance tryint, see mipstart)
  - Multiple MIP starts provided via GDX (see multimipstart)
  - Lower / upper bound ranging / sensitivity analysis (see bndrng)
  - Lazy constraints (see userlazyconcall or .lazy)
  - Added MIP statistics (e.g. number of nodes, solve time, objective lower and upper bounds) to BCH call via GDX files usergdxnameinc and usergdxname.

- Added options:
  - Added option multimipstart for multiple MIP starts
  - Added option conflictdisplay
  - Added option multobjtolmip: the default value changes the behavior of the multi-objective optimization for continuous models
  - Added option bndrng: lower / upper bound ranging / sensitivity analysis
  - Added option indicoptstrict: handle erroneous indicator constraints
  - Added option userlazyconcall: adding lazy constraints using BCH
  - Added dot option .lazy: select linear constraints as lazy constraints
  - Added option mipstopexpr: custom MIP stopping criterion to terminate during branch-and-bound
- Changed options:
  - Changed behaviour of rngrestart: If the specified file extension is GDX, the ranging results will be exported as GDX instead of GMS.
  - Changed behaviour of miptracenode and miptracetime: option has no effect and MIP trace is appended when the solver reports global progress
  - Changed behaviour of quality: added entries regarding indicator constraints and MIP condition numbers
  - Changed default of randomseed from 0 to 201909284
  - Changed default of tuningtilim from 10000 to 1e75
  - Changed default of tuningdettilim from 1e7 to 1e75
  - Changed default of workmem from 128 to 2048
  - Changed name rtlcuts to rltcuts
  - Added documentation for partial MIP starts to mipstart
- Updated documentation for multiobjective optimization about CPLEX's two different strategies for continuous and discrete models (see also multobjtolmip).
- Fixed an error in BCH user heuristics where a false objective function of the incumbent was reported to CPLEX (in former CPLEXD link).

GUROBI

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

IPOPT

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

Kestrel

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

Lindo/LindoGlobal

- New libraries 12.0.210.
- Added the facility to detect irreducible infeasible and unbounded sets (IIS and IUS) and some options to control this.
- Added support for functions `div`, `log2`, `round`, and `trunc`.

LocalSolver

- New libraries 9.5 (20200923).

LS

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

MOSEK

- New libraries 9.2.28.

ODHCPLEX

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

SCIP

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

SHOT

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

SolveEngine

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

SoPlex

- New libraries 5.0 (2afa64b6).

XPRESS

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

3.16.1.5 Tools

GAMS IDE and GAMS Studio

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

GAMS Studio

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

MessageReceiverWindow

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

MODEL2TEX

- Deprecated command line parameter `-e` has been removed.

3.16.1.6 APIs

.NET

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

C++

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

Python

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

3.16.1.7 Expert Level APIs

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

3.16.1.8 Model Libraries

GAMS API Library

New models:

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

GAMS Test Library

New models:

- **lindgl05.gms : Test new functions added to GAMS/Lindo(Global)** (828)
- **cplex04.gms : CPLEX test suite - free gams model test** (829)
- **cplex05.gms : CPLEX test suite - interactive** (830)
- **cplex06.gms : CPLEX test suite - multiobjective** (831)
- **cplex07.gms : CPLEX test suite - bch incumbent checker** (832)
- **sens01.gms : sensitivity / ranging test** (833)
- **mipstopexpr1.gms : mip stop expression test** (834)
- **attrib01.gms : Test attribute setting and preservation in Convert** (835)
- **gamsjupyter01.gms : Test GAMS Jupyter Notebooks** (836)
- **cmi01.gms : Test captureModelInstance option with all solveLink combinations** (837)

3.16.1.9 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.10	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
BDMLP	✓	✓	✓
BONMIN 1.8	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
CPLEX 12.10	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓
GUROBI 9.1	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.13	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO	✓	✓	✓
LGO	✓	✓	✓
LINDO 12.0	✓	✓	✓
LINDOGLOBAL 12.0	✓	✓	✓
LOCALSOLVER 9.5	✓	✓	✓
MILES	✓	✓	✓
MINOS	✓	✓	✓
MOSEK 9	✓	✓	✓
MSNLP	✓	✓	✓
NLPEC	✓	✓	✓
ODHCPLEX 5	✓	✓	
PATH	✓	✓	✓
SBB	✓	✓	✓
SCIP 7.0	✓	✓	✓
SHOT 1.0	✓	✓	✓
SNOPT	✓	✓	✓
SOPLEX 5.0	✓	✓	✓
XA	✓	✓	
XPRESS	✓	✓	✓

3.16.2 33.2.0 Minor release (December 01, 2020)

3.16.2.1 Acknowledgments

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

3.16.2.2 GAMS System

GAMS

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

3.16.2.3 Solvers

CPLEX

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

KNITRO

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

3.16.2.4 Tools

GAMS Studio

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

3.17 32 Distribution

3.17.1 32.1.0 Major release (July 31, 2020)

3.17.1.1 Acknowledgments

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

3.17.1.2 Platforms

- Dropped support for Sparc Solaris and AIX systems.

3.17.1.3 GAMS System

GAMS

- New default values for some GAMS options:

	new default	old default
logOption	3	1
empty	on	off
errMsg	1	0
errorLog	2147483647	0
lstTitleLeftAligned	1	0
putPW	32767	255
pageContr	2	3
pageSize	0	58
pageWidth	32767	255
optCR	0.0001	0.1
intVarUp	0	1
resLim	1e10	1000
solveLink	2	0
iterLim	2147483647	2e9

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

- Extended the impact of command line parameter etLim, so that it reduces resLim for models solved automatically if that is greater than `etLim - timeElapsed`.
- Added new option zeroToEps to allow reading zero values as EPS at execution time.
- Added new options procTreeMemMonitor and procTreeMemTicks to enable monitoring of a memory high-water mark for the GAMS process and all its children.
- Adjusted behavior of command line parameters restart, restartNamed, save, saveObfuscate, xsave, and xsaveObfuscate: If a file name is given including a file extension, this extension is kept. In the past it was replaced by a default extension. Also, `?` is not allowed in the file name when writing a save file anymore.
- Extended influence of command line parameter inputDir, so that it also extends the search path for GDX files loaded with $gdxIn.
- Extended the search space for files included via $libInclude, extrinsic function libraries, and external equation libraries: Now, the directories `inclib`, `extrinsic_functions`, and `external_equations`, resp., are checked in all GAMS standard locations before `[GAMS System Directory]/inclib`, `[GAMS System Directory]`, and `[GAMS Working Directory]`, respectively.
- Renamed the command line parameter `execErr` to maxExecError to be in sync with the related function maxExecError. The old name is still available as a synonym.
- Added command line parameter previousWork to instruct GAMS to write a workfile using the format of an older GAMS version. This is useful in particular with submission to the NEOS server.
- Adjusted the function values returned by the power functions (e.g. `x**y`). In some cases where `x` is very small and positive the derivatives may become unbounded but the function values are well-behaved and can still be computed precisely and without overflow.
- Fixed a problem on Linux when the `XGD_CONFIG_DIRS` or `XGD_DATA_DIRS` variables have more than 7 directory entries.

GMSPython

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

macOS Installer

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

Windows Installer

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

3.17.1.4 Solvers

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

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

CONOPT

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

CONVERT

- New output format: JuMP scalar model.

GUROBI

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

Ipopt

- Updated Mumps to 5.3.3.

KESTREL

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

KNITRO

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

LocalSolver

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

Mosek

- New libraries 9.2.14.

ODHCPLEX

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

SoPlex

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

XPRESS

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

3.17.1.5 Tools

CHK4UPD

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

FINDTHISGAMS

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

GAMS Studio

- New version 1.3.3.
  - New feature: Code Folding
    - Shortcut `Alt+L` switches folding state of the current block.
    - Shortcut `Alt+O` folds and `Shift+Alt+O` unfolds all foldable blocks.
    - Code folding is possible for pairs of parentheses that span multiple lines or Dollar Control Options that come in pairs.
  - New feature: Remote execution on NEOS Server for Optimization
    - Models can be executed remotely via the menu `GAMS -> Run NEOS` or the execute button located in the toolbar.
    - Note: Usually it takes some time before the GAMS version at NEOS gets updated. During that time one might get an error because of an incompatible workfile version. This can be avoided by setting the command line parameter previousWork to 1.
  - New feature: Navigation History
    - `Mouse button 4/5` or `Alt+Left/Right` on Windows/Linux and `Ctrl+[/]` on macOS can now be used to jump back to a previous text cursor positions or forward again. This is helpful to quickly navigate between different locations in files that one is currently working with.
  - New feature: Distraction Free Mode
    - Distraction Free Mode quickly turns off everything except for the editor window for better focus on modeling work. This is especially useful for small screens. When leaving Distraction Free Mode all Studio widgets will be put back where they were. To switch to and from Distraction Free Mode go to the Menu > View > Distraction Free Mode, or press `Ctrl+Alt+Enter` on Windows and Linux, or `Cmd+Option+Enter` on macOS.
  - New feature: Print support for text files (`gms`, `txt`, `lst`, `dmp`, ...). This is accessible with the shortcut `Ctrl+P` or in the menu `File -> Print`.
  - Added automatic switch to first non-empty tab when searching the Model Library Explorer. The old tab selection is restored if that tab contains results again.
  - Added header labels to copy actions in GDX Viewer table view. Two additional entries that allow to copy without the labels are available via the context menu.
  - Added `dmp` extension to list of executable file types.
  - Added `Shift+Option+Up/Down` hotkey to start block-edit on macOS.
  - Changed terminal to native macOS Terminal on macOS.
  - Stability improvements and minor bug fixes, e.g.:
    - Fixed syntax highlighter not recognizing "+" and "-" in "Model" statement.
    - Fixed missing `LXI` pane for files containing more than one dot.
    - Fixed group name for `gms` files containing more than one dot in the name.
    - Fixed rare random crash when opening the MIRO Model Assembly File dialog.
    - Fixed some icons not following selected theme.
    - Fixed line wrapping behavior not following settings.
    - Fixed that bookmarks in help view were not being saved permanently.
    - Fixed crash on reading LXI files containing empty lines.
    - Fixed some issues with the license file detection and info in the "About GAMS" dialog.
    - Fixed missing reload if a opened GDX file gets recreated.

MODEL2TEX

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

3.17.1.6 APIs

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

.NET

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

Python

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

3.17.1.7 Model Libraries

GAMS Model Library

New models:

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

GAMS Test Library

New models:

- **zerotoeps1.gms : Test data loading with $onEps and option zeroToEps** (821)
- **idir01.gms : Test including files from different locations** (822)
- **mrw02.gms : Test MessageReceiverWindow.exe from Python** (823)
- **memmon1.gms : Test procTreeMemMonitor=1 behavior** (824)
- **gams2numpy01.gms : Test gams.core.numpy Python API string mode** (826)
- **gams2numpy02.gms : Test gams.core.numpy Python API raw mode** (827)

GAMS API Library

New models:

- **Pgamstransfer.gms : Test gamstransfer API** (58)
- Pgams2numpy.gms : Test gams.numpy API (59)

3.17.1.8 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS
ALPHAECP 2.10	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓
BARON	✓	✓	✓
BDMLP	✓	✓	✓
BONMIN 1.8	✓	✓	✓
CBC 2.10	✓	✓	✓
CONOPT 3	✓	✓	✓
CONOPT 4	✓	✓	✓
COUENNE 0.5	✓	✓	✓
CPLEX 12.10	✓	✓	✓
DECIS	✓	✓	✓
DICOPT 2	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓
GUROBI 9.0	✓	✓	✓
GUSS	✓	✓	✓
IPOPT 3.13	✓	✓	✓
KESTREL	✓	✓	✓
KNITRO	✓	✓	✓
LGO	✓	✓	✓
LINDO 12.0	✓	✓	✓
LINDOGLOBAL 12.0	✓	✓	✓
LOCALSOLVER 9.5	✓	✓	✓
MILES	✓	✓	✓
MINOS	✓	✓	✓
MOSEK 9	✓	✓	✓
MSNLP	✓	✓	✓
NLPEC	✓	✓	✓
ODHCPLEX 5	✓	✓	
PATH	✓	✓	✓
SBB	✓	✓	✓
SCIP 7.0	✓	✓	✓
SHOT 1.0	✓	✓	✓
SNOPT	✓	✓	✓
SOLVEENGINE	✓	✓	✓
SOPLEX 5.0	✓	✓	✓
XA	✓	✓	
XPRESS 35.01	✓	✓	✓

3.17.2 32.2.0 Minor release (August 26, 2020)

3.17.2.1 Acknowledgments

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

3.17.2.2 GAMS System

GAMS

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

Windows Installer

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

3.17.2.3 Solvers

Gurobi

- New libraries 9.0.3.

Lindo/LindoGlobal

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

Mosek

- New libraries 9.2.18.

SCIP

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

SoPlex

- New libraries 5.0 (3623dbc0).

3.17.2.4 Tools

GAMS Studio

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

GDX2VEDA

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

3.17.2.5 Model Libraries

GAMS Test Library

Updated models:

- **savep1.gms : execute_loadpoint: save point - clear - loadpoint - reoptimize** (65)

3.18 31 Distribution

3.18.1 31.1.0 Major release (May 01, 2020)

3.18.1.1 Acknowledgments

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

3.18.1.2 Platforms

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

3.18.1.3 GAMS System

GAMS

- Extended the syntax of the model statement to allow to restrict the domain of variables used in the model at one spot.
- Added dollar control options $on|offSuffixDLVars and command line parameter SuffixDLVars to control whether it is allowed to use suffixes on variables with limited domains in a model.
- Added new command line parameters to initialize the state of certain dollar control options: Digit, Empty, EolCom, Filtered, InlineCom, Listing.
- Extended the behavior of the command line parameter checkErrorLevel, so that it also initializes the state of the dollar control option $on/offCheckErrorLevel.
- Extended the behavior of the command line parameter strictSingelton, so that it also initializes the state of the dollar control option $on/offStrictSingleton.
- Added new command line parameters to set the default values for certain put file attributes: PutND, PutNR, PutPS, PutPW.
- Removed the limit on the number of significant digits allowed in floating-point values in GAMS source.
  Historically, GAMS has run on widely different platforms where even the floating-point arithmetic varied. To ensure the precision specified in the GAMS input didn't exceed what could be stored and computed with as GAMS ran, we limited the amount of precision (i.e. the number of digits used for numbers in the GAMS source) that could be specified. This limit could be relaxed by using the $offDigit control.
  Modern computers all support IEEE double-precision arithmetic. It's a well-accepted and familiar standard, and the new behavior - to convert the decimal value in the GAMS source into the nearest double-precision value, regardless of the number of digits specified - is what is generally expected. There is a limit (currently 40) on the number of digits treated as significant: digits beyond this point are quietly treated as zero.
  For "nice" values (i.e. those with fewer than 17 digits of precision and exponents of limited size, like 3.14159 or 2.5032e10) there is no difference in the old and new behavior. But for values specified with 17 or more digits of precision or having exponents of large size, the double-precision values can vary slightly between the old and new systems. The new values are the correct ones.
  The new command line parameter sys18 has been added in case any users require the old behavior.
- New options encryptKey and decryptKey to encrypt and decrypt source based on a key rather than a target license. This allows developers to create encrypted input files that can be executed with any GAMS license and the right key.
- GAMS will search standard locations for the license file `gamslice.txt` including the GAMS system directory. The standard locations for your operating system are described in the installation notes. You can also continue to use the license command line parameter to point to a specific license file.
- GAMS can now use a configuration file `gamsconfig.yaml` to specify default values for command line parameters and environment variables: see details in GAMS Configuration in YAML Format. This can be used to implement user-specific default option settings that are shared between different GAMS versions. In the future, we also expect to use this capability to support the setting of options and defaults that support backward compatibility: this will become important as we change GAMS defaults in significant ways to modernize the user experience in the light of current computing environments and user expectations.

- Added system suffixes `%system.licenseFileName%`, `%system.userConfigDir%`, and `%system.userDataDir%`.
- Embedded Python Code Facility:
  - Embedded Python code can be used with Python 3.6, 3.7, and 3.8. `GMSPython` is used for embedded code by default on the core platforms.
  - The GAMS command line parameter `pySetup` has been deprecated. The environment variable `GMSPYTHONLIB` that needs to point to the Python library (e.g. `python38.dll` on Windows) determines the Python installation to be used in embedded Python code.
  - The `gams.set` method to write data back to GAMS has a new argument `mapKeys` that allows to pass a callable to remap the elements of the key.
- Removed GAMS return code 116 ("Could not find GMSPython").
- Fixed the `card` operator for scalar symbols, so that it always returns 1 when the symbol has data. In the past it might have been wrong for symbols that had no data in the declaration, but just from an assignment.
- Fixed issues where GAMS would crash on exceptionally large models (more than 268 million rows or columns) with certain non-default options selected.
- Fixed a problem which allowed multiple declarations of the same equation or variable when $offMulti was active.
- Fixed some 3-argument intrinsic functions whose third argument was optional (e.g. `ncpVUSin`) - these were not taking the default value for the third argument correctly into account when called with two arguments.
- Fixed a bug which caused wrong execution of $onPut and embeddedCode after Programming Flow Controls that were not terminated with a semicolon.
- Fixed a bug which caused variable and equation records fixed at 0 to be dumped incorrectly with dumpOpt.
- Fixed incomplete warning when writing a non-default status of $on/offEmpty to a save file.
- execMode did not prevent the execution of scriptExit. This has been fixed.

GMSPython

- GMSPython has been upgraded from Python 3.6 to Python 3.8 and comes without Python package installer `pip`.
- Added the package `ply`.

Libinclude

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

Documentation

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

Windows Installer

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

3.18.1.4 Solvers

ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2020.0 on Windows.

BARON

- New libraries 20.4.14.
  - Improved local search strategies.
  - Increased effectiveness of cut pool manager.
  - Improved relaxations for MIQPs.
  - Added new LP subsolver based on HSL's LA04.

CBC

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

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

CONOPT

- New libraries 3.13L.
  - Added new facilities to print the variables changed by the pre-processor. See options prprec, rtpprec, and rtprel.
- New libraries 4.19.
  - Some errors related to the duals and basic/nonbasic flags for some definitional constraints and forcing constraints have been corrected.
  - The two tolerances `Tol_IFixed` and `Tol_DFixed` for when a variables is fixed have been unified into a single tolerance, Tol_Fixed.
  - The routine for detecting definitional constraints has been simplified and some options from previous versions are no longer used (`Flg_DC_Unique`, `Lim_DFVars`, and `Tol_Def_Ini`) and Tol_Def_Mult has a new default value.

CONVERT

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

COUENNE

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

CPLEX

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

GUROBI

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

IPOPT

- New libraries 3.13.2.
  - MA28 can no longer be loaded at run-time from a user-provided HSL library. Removed option 'ma28_pivtol'.
  - The default of option ma77_order has changed from 'amd' to 'metis'.
  - The default of option ma97_print_level has changed from 0 to -1.
  - The default of option mumps_pivot_order has been changed from 7 to 5 to work around a bug in MUMPS.
- Updated MUMPS to version 5.3.1. MUMPS has been build with OpenMP support enabled.
- Updated METIS (used by HSL codes and MUMPS) to version 5.1.0.

KESTREL

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

Lindo/LindoGlobal

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

LocalSolver

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

MOSEK

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

SCIP

- New libraries 7.0.0 (0bc4dc9c65).
  - The symmetry code has been completely restructured and modularity has been increased.
    - It is now possible to use orbitopes (i.e., polyhedral symmetry handling techniques) and orbital fixing on the same instance.
    - Branching decisions can now take symmetry information into account by filtering first variables from orbits (new parameter branching/relpscost/filtercandssym) and transfering pseudo cost information to variables in orbit (new parameter branching/relpscost/transsympscost).
    - Improved performance of orbital fixing and orbitope propagation.
    - Reduced memory usage of symmetry detection.
    - Improved detection of packing/partitioning orbitopes.
    - New possible value 3 for misc/usesymmetry and changed default from 2 to 3.
    - Renamed parameter `propagating/orbitalfixing/enableafterrestart` to propagating/symmetry/recom which decides whether to recompute symmetries after a restart or not.
    - Changed default of constraints/orbitope/propfreq from 5 to 1.
    - Changed default of constraints/orbitope/sepafreq from 5 to -1.
    - Removed parameters `constraints/orbisack/checkalwaysfeas`, `constraints/orbitope/checkalwaysfe` `constraints/symresack/checkalwaysfeas`.

  * Removed parameters `presolving/symmetry/maxgenerators`, `presolving/symmetry/checksymmetries`, and `presolving/symmetry/displaynorbitvars` and added various new parameters in group propagating/symmetry.
  * Removed parameters `presolving/symbreak/conssaddlp`, `presolving/symbreak/addsymresacks`, `presolving/symbreak/computeorbits`, `presolving/symbreak/detectorbitopes`, and `presolving/symbreak/addconsstiming`.
  * Removed parameters `propagating/orbitalfixing/symcomptiming`, `propagating/orbitalfixing/perfo` and `propagating/orbitalfixing/recomputerestart`.
  * New parameter `constraints/orbitope/usedynamicprop` to enable propagation of orbitope constraints by reordering the rows based on the branching strategy.
  * New parameter constraints/symresack/checkmonotonicity to enable upgrade to packing/partitioning symresacks even if the underlying permutation is not monotone.
  * New parameters constraints/orbisack/forceconscopy, constraints/orbitope/forceconscopy, and constraints/symresack/forceconscopy.

- Presolve:
  * The presolve library PaPILO (Parallel Presolve for Integer and Linear Optimization) 1.0 has been integrated as additional presolver for mixed-integer linear programs.
  * Expressions of form $|x|^p x$ in a nonlinear constraint are now sometimes recognized and handled by abspower constraints.
  * Some more quadratic constraints are now recognized as second-order cone constraints.
  * New presolver dualsparsify that tries to combine variables to cancel nonzero coefficients in the constraint matrix of a MIP.
  * Improved performance of tworowbnd presolver. New parameters added. Changed default of presolving/tworowbnd/priority from -500000 to -2000.
  * Extended presolver dualinfer by the ability to perform convex combinations of continuous variables to calculate better bounds for the dual variables. New parameters added.
  * Huge clique tables are now avoided by restriciting the number of nonzero entries relative to the number of nonzeros in the problem. New parameter presolving/clqtablefac.
  * New parameter constraints/linear/extractcliques to turn clique extraction off.
  * Improved presolver "domcol" to not require a complete representation of all constraints in the matrix.
  * Enabling aggressive presolving now activates all available presolving plugins, and decreases parameter presolving/restartfac correctly with respect to default.
  * New parameter constraints/linear/maxdualmultaggrquot and constraints/linear/maxmultaggrquot to limit the maximum coefficient dynamism of an equation on which multiaggregation is performed. Smaller values make multiaggregations numerically more stable.
  * Changed default of `constraints/quadratic/empathy4and` from 0 to 2. This leads to stronger but larger reformulations for products of binary variables.

- Primal heuristics:
  * New Large Neighborhood Search heuristic "Trust region", available as both a standalone heuristic and a neighborhood inside of Adaptive Large Neighborhood Search.
  * New heuristic Adaptive Diving, which registers all publicly available dive sets from other diving heuristics, learns online which divesets reach the best score, and executes them more frequently.
  * New penalty alternating direction heuristic PADM that splits the problem into several sub-SCIPs according to a user-provided decomposition (see below). The sub-SCIPs are solved on an alternating basis until a feasible solution of the original problem is found.
  * The GINS heuristic can make use of a user-provided decomposition (see below) by selecting a block of variables that maximizes the potential, and randomly selecting a start variable for the neighborhood and/or by selecting an interval of consecutive blocks as neighborhood, until fixing rate is reached. In this case, no variable is randomly selected. Several new parameters added for the use of a decomposition in GINS.
  * The LP face heuristic now wastes less time if it decides not to run.
  * Improved performance of Adaptive Large Neighborhood Search heuristic on merely continuous problems.

  - All primal heuristics that use sub-SCIPs are now disabled within the heuristics fast emphasis setting.
  - Replaced parameter `heuristics/localbranching/useuct` by heuristics/useuctsubscip, which affects all LNS heuristics.
  - New parameters heuristics/shiftandpropagate/minfixingratelp and heuristics/locks/minfixingratelp to stop the heuristics after propagating integer fixings if no sufficient fixing of the all variables (including continuous) could be achieved.
  - Changed default value of heuristics/coefdiving/freq from 10 to -1.
  - Changed default value of heuristics/conflictdiving/freq from -1 to 10.
  - Changed default value of heuristics/conflictdiving/lockweight from 1.0 to 0.75.
  - Changed default value of heuristics/rins/nodesquot from 0.1 to 0.3 (due to other code changes, this should not affect default behavior).
  - New value `1` available for heuristics/gins/potential to allow computation based on local LP solution.
  - The display column for memory usage ("mem") now shows the creator name for every new incumbent solution. The heuristic display characters now represents the type of the heuristic (diving, Large Neighborhood Search, propagation, etc.). Changed default of display/width from 139 to 143.

- Branching:
  - New rule vanillafullstrong, mostly for scientific purpose.
  - Improved branching point selection when branching on nonlinear variables. Instead of using exactly the LP solution, a point closer to the middle of the variables domain is chosen. Added parameters branching/midpull and branching/midpullreldomtrig.
  - Many updates to parameters of lookahead branching rule.
  - New parameter branching/relpscost/degeneracyaware to switch degeneracy-aware hybrid branching, which adjusts weights of different scores in relpscost (hybrid) branching rule based on degeneracy information and skips strong branching for very high degeneracy rates.
- Separation:
  - Improvements in cut presolving and cgmip separator.
  - New parameter separating/cgmip/genprimalsols that allows to generate initial primal solutions from Gomory cuts.
  - Changed default of separating/rapidlearning/freq from -1 to 5. Various new parameters for rapidlearning.
  - Changed default of `separating/efficacyfac` from 1.0 to 0.6.
- Approximations on the search tree completion and estimates on the tree size have been added.
  - The approximate search tree completion is shown in a new display column.
  - The tree size estimate is used to trigger a restart if, after a reasonable initialization, the estimated size of the remaining tree is large.
  - New parameter group branching/treemodel to specify the tree model.
- SCIP can now store user decompositions. A GAMS user can specify one decomposition via the `.stage` variable suffix. Statistics regarding the decomposition are printed to the log before presolving (if display/statistics is enabled) and after presolving. New parameter decomposition/maxgraphedge.
- Extended the dual proof analysis for infeasible LPs to consider also locally valid rows (new parameter conflict/uselocalrows).
- If a reference value is given, the primal-reference and reference-dual integrals are calculated automatically and printed within the SCIP statistics.
- Emphasis settings for numerically challenging instances have been added. They increase numerical stability of (mostly) presolving operations such as (multi-)aggregations at the cost of performance.
- Renamed parameters `misc/allowdualreds` and `misc/allowobjprop` to misc/allowstrongdualreds and misc/allowweakdualreds, respectively.

  - Replaced parameter `numerics/lpfeastol` by numerics/lpfeastolfactor to specify which factor should be applied to the SCIP feasibility tolerance to initialize the primal feasibility tolerance of the LP solver.
  - New parameter lp/minmarkowitz to set the Markowitz stability threshold. High values sacrifice performance for stability.
  - Changed default of propagating/redcost/useimplics from TRUE to FALSE.
  - See also the `full release notes` and the `release paper`.
- Symmetry detection and handling is now available on Windows, too.
- Updated Ipopt to 3.13, see above.
- Removed value `soplex2` for option lp/solver. `soplex` is now the default if no CPLEX license is available.

SHOT

- New solver by Andreas Lundell (Åbo Akademi University, Finland) and Jan Kronqvist (Imperial College London, UK).
- SHOT is a deterministic solver for mixed-integer nonlinear programming problems (MINLPs). It can solve convex and some nonconvex MINLP problems to global optimality. For other nonconvex problems, SHOT acts as a heuristic method, i.e., without providing guarantees of global optimality. For more information, see the GAMS/SHOT solver manual, the `SHOT website`, and the publications [126] [122].
- SHOT 1.0.0 (527f1a11) is available for Linux, macOS ($\geq$ 10.13), and Windows. It can be used with a GAMS base system license.

SoPlex

- New libraries 5.0.0 (6535a3c8).
  - Improvements in memory management.
  - The number of violations is now shown in the log.
  - New options bool:forcebasic, int:stattimer, and real:min_markowitz. Replaced option `int:printcondition` by int:printbasismetric.
- GAMS/SoPlex is now available for multithreaded solves.

XPRESS

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

3.18.1.5 Tools

GAMS IDE and GDXVIEWER

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

GAMS Studio

- New version 1.1.0.
  - Stability improvements.
  - Introduced new settings format.

    Attention

    This required a reset of all settings. They are saved now in JSON format. As a consequence, Studio 1.0.0 cannot use settings that were written by Studio versions prior to 1.0.0 and vice versa.
  - Changed the default file encoding to UTF-8 for all platforms (this mostly affects Windows, there should be no change for other platforms).
  - Added option to change the default text encoding to `Edit -> Encoding -> Select encodings`.
  - Introduced Dark Theme for Windows and Linux as well (can be changed through `Settings -> Editor`).
  - Added editor for GAMS Configuration and menu entry for default GAMS Configuration file.
  - Added option to change display format of numerical values in GDX Viewer.
  - Changed location of default workspace and model library.
  - Changed location where `gamslice.txt` is put when created using the "About GAMS" dialog to the new standard locations searched by GAMS (see installation notes for details).
  - Improved shortcut layout:
    - Reintroduced "Open Terminal" shortcut as `Ctrl+T`.
    - Introduced shortcut for "Run MIRO base mode": `F8`.
    - Introduced shortcut for "Run MIRO hypercube mode": `Shift + F8`.
    - Introduced shortcut for "Run MIRO configuration mode": `Shift + F7`.
    - Changed shortcut for focus "Project Explorer" to `Ctrl + 1`.
    - Changed shortcut for focus "Editor" to `Ctrl + 2`.
    - Changed shortcut for focus "Parameter Editor" to `Ctrl + 3`.
    - Changed shortcut to extend and collapse "Parameter Editor" to `Ctrl + Alt + 3`
    - Changed shortcut for focus "Tab Browser" to `Ctrl + 4`.
    - Changed shortcut for focus "Process Log" to `Ctrl + 5`.
    - Changed shortcut of duplicate line to `Ctrl+D`.
    - Changed shortcut of "Toggle Comment" in "Parameter Editor" to `Ctrl + *`
    - Changed shortcut of action "Go to matching parentheses" to: `Ctrl + B`
    - Changed shortcut of action "Select to matching parentheses" to: `Ctrl + Shift + B`
    - Fixed shortcut on macOS: block edit mode is now `CMD+Shift+Arrow Key`.

GDXMERGE

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

GDXXRW

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

3.18.1.6 Object Oriented APIs

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

C++

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

Python

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

3.18.1.7 Expert Level APIs

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

3.18.1.8 Model Libraries

GAMS API Library

- All Python examples have been reworked. Tests are performed also with the Python installation in `GMSPython`.
- New model **PBuildXPLevelAPI.gms : Test building and installing the GAMS Python API from source distribution** (57).

GAMS Data Library

New models:

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

GAMS Model Library

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

GAMS Test Library

New models:

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

PSOPT Model Library

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

3.18.1.9 Solver/Platform availability matrix

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP 2.10	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓		
BARON	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓		
CBC 2.10	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓		
CPLEX 12.10	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	
DICOPT 2	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓		
GUROBI 9.0	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓
IPOPT 3.13	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓
KNITRO 11.1	✓	✓	✓		
LGO	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓		
LINDOGLOBAL 12.0	✓	✓	✓		
LOCALSOLVER 9.5	✓	✓	✓		
MILES	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓
MOSEK 9	✓	✓	✓		
MSNLP	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓
ODHCPLEX 4	✓	✓			
PATH	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓
SCIP 7.0	✓	✓	✓		
SHOT 1.0	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓		
SOPLEX 5.0	✓	✓	✓		
XA	✓	✓			
XPRESS 35.01	✓	✓	✓	32.01	29.01

3.18.2 31.1.1 Maintenance release (May 16, 2020)

3.18.2.1 Acknowledgments

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

3.18.2.2 GAMS System

GAMS

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

GAMS Data Exchange (GDX)

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

3.18.2.3 Solvers

GUROBI

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

MOSEK

- New libraries 9.2.7.

3.18.2.4 Tools

GAMS Studio

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

3.18.2.5 Object Oriented APIs

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

3.18.2.6 Model Libraries

GAMS Test Library

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

3.18.3 31.2.0 Minor release (June 19, 2020)

3.18.3.1 Acknowledgments

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

3.18.3.2 GAMS System

GAMS

- The demo/community license limit of indexed external in- and output symbols for the GAMS/MIRO connector has been increased from 5 to 10. There is no limit on *scalar* external in- and output symbols. Starting with this version, one dimensional singleton sets are also not limited.
- Extend the influence of the command line parameter multiPass, so that it also ignores errors about unknown dimension with empty data statements when set to 2.
- Added three new put_utility commands `incMsg`, `incLog`, and `incMsgLog` to include the content of a file into the log or listing file.
- Fixed a crash for CNS models which get solved asynchronously while asyncSolLst is set to `1`.

3.18.3.3 Solvers

Gurobi

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

LocalSolver

- New libraries 9.5 (20200429).

Mosek

- New libraries 9.2.10.

SCIP

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

SHOT

- New library 1.0.1.

3.18.3.4 Tools

GAMS Studio

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

3.18.3.5 Object Oriented APIs

.NET

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

3.18.3.6 Expert Level APIs

C#

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

3.19 30 Distribution

3.19.1 30.1.0 Major release (January 10, 2020)

3.19.1.1 Acknowledgments

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

3.19.1.2 Platforms

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

3.19.1.3 Licensing and Demo Limits

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

3.19.1.4 GAMS System

GAMS

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

Installer

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

3.19.1.5 Solvers

BARON

- New libraries 19.12.7.
  - Performance improvements for large-scale and mixed-integer quadratic problems.
  - Improvements in interface to IPOPT.

CBC

- New libraries.

CONOPT

- New library 4.16 with bug fixes.
  - Improvement in the transition from Phase 0 to Phase 1 and in the one-dimensional search close to optimality.

CPLEX

- New libraries 12.10.0.0.
- New options epLin and heuristicEffort.

GUROBI

- New libraries 9.0.0.
  - Gurobi 9 comes with a new bilinear solver, which allows to solve non-convex quadratic programming problems. The new parameter nonConvex needs to be set.
- New general constraint types have been added to model functions like log, exp, sin, cos, and tan. New options funcPieces, funcPieceError, funcPieceRatio, funcPieceLength, and funcMaxVal have been added to control the piece-wise-linear discretization. The dot options doFuncPieces, doFuncPieceError, doFuncPieceRatio, and doFuncPieceLength allow control on an individual constraint basis.
- New parameters for new cutting planes: BQPCuts, relaxLiftCuts, and RLTCuts.
- Fixed a problem when reporting the primal solution only.
- Report the dual bound for continuous non-convex QP and QCP problems.

JAMS/LogMIP

- New option userPFFile to provide additional GAMS command line parameters for the GAMS run of the reformulated model.

LocalSolver

- New libraries 9.0 (20191219).

MOSEK

- New libraries 9.1.9.

ODHCPLEX

- New libraries 4.3.2 with bug fixes and performance improvements.

PATH

- New library 5.0.01 with bug fixes.
  - Fix seg fault when using the hidden option output_hidden_options.

SCIP

- New libraries 6.0 (90a56e0b).
  - Default for option heuristics/rins/nodesquot changed from 0.1 to 0.3.

3.19.1.6 Tools

GAMS Studio

- New version 0.14.1.
  - Stability improvements.
  - Added GDX Diff dialog to compare two GDX files. It can be opened by choosing `Tools` > `GDX Diff` from the menu or through the context menu off a GDX file in the Project Explorer.
  - Allow to open OS specific terminal in current working directory (`Tools` > `Terminal`) or in working directory of a specific group (using the context menu in the Project Explorer)
    - Note: This feature does not work on all Linux distributions, e.g., Ubuntu, yet.
  - Added basic integration of `GAMS MIRO`.
    - Note: This requires a separate installation of the GAMS MIRO package.
  - Improved GDX Viewer layout to optimize used screen space.
  - Adjusted layout of Reference File Viewer.
  - Added clear button to filter input field in GDX Viewer and Reference File Viewer.

3.19.1.7 Object Oriented APIs

C++

- We plan to drop support for Microsoft Visual Studio 2013 and maybe also 2015 with the next major release.

Python

- Added support for Python 3.8.
- As announced, dropped support for Python 3.4.

3.19.1.8 Expert Level APIs

- Added support for Python 3.8.
- As announced, dropped support for Python 3.4.

3.19.1.9 Model Libraries

GAMS Data Library

New models:

- **pickstock.gms : Stock selection problem with MIRO** (133)
- **transport.gms : Classical transportation problem with MIRO** (134)

GAMS Test Library

New models:

- **embpy05.gms : Test merge/replace when loading data from embedded code** (804)
- **embpy06.gms : Test domain check/filtered when loading data from embedded code** (805)
- **load12.gms : Tests filtered load** (806)
- **idc01.gms : Implicit Data Contract Test** (807)
- **gurobi05.gms : GUROBI test suite - general constraints sin/cos** (808)
- **studio01.gms : Test Studio Startup** (809)

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX

3.19.1.10 Solver/Platform availability matrix

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	18.5.8	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.10	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.10	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 9.0	7.5	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 11.1	11.0	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓	✓		
LINDOGLOBAL 12.0	✓	✓	✓	✓		
LOCALSOLVER 9.0	8.5	✓	✓	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 9	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
ODHCPLEX 4		✓	✓			
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 6.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 4.0	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 33.01	✓	✓	✓	✓	32.01	29.01

3.19.2 30.2.0 Minor release (February 07, 2020)

3.19.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Iswar Radhakrishnan, Simon Scolan, and Kirk Yost.

3.19.2.2 GAMS System

GAMS

- Fixed a problem when writing implicit data contract in JSON format (IDCGenerateJSON) with symbols that use a set more than once in its domain.
- Fixed a potential wrong domain violation error for symbols declared under $onExternalInput.
- Fixed `gamsinst` and the IDE to give better messages in case a license file is absent.
- GAMS incorrectly removed indexed semi-integer and semi-continuous variables that have been fixed at a level other than zero with holdFixed set to 1. This has been fixed.
- GAMS also removes scalar semi-integer and semi-continuous variables now, when they have been fixed at zero with holdFixed set to 1.
- Fixed an issue where protected symbols could be changed using certain option statements.
- Added new command line parameter IDCProtect to control assignment behavior of external input symbols at execution time: `0` allows assignment, `1` (default) protects external input symbols from being changed.
- Adjusted reading of GAMS command-line arguments on non-Windows platforms to allow for empty arguments and arguments with preceding and trailing blanks. Windows users will not be affected by this change, nor will users who do not use quotes to protect arguments. But these invocations will now work differently on Unix, and will give the same result as on Windows:
  - `gams main.gms --myPrefix "" user1 " four blanks here "`
  - `$call gams subjob.gms --cityA Tokyo --cityB ""`

3.19.2.3 Solvers

CONOPT

- New library 4.17 with bug fixes.
  - Fixed a problem where small infeasibilities were not correctly reported back to GAMS.

DICOPT

- Fixed a possible crash during exit when having used Ipopt as NLP solver.

GUROBI

- New libraries 9.0.1.
- Fixed a problem with tuning of lp/mps files. In contrast to the documention, the tuning suite contained the model generated by GAMS when providing a list of models in lp/mps files to tune.
- Made value 3 available for Gurobi option scaleFlag, available since Gurobi 7.5.

MOSEK

- New libraries 9.1.11.

PATH

- New library 5.0.02.
  - Fix to two preprocessor issues: wrong solutions were being returned.

3.19.2.4 Tools

GAMS Studio

- New version 0.14.3.
  - Stability improvements.
  - New icon design.
  - Added filter facility for value columns in GDX Viewer.

3.19.3 30.3.0 Minor release (March 06, 2020)

3.19.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Andres J. Calderon, Antti Lehtila, Grégoire Pichenot, and Karlo Šepetanc.

3.19.3.2 GAMS System

GAMS

- Allow to create a secure work file on any platform for any target license platform.
- Fixed a bug which could have caused wrong domain violation errors with $offEmbeddedCode.
- Symbols set in embedded Python code via the GAMS Python object oriented API (i.e. `gams.db[sym]`) were imported with `MergeType` "replace" and `DomainCheckType` "filtered". This has been adjusted to both `MergeType` and `DomainCheckType` set to "default".
- Fixed a problem which was introduced with GAMS 30.1: The default for $(on|off)Filtered should always be `$onFiltered`. This was not the case when restarting from an old work file.

3.19.3.3 Solvers

Antigone, Bonmin, Cbc, Couenne, Ipopt, SCIP

- The number of threads as specified by option threads was disregarded. This has been fixed.

CONOPT

- New library 4.18 with bug fixes.
- The degeneracy-breaking logic has been improved for numerically difficult models and the interaction between defined variables and initialized variables has been improved.

JAMS/LogMIP

- The maximum length of the directory name plus file name that GAMS can execute is 255. JAMS and LogMIP stopped working earlier. This has been fixed.

LocalSolver

- Added missing `lskeygen` tool for Linux system.

MOSEK

- New libraries 9.1.13.

3.19.3.4 Tools

GAMS Studio

- New version 0.14.4.
  - Stability improvements.
  - Introduced Dark Mode support for macOS.

3.20 29 Distribution

3.20.1 29.1.0 Major release (November 15, 2019)

3.20.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, David Bernal Neira, Erwin Kalvelagen, and Scolan Simon.

3.20.1.2 Platforms

- As announced, dropped support for macOS 10.12 (Sierra).
- After more than 20 years we plan to drop support for 32-bit MS Windows with the next major release. Some components, e.g., GAMS IDE and gdxxrw, will remain 32-bit executables in the 64-bit MS Windows GAMS distribution.
- We plan to drop support for MS Windows 7 with the next major release.

3.20.1.3 GAMS System

GAMS

- Extended the GAMS log to print out the Double Dash Parameters set in addition to the non-default Command Line Parameters.
- Added dollar control option $setDDList.Cont to allow to define a list of double dash parameters to be checked over multiple lines.
- Added new ways to execute external programs which implicitly check the errorLevel returned and raise a compilation or execution error if it is not zero, namely $call.checkErrorLevel, $hiddencall.checkErrorLevel, execute.checkErrorLevel, put_utility exec.checkErrorLevel and put_utility shell.checkErrorLevel. This behavior can also be triggered without using the `checkErrorLevel` suffix by setting $onCheckErrorlevel and checkErrorLevel, respectively.
- Added command line option captureModelInstance to capture all model instances generated and solved by a solve statement during a GAMS run.
- Allow to declare a parameter as table without entering data.
- Added new Dollar Control Options eval.Set, evalGlobal.Set, and evalLocal.Set to define a compile-time variable based on the content of a GAMS set.
- Removed limit of 10,000 columns in table declaration.
- Changed trace record fields of quadratic model types from historic to today's names (QP --> QCP, QMIP --> MIQCP, RQMIP --> RMIQCP).
- The `gams.exe` and `gamskeep.exe` programs on 64-bit Windows are now native 64-bit executables. Because of some technical requirements related to the Interrupt functionality, the IDE invokes GAMS via the new 32-bit program `gams32.exe`.
- Fixed wrong return string for `%gams.action%`.
- Fixed a problem with interval evaluation for Extrinsic Functions.
- Fixed a problem with GAMS parameter scriptExit when running from the object-oriented APIs, GAMS Studio, or GAMS IDE.
- Fixed a problem with trace record field ETSolve which reported `NA` always.
- Fixed a problem with potentially wrong explanatory text when implicit sets are loaded using $load.

Installer

- The DMG installer for macOS has been reorganized and is now notarized, so that it should work on macOS 10.15.
- The selfextracting archive for macOS is now a 64-bit application.

3.20.1.4 Solvers

ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2019.5 on Linux and macOS.
- Updated Intel MKL libraries to version 2018.4 on Windows 64-bit again.

CBC

- New libraries.

CONOPT

- New library 4.15 with bug fixes.
- Option `DF_Method` has been replaced by the options Tol_Def_Mult and `Tol_Def_Ini`.

Convert

- Fixed a bug that caused a crash with option `nlp2mcp` selected.

DICOPT

- Fixed so called "integer cuts" when both binary and integer variables are present. DICOPT generated two integer cuts in this case, one that excluded the current values of binary variables and another one that excluded the current values of integer values. Now a single combined cut is generated, which allows that the values of the binary variables can repeat if the values of the integer values change and vice versa.
- Setting option weight to a value larger 10^{20} now disables the augmented penaly relaxation, that is, linearizations of nonlinear equations are added as hard constraints to the MIP relaxation instead of soft constraints.
- Enabling option convex now changes the default for option weight to `maxdouble`.

Examiner

- Added option objvarAutoAdjust to adjust objective variable to make objective equation feasible for models that allow objective reformulation.

Lindo/LindoGlobal

- New libraries 12.0.169.
- Added support for function `signPower`.

LocalSolver

- New libraries 9.0 (20191004) for Linux, macOS, and 64-bit Windows.
  - Performance improvements on packing, routing, and scheduling problems as well as on nonlinear continuous problems.

LGO

- Fixed a bug in the computation of the best feasible solution - an uninitialized variable was used.

MOSEK

- New libraries 9.1.1.
- Correct rejection of problems with non-quadratic nonlinear objective function when constraints are linear.

SoPlex

- New libraries 4.0 (083219e0).

SCIP

- New libraries 6.0 (a15f6c41).

3.20.1.5 Tools

GAMS Studio

- New version 0.13.2
  - Stability and performance improvements.
  - Allow to adjust the numerical precision in the GDX Viewer.
  - Added dialog to create/update license file (pops up when a license is in the user's clipboard and "About GAMS" is opened).
  - Added full screen mode (shortcut on Windows and Linux: `ALT+Return`, shortcut on macOS: `META+CMD+F`).
  - Added auto-resizing of columns when opening the Reference File Viewer.
  - Reworked process log to better handle long and fast output.
  - Improved behavior of `F1` (Help) Key: Studio now opens widget specific help if available.
  - Renamed "Output View" to "Process Log".
- Added a tutorial about the usage of GAMS Studio to the documentation.
- As announced, dropped Studio for 32-bit Windows.

3.20.1.6 Object Oriented APIs

Python

- Added support for Python 3.7 on Windows 64-bit, Linux, and macOS.
- We plan to drop support for Python 3.4 with the next major release.
- Support for Python 2.7 officially stops January 1, 2020. We plan to drop support for Python 2.7 with some future release. If you rely heavily on the availability and support of the GAMS API for Python 2.7 please contact support@gams.com to discuss your options.

Java

- Increase the minimum version requirement of the Java Runtime Environment to Java SE 8.
- The remove operation **GAMSDatabaseIterator.remove** is no longer supported. The method now throws an instance of `java.lang.UnsupportedOperationException` and performs no other actions. (To remove all records of the current iterating symbol, use the method **GAMSSymbol.clear** instead.)
- Fixed the behavior of the remove operation **GAMSSymbolIterator.remove**. The method now removes all records of the last GAMSSymbol element returned by an instance of **GAMSSymbol-Iterator** and can be called only once per call to **GAMSSymbolIterator.next**. The behavior of an iterator is unspecified if the underlying collection is modified while iterating.

3.20.1.7 Expert Level APIs

- Added support for Python 3.7 on Windows 64-bit, Linux, and macOS.
- We plan to drop support for Python 3.4 with the next major release.
- We plan to drop support for Python 2.7 with some future release.

3.20.1.8 Model Libraries

GAMS Model Library

- Added **inscribedsquare.gms : Inscribed Square Problem**.

GAMS Test Library

- Extrinsic function test models: Upgraded query-library generation code ql.py to Python 3. The default for property "description" has been changed to be an empty string.

3.20.1.9 Solver/Platform availability matrix

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	18.5.8	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.10	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.9	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 8.1	7.5	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 11.1	11.0	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓	✓		
LINDOGLOBAL 12.0	✓	✓	✓	✓		
LOCALSOLVER 9.0	8.5	✓	✓	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 9	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
ODHCPLEX 4		✓	✓			
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 6.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 4.0	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 33.01	✓	✓	✓	✓	32.01	29.01

3.21 28 Distribution

3.21.1 28.1.0 Major release (August 02, 2019)

3.21.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Johan Löfberg, Hans Mittelmann, and José Luis Vilar Zanón.

3.21.1.2 Platforms

- We plan to drop support for macOS 10.12 (Sierra) with the next GAMS major release.

3.21.1.3 GAMS System

GAMS

- Added new dollar control option $scratchFileName.
- Fixed a bug which caused a problem when unloading a multi-dimensional Alias at compile-time.

Documentation

- Improved ranking of results in `offline search` by minimizing the effect of the length of a document on the importance of a search term in that document.
- When searching the `index`, only relevant results are shown for subcategories as well.
- The `full-text search` now shows also results for a search in the `index`.

3.21.1.4 Solvers

ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Downgraded Intel MKL libraries to version 2018.3 on Windows 64-bit.

ANTIGONE

- The timelimit is now applied to the wallclock time and wallclock time is reported in the log and back to GAMS.

BARON

- New libraries 19.7.13.
  - New relaxations and branching rules for nonconvex mixed-integer quadratic programs, leading to significant speedups for this class of problems.
  - Increased robustness in various range reduction routines.
  - Increased solvability of large-scale problems by BARON's cut generators.
  - Enhanced convexity recognition routines.
  - Updated to IPOPT 3.12.13 and CBC 2.10.1.
- Fixed passing constant equations to BARON.

CBC

- Revised solver options.
  - Updated the documentation and possible values for many options, bringing them closer to the original Cbc option names and values.
  - The following options changed their type from boolean to string: crossover, cutoffConstraint, combineSolutions, Dins, DivingRandom, DivingCoefficient, DivingFractional, DivingGuided, DivingLineSearch, DivingPseudoCost, DivingVectorLength, feaspump, naiveHeuristics, pivotAndFix, randomizedRounding, Rens, Rins, roundingHeuristic, proximitySearch, presolve
  - For options dualPivot, primalPivot, and scaling value "auto" has been renamed to "automatic".
  - For option costStrategy, values "binaryfirst" and "binarylast" have been renamed to "01first" and "01last", respectively.
  - The following options have been added: factorization, denseThreshold, smallFactorization, sparseFactor, biasLU, psi, substitution, primalWeight, autoScale, bscale, gamma, KKT, expensiveStrong, OrbitalBranching, infeasibilityWeight, fixOnDj, sosPrioritize, cutLength, lagomoryCuts, latwomirCuts, hOptions, combine2Solutions, diveSolves, feaspump_artcost, feaspump_fracbab, feaspump_cutoff, feaspump_increment, dwHeuristic, pivotAndComplement, VndVariableNeighborhoodSearch

CONOPT

- CONOPT3 accepts models with empty rows. The GAMS/CONOPT3 link sets the hidden option `LSAZRW` to true.

Convert

- Replaced the OSiL writer. The new writer supports more GAMS intrinsic functions and produces slightly smaller OSiL files with more compact expression trees.

IPOPT

- Fixed handling of Ipopt's feasible-point-found status.

LocalSolver

- Added option `logfreq`.

MOSEK

- New libraries 9.0.98.
  - Introduced the power and exponential cones, see also Conic Programming.
  - As announced, the optimizer for general convex nonlinear problems has been removed. Thus, solving a NLP, DNLP, MINLP, or RMINLP for which a conic reformulation is not recognized will now result in a missing capability solver status with GAMS/Mosek.
  - Improved presolve for particular conic problems.
  - Tighten the stopping criteria when solving conic optimization problems: Changed default of MSK_DPAR_INTPNT_CO_TOL_INFEAS from 1e-10 to 1e-12, of MSK_DPAR_INTPNT_CO_TOL_REL_GAP from 1e-7 to 1e-8, and of MSK_DPAR_INTPNT_QO_TOL_INFEAS from 1e-10 to 1e-12. However, note that one can let Mosek relax feasibility and optimality tolerances if it cannot make sufficient progress, see MSK_DPAR_INTPNT_CO_TOL_NEAR_REL and MSK_DPAR_INTPNT_QO_TOL_NEAR_REL.
  - Changed scaling of interior-point optimizer so better accuracy is obtained in some cases.
  - Introduced an outer approximation method for solving conic mixed integer optimization problems. This can be enabled with the parameter MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION.
  - Much improved performance on recent AMD CPUs. For linear algebra, the BLIS library is employed when run on AMD CPUs.
  - Better performance on CPUs that support AVX-512 instructions.
  - Removed parameters: MSK_DPAR_DATA_TOL_AIJ, MSK_DPAR_INTPNT_NL_*, MSK_DPAR_MIO_DISABLE_TERM_TIME, MSK_DPAR_MIO_NEAR_TOL_ABS_GAP, MSK_DPAR_MIO_NEAR_TOL_REL_GAP, MSK_IPAR_MIO_CONSTRUCT_SOL and synonym "mipstart" (GAMS now always passes the starting point of a MIP to Mosek), MSK_IPAR_OPF_MAX_TERMS_PER_LINE, MSK_IPAR_WRITE_DATA_FORMAT
  - Further new parameters: MSK_IPAR_PRESOLVE_MAX_NUM_PASS, MSK_IPAR_PRESOLVE_MAX_NUM_P
    MSK_IPAR_SIM_SEED, MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS, MSK_IPAR_MIO_MAX_NUM_RO
    MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT, MSK_IPAR_MIO_SEED,
    MSK_IPAR_MIO_FEASPUMP_LEVEL, MSK_IPAR_WRITE_COMPRESSION, MSK_IPAR_OPF_WRITE_LI
    MSK_IPAR_PTF_WRITE_TRANSFORM
  - Changed default of MSK_DPAR_MIO_TOL_ABS_GAP from 0 to value of GAMS option OptCA (was used for MSK_DPAR_MIO_NEAR_TOL_ABS_GAP before) and of MSK_DPAR_MIO_TOL_REL_GAP from 0 to value of GAMS option OptCR (was used for MSK_DPAR_MIO_NEAR_TOL_REL_GAP before).
- Fixed handling of Mosek certificiates for primal and dual infeasibility, see Infeasible/Unbounded Models.

SCIP

- New libraries 6.0 (f79421d).

3.21.1.5 Tools

GAMS Studio

- New version 0.12.3.
  - Stability and performance improvements.
  - Added GAMS Solver Option Editor to view and edit a solver-specific option file.

  - Renamed Option Editor to Parameter Editor to better distinguish solver options and GAMS parameters.
  - Added menu entry `GAMS -> Delete scratch directories` to delete all scratch directories in current working directory. A similar dialog will also be triggered if GAMS returns `110` ("Too many scratch directories").
  - Added file associations and icons for `.gms` and `.gdx` files on macOS.
  - Changed shortcut to focus Main widget from `Ctrl+H` to `CTRL+E` (there was a collision on macOS).
  - Improved keyboard navigation between files and widgets.
- We plan to drop Studio for 32 bit Windows with the next major release.

3.21.1.6 APIs

- As announced, dropped support for Python 2.6.
- We plan to increase the minimum version requirement of the Java Runtime Environment to Java SE 8 with the next major release.

3.21.1.7 Model Libraries

GAMS Model Library

- Changed model sigma in **immun** to minimize standard-deviation instead of variance.
- Added missing nonnegativity requirement on variables in model **wall**.

3.21.1.8 Solver/Platform availability matrix

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	18.5.8	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.10	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.9	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
GUROBI 8.1	7.5	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 11.1	11.0	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓	✓		
LINDOGLOBAL 12.0	✓	✓	✓	✓		
LOCALSOLVER 8.5	✓	✓	✓	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 9	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
ODHCPLEX 4		✓	✓			
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 6.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 4.0	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 33.01	✓	✓	✓	✓	32.01	29.01

3.21.2 28.2.0 Minor release (August 19, 2019)

3.21.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Nayeem Chowdhury, Salvador Doménech Martínez, Michael Ferris, Antti Lehtila, Evangelos Panos, Jochen Uhrich, and Alexey Ziuzin.

3.21.2.2 GAMS System

GAMS

- Setting $onMultiR allows the redefinition of macros as well now.
- Fixed a problem finding the Python interpreter, that comes with GAMS, in certain cases.
- Fixed a bug in the model generation, which could have caused a crash or wrong results in some cases.

3.21.2.3 Solvers

CPLEXD

- Corrected reporting of solution for (MI)QCP problems when using feasopt.
- CplexD now reports the objective of the feasible relaxation in the solution summary when using feasopt.

3.21.2.4 Tools

GAMS Studio

- New version 0.12.4
  - Stability and performance improvements.
  - Improved robustness of glb file parser (see User Libraries).
  - Added "What's new" menu entry to show latest changes of Studio.

3.21.2.5 Object Oriented APIs

- Fixed a problem, where `GAMSDatabase.Export` triggered an exception because of domain violations, even if only relaxed domains were used.

Java

- Added new method **GAMSSymbolRecord.dispose** for on-demand release of external resources hold by non-java library.

3.22 27 Distribution

3.22.1 27.1.0 Major release (April 24, 2019)

3.22.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Arne Drud, Jan-Erik Justkowiak, Renger van Nieuwkoop, and Manuel Wetzel.

3.22.1.2 Platforms

- As announced, support for Mac OS X 10.11 (El Capitan) has been dropped.
- As announced, support for Windows Vista has been dropped.

3.22.1.3 GAMS System

GAMS

- Embedded Python Code:
  - Allow Implicit Set Definition (or: Domain Defining Symbol Declarations) now for symbols loaded from embedded code as well.
  - Changed the default behavior of $offEmbeddedCode and endEmbeddedCode: Previously, internal resources got freed and with the next start of an embedded Python code section, the Python environment had to be reinitialized. This is not the case anymore. Also, the interpreter stays "alive" now, which allows to access Python symbols defined in one embedded code block in a following block. The old behavior could still be activated by setting the new command line parameter `freeEmbeddedPython` to 1.
  - Changed the optional boolean argument `merge` to an enumerated option `mergeType` in the `gams.set` method in embedded code. *This might break existing code.* Even if the `merge` argument was not used, the default behavior at compile time changes. In previous versions the content of an already defined symbol was quietly replaced while GAMS now triggers a compilation error `194 Symbol redefined`.
  - Fixed a bug in embedded code when setting `pySetup` to 0. Additional steps are required on Linux and macOS to use an alternative Python installation. Details can be found in section `Porting to a Different Version of Python`.
  - A `GAMSDatabase` created in embedded code via `gams.ws.add_database(source_database = gams.db)` had all the symbols but no data. This has been fixed.
  - `gams.db.export(file_name)` in embedded code resulted in an error for scalar symbols without data. This has been fixed.
  - Fixed a problem with embeddedCode/continueEmbeddedCode after an assignment that was not finished with a `;` like in this example:

    ```
    Set i /1*3/;
    Parameter xl(i);
    xl(i) = uniform(0.5,1)
    embeddedCode Python:
    print(list(gams.get('xl')))
    endEmbeddedCode
    Display xl;
    ```

    This leads to an error now. In the past, the assignment in line 3 was executed **after** the `embeddedCode` block leading to unexpected results.
- Extended the GAMS log to print out the non-default Command Line Parameters.
- Improved performance of execute_unload when option gdxUELs is set to `full`.
- Improve performance of model generation when multi-dimensional variables are used, where the controlling set for dimension `n>1` is big, e.g.:

  ```
  Set tiny /1*2/, huge /1*100000/ ;
  Variable x(i,j); Equation e;
  ...
  e.. sum((tiny,huge), x(tiny,huge)) =e= 42;
  ...
  ```
- Fixed a potential crash when more than 250 million labels are used in one model.
- Fixed a bug with Implicit Set Definition: When used with $onMulti to merge data into a non-empty set, it could have happened that an element was added twice in some cases.
- Fixed a bug causing a broken dump file if $onVerbatim and $ifThen was used with DumpOpt.

Documentation

- Added documentation for the put_utility keyword `assignText` (which has been in the system since 24.6).

GDX library

- Improve the behavior of GDX when handling large data sets (e.g. writing a set with 210 million elements, each having its own lengthy set text). Performance is better - less memory and less time are required. The limits on the amount of data that can be handled have been extended somewhat, and when these limits are reached this is handled more gracefully than previously - with a helpful message instead of a segmentation fault or erroneous results.
- Introduce consistent handling of the empty string vis-a-vis set text for UELs. *This is only relevant for user applications that use the low-level GDX API in an unsupported way.*
  - Previously the proper usage was to avoid adding the empty string to the list of set text strings. Applications that broke this rule would produce a GDX file with an abnormal internal structure. This structure may lead to jumbled set text.
  - With the updated GDX, it is not necessary to avoid adding an empty string to the set text list.
  - Applications holding to previous proper usage (i.e. those that avoid adding the empty string to the list of set text strings) will produce good GDX files regardless of GAMS version, and these files will behave as expected regardless of the GAMS version in which they are used.

3.22.1.4 Solvers

ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2019.3 on Linux and macOS and to 2018.4 on Windows 64-bit.

BARON

- New libraries 19.3.24.
  - Improved automatic differentiator that is up to two orders of magnitude faster for large-scale problems.
  - Improved data structures and cut pool manager to reduce memory footprint and time requirements for large-scale problems.
  - Improvements in various components, including parallel threads and BARON's IPOPT interface for macOS.
  - New tree management algorithms.
  - Improved range reduction and probing algorithms.
  - Using Intel MKL for faster BLAS and LAPACK routines.
  - Ipopt now uses METIS for linear system partitioning.
  - New option FirstLoc to terminate BARON as soon as a local optimum is found.
- New option ClockType to determine whether wall-clock or CPU time should be reported back to GAMS.

BDMLP and SBB

- The definition of SOS1 and SOS2 variables has changed. BDMLP and SBB used to *count* the number of elements in a set that are *off the lower bound*. BDMLP and SBB were the only two solvers in the GAMS solver portfolio with this definition. All other solvers counted elements that are *off zero*. The definition of SOS1 and SOS2 variables in BDMLP and SBB has been changed to the *off zero* definition to make their behavior more consistent with other GAMS solvers.

CBC

- New libraries 2.10.
  - Improved handling of SOS, starting point, and symmetries.
  - Improved performance of primal heuristics regarding the handling of implicit integer variables.
  - Mini-B&B is now disabled when solving with multiple threads.
  - Changed default value for zero half cuts parameter from `off` to `ifmove`.

CONOPT 4

- New library 4.11 with improvements in reliability and efficiency for large and difficult models.
- New option Tol_Opt_LinF: Optimality tolerance when infeasible in Linear Feasibility Model.

CPLEX/CPLEXD

- New libraries 12.9.0.
  - New options: CPLEXfolding, CPLEXwriteprob (CplexD only), CPLEXwarninglimit.
  - New hierarchical multi-objective optimization (CplexD only): CPLEXmultobj, CPLEXmultobjmethod, CPLEXobjnabstol, CPLEXobjnreltol.
  - The symmetry breaking parameter CPLEXsymmetry now applies only to MIP models. Use the new CPLEXfolding parameter for LP models.

GUROBI

- New libraries 8.1.1.

JAMS

- Experimental indexed EMP syntax has been updated: It can now take all the equilibrium related keywords (max, min, vi, qvi, dualvar, visol, and implicit) and shared constraints and variables.

LocalSolver

- New libraries 8.5 (20190130).
  - Strong lower bounds based on nonlinear relaxation techniques coupled with innovative branch-and-bound heuristics.

Mosek

- New libraries 8.1.0.80.

ODHCPLEX

- New libraries 4.23.

OQNLP

- As announced, OQNLP has been dropped.

3.22.1.5 Tools

CSDP

- See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.
- New library 6.2.0 on macOS.

GAMS Studio

- New version 0.11.1.
  - Stability and performance improvements.
  - Introduced a toolbar to replace the "Execution and Option Parameter Editor".
  - Editor
    - Improved the syntax highlighting.
    - Allow to open huge read-only (lst) files without significant memory requirement.
    - Added bookmark support.
    - Added `Ctrl+Up`/`Ctrl+Down` as shortcut for line wise scrolling without moving the text cursor.
    - Added `Ctrl+Home`/`Ctrl+End` as shortcut for page wise scrolling without moving the text cursor.
    - Added warning pop-up when trying to open a >50 Mb file in edit mode.
  - Various
    - Added `Ctrl+Alt+L` to extend/collapse the extended option editor
  - Forced light application theme on macOS to circumvent problems using the Dark Mode of macOS Mojave.

GDXDUMP

- New command line option setText to show the list of set text stored in a GDX file.
- Option symbols now lists the cardinality of a symbol in the column `Records`.

GDXXRW

- Fixed exit behavior if there were some duplicate records found when reading a symbol but less than specified by maxDupeErrors: If there are no other errors the exit code is 0 now (it was 16 in the past)

GMSZIP/GMSUNZIP

- Replaced the 32bit binaries `gmszip` and `gmsunzip` by more recent 64bit versions on macOS.

MPS2GMS

- Turn some errors that can be safely ignored when reading an MPS file into warnings.

3.22.1.6 Object Oriented APIs

Python

- Fixed a bug with default variable levels being `1.0` instead of `0.0`.

3.22.1.7 Model Libraries

GAMS Data Library

- **GDX2ACCESSExample1.gms : Dumping the Contents of trnsport.gdx** (125)
- **GDX2ACCESSExample2.gms : Writing Explanatory Text to Database** (126)
- **GDX2ACCESSExample3.gms : Dumping a large Table to Database** (127)
- **GDX2ACCESSExample4.gms : Special Value Mapping** (128)
- **GDX2ACCESSExample5.gms : Renaming Fields** (129)

GAMS Test Library

- **sosmiqcp01.gms : SOS1 and SOS2 behavior - MIQCP** (794)
- **sosmip01.gms : SOS1 and SOS2 behavior - MIP** (795)
- **sosminlp01.gms : SOS1 and SOS2 behavior - MINLP** (796)

3.22.1.8 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	18.5.8	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.10	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.9	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 8.1	7.5	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 11.1	11.0	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓	✓		
LINDOGLOBAL 12.0	✓	✓	✓	✓		
LOCALSOLVER 8.5	✓	✓	✓	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
ODHCPLEX 4		✓	✓			
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 6.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 4.0	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 33.01	✓	✓	✓	✓	32.01	29.01

3.22.2 27.2.0 Minor release (May 23, 2019)

3.22.2.1 Acknowledgments

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

3.22.2.2 GAMS System

GAMS

- Fix problem with missing entries for scalar variables and equations in the index file for the GAMS Output in certain cases.

3.22.2.3 Solvers

CONOPT

- New library 3.17J.
- New library 4.12.

Lindo/LindoGlobal

- New libraries 12.0.157.

LOCALSOLVER

- New libraries 8.5 (20190430).

NLPEC

- Added new option parmFile to allow to point to additional options for the GAMS run of the scalar model produced by NLPEC.

ODHCPLEX

- New libraries 4.26.

OSIXPRESS

- Fixed use of standalone Xpress license.

3.22.2.4 Tools

GAMS Studio

- New version 0.11.2.
  - Stability and performance improvements.
  - GDX Viewer
    - Added Drag and Drop feature in GDX Table View

3.22.2.5 Object Oriented APIs

Python

- Fixed a memory leak in the Python 3 version of the API.
- We plan to drop support for Python 2.6 with the next major release.

3.22.2.6 Expert Level APIs

- Fixed a memory leak in the Python 3 version of the APIs.
- We plan to drop support for Python 2.6 with the next major release.

3.22.3 27.3.0 Minor release (July 04, 2019)

3.22.3.1 Acknowledgments

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

3.22.3.2 GAMS System

- Fixed a bug that resulted in undercounting the number of physical and logical processors on Windows systems with more than 64 logical processors. GAMS/Base and many solvers use this count, e.g. when setting threads to 0 or -1.

GAMS

- Allow domain projection with $load also if the source symbol is a variable or equation.
- Fixed the cause for an unwanted compilation error with `embeddedCode` in certain situations.
- Unified behavior of the profile option: In the past, the `Profile Summary` was always written, when the profiling was activated from the command line, but was sometimes omitted when set as an option. Now, the summary gets always written. It does not matter anymore, where profiling was activated.

Documentation

- Added a Preface that reflects the history of the GAMS documentation. In particular it points out the importance of external contributors, in particular Bruce McCarl.

3.22.3.3 Solvers

BARON

- Fixed load of Xpress library on Linux.

CBC

- New libraries.

CONOPT

- New library 3.17K.
- Bug fix in library 4.12.

GAMSCHK

- Fixed error in postopt report. Previously, the report for the column occurring last in the listing file would be omitted, and all its entries would appear in the report for the preceding column.
- Fixed crash that occurred in some cases.
- The above problems occurred in GAMS 26 and previous. A fix was introduced with GAMS 27, but this fix went too far and introduced new problems. With this version, the fix to 27.1 is reverted and a more moderate fix is used instead.

LocalSolver

- Fixed missing solver log.

Mosek

- New libraries 8.1.0.81.
- Fixed ignoring option solvefinal.

SCIP

- New libraries 6.0.2.

SoPlex

- New libraries 4.0.2.

3.22.3.4 Tools

GAMS Studio

- New version 0.11.5
  - Stability and performance improvements.
  - Added buttons to toolbar to show/hide Project Explorer, Output-, and Help widget.
  - Added shortcut `Ctrl+H` to focus Main widget.
  - Added shortcut `Ctrl+Shift+G` to focus Output widget.
  - Added shortcut `Esc` to close several widgets.
  - GDX Viewer:
    - Columns in the data view are now resized to the minimum width necessary to see all the data automatically.
    - Added shortcut to auto-resize all columns in the data view after the width was changed by hand (`Ctrl+R`).
  - Search:
    - Added shortcut `Ctrl+Enter` for "Search All".
    - Added ability to replace in more than one file at a time.
    - Find Next/Previous now respects all search options and will jump to the next file if there are also matches.

3.22.3.5 Object Oriented APIs

Python

- Fixed a bug in GamsJob.interrupt() that caused an `OverflowError` on 64 bit platforms.

3.23 26 Distribution

3.23.1 26.1.0 Major release (February 02, 2019)

3.23.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Stefano Alva, Adam Christensen, Hanna Donau, Stephen Frank, Anastasis Giannousakis, Jan-Erik Justkowiak, David Laudy, Andreas Lundell, Thomas Maindl, Nils Mattus, Scott McDonald, Noah Rhodes, Tom Rutherford, and Anna Straubinger.

3.23.1.2 Versioning

From this release on, we use a new interpretation on the GAMS version number scheme XX.Y.Z. XX alone now indicates the GAMS distribution number (was XX.Y before). Y is used to distinguish between beta, major, and minor releases of the XX distribution. Z is used to distinguish different maintenance releases. That is, we now use

- XX.0.Z for beta versions,
- XX.1.0 for the major release of distribution XX,
- XX.Y.0, Y $>$ 1, for minor releases,
- XX.Y.Z, Y $\geq$ 1, Z $>$ 0, for maintenance releases.

See Release Types regarding the meaning or what constitutes a major, minor, or maintenance release. As before, we increase the required license date for major releases only.

Further, we changed the default installation directory of the GAMS system from using only the distribution number to also include the minor version number. Thus, as before, GAMS is now installed into directory names which contain the XX.Y part of the GAMS version number.

3.23.1.3 Platforms

- We plan to drop support for Mac OS X 10.11 (El Capitan) with the next major release.
- We plan to drop support for Windows Vista with the next major release.
- Fixed problem using solvers with Fortran dependencies on a Mac OS X 10.11 system.

3.23.1.4 GAMS System

GAMS

- Changed behavior of $compress, $decompress, and $encrypt: `source` and `target` cannot be identical anymore; this could have lead to unreliable behavior in the past.
- Extended the influence of $onMulti: This affects the model statement as well now.
- Added new dollar control option $onMultiR, which is similar to $onMulti but replaces existing data instead of merging into it.
- Added put_utility keyword `solver` to select a solver for a given or all possible model types by name at execution time.
- Extended the syntax to allow Implicit Set Definition (or: Domain Defining Symbol Declarations). Here is a small example showing how this can be used:

```
Set
   i 'canning plants'
   j 'markets';

Table d(i<,j<) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;

Display i,j;
```

And this is the result of the `Display` statement:

```
----     10 SET i  canning plants

seattle  ,    san-diego


----     10 SET j  markets

new-york,    chicago ,    topeka
```

- Updated the base module and components as necessary to allow for spaces in names of any files and directory used by GAMS, including the system directory, the working directory, the scratch directory, the process directory, and remove any checks warning about and preventing this. While the Windows system had already allowed spaces in previous versions, this change updates the other platforms to behave similarly.
- Changed the reading of command-line arguments on non-Windows platforms! This should only affect how arguments with spaces in them are passed on to GAMS. Previously, such arguments needed extra quoting or protection on non-Windows platforms. With this update, the extra protection is no longer needed: the same quoting of space-containing arguments can be used on all platforms. For example, with GAMS 25.1 and previous, two pairs of quotes were required on non-Windows platforms to protect spaces in an argument:

  ```
  gams xx.gms --MY_GDX '"outer space"'
  ```

  while now only one pair of quotes is needed on all platforms:

  ```
  gams xx.gms --MY_GDX "outer space"
  ```

- Increased the number of labels, which can be handled by GAMS from 200 million to ∼2.1 billion and let GAMS terminate with the meaningful error message, if that limit is reached.
- Windows only: Fixed a bug which treated a path (e.g. to an input file or used as inputDir) as relative path, if it starts with `/`. Example: When running `cd C:\tmp && gams.exe /data/myModel.gms`, GAMS tried to load the file `C:\tmp\data\myModel.gms` in the past. Now, it loads `C:\data\myModel.gms`.
- Fixed some problems with option asyncSolLst.
- Fixed a bug which could have caused GAMS to accept duplicate entries in a data statement, when they came in unsorted order.
- Fixed the dollar control option $onUpper (it did not do anything before).

GMSPython

- Updated `pip` to work with recent macOS versions.

3.23.1.5 Solvers

ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Fixed an MKL error when running any of these solvers after CPLEX with solvelink=5 in the same GAMS run on Linux.
- Updated Intel MKL libraries to version 2019.0 on Linux and to version 2018.3 on Windows 64-bit.

BARON

- New libraries 18.11.12 for 64-bit platforms.
  - New cutting planes for convex-transformable functions.
  - New cutting planes for polynomial optimization problems.
  - More robust treatment of cutting planes.
  - Improved optimality-based range reduction mechanisms.
  - CBC can now use multiple threads on Windows, too.
- We plan to drop BARON for Windows 32-bit with some future release.
- The solver time reported back to GAMS (resource usage) is now the wall clock time instead of the CPU time.

Bonmin, CBC, Couenne, Ipopt

- New libraries.

CONOPT 4

- New library 4.09 with many reliability improvements and a fix for wrong non-opt markers.

CPLEX

- Ctrl-C (or Interrupt) was broken for GAMS/CplexD. This has been fixed.
- Improved performance of retrieving the solution for a QCP with many quadratic rows.

Gurobi

- The solver time reported back to GAMS (resource usage) is now the wall clock time instead of the CPU time on Unix.
- Allow quadratic rows with `=E=` and let Gurobi deal with potential errors. Gurobi can sometimes substitue quadratic terms defined by `=E=` rows. The check of quadratic row to be either `=L=` or `=G=` in the GAMS/Gurobi link prevented this.

KESTREL

- Added support for MPSGE models.

Knitro

- New libraries 11.1.1 (11.0.1 for 32-bit Windows) containing several bug fixes and performance improvements, including:
  - improved performance on convex quadratic models,
  - improved generation of knapsack cuts, resulting in faster performance on some MINLP models,
  - improved performance when using the SQP algorithm,
  - new values for option linsolver to choose parallel factorization routines MA97 and MA86.
- Artelys has announced that Knitro 11.0 was the last Knitro release to support 32-bit Windows. We will be dropping GAMS/Knitro for 32-bit Windows in the near future.

Lindo/LindoGlobal

- New libraries 12.0.90.
  - LP Solver Improvements:
    - With new enhancements made to the simplex solvers, the average performance on large instances has increased by 18% for the primal simplex and 15% for the dual simplex compared to the previous version.
    - Improved performance on LP's when using multiple cores with concurrent execution of Primal, Dual, and Barrier.
  - Quadratic and Nonlinear Solver Improvements:
    - Much faster handling of large quadratic matrices, e.g. 1000 x 1000.
  - Global Solver Improvements:
    - Improved handling of discontinuous functions, e.g., MOD(x,k), ROUND(x).

LocalSolver

- New libraries 8.0 (20181106).
  - Performance improvements on routing & scheduling problems, especially Pickup & Delivery problems.
  - Learning algorithms inside LocalSolver, which allow to automatically and dynamically tune up the search process for each solved instance, were improved. This leads to improved performances, especially for short resolution times (minutes).

Mosek

- New libraries 8.1.0.72.

ODHCPLEX

- New libraries 4.09 for Linux with many reliability improvements.
- New libraries 4.13 for Windows 64-bit with many reliability improvements and addressing dynamic loading in concurrent threads and processes.
- Updated options: DynamicSearch, FirstFeas, and Strategy
- New options: DecompDensity, Divisor, ODHFeasOpt, MaxBacktrack, `NewCallback`, Recurse, RecurseDecomp, RecurseIterLim, RecurseLog, RecurseMinIterLim, RecurseSolIterLim, RelaxSOS2, and ThreadLog.

OQNLP

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

PATH

- New libraries 5.0.00.
  - Fixed crash in Lemke's method that could happen for models with extremely poor scaling.
  - New capability to use alternate basis-handling packages via dynamic loading of shared libraries. Two alternate packages are currently supported:
    * BLU-LUSOL: Block LU updating using the LUSOL routines. The shared library required for this is included in the distribution.
    * UMFPACK: Tim Davis' multifrontal LU factorization package. The shared library required for this is **not** included in the GAMS distribution, but can be downloaded from the SuiteSparse site. N.B.: We have verified UMFPACK on Linux, macOS, and Windows using UMFPACK v5.7.7 from SuiteSparse 5.3.0.

SCIP

- New libraries 6.0.1.
  - Primal Heuristics
    * new diving heuristic farkasdiving that dives into the direction of the pseudosolution and tries to construct Farkas-proofs
    * new diving heuristic conflictdiving that considers locks from conflict constraints
    * performance improvements for Adaptive Large Neighborhood Search
      · changed default of parameter heuristics/alns/adjustminimprove from 1 to 0
      · changed default of parameter heuristics/alns/alpha from 0.2 to 0.0016
      · new parameter heuristics/alns/adjusttargetnodes
      · changed default of parameter heuristics/alns/eps from 0.5 to 0.468584
      · changed default of parameter heuristics/alns/gamma from 0.2 to 0.0704146
      · changed default of parameter heuristics/alns/minimprovehigh from 0.1 to 0.01
      · changed default of parameter heuristics/alns/minimprovelow from 0.0001 to 0.01
      · removed parameter heuristics/alns/stallnodefactor
      · changed default of parameter heuristics/alns/startminimprove from 0.05 to 0.01
      · changed default of parameter heuristics/alns/targetnodefactor from 1.5 to 1.05
      · new parameter heuristics/alns/unfixtol
      · changed default of parameter heuristics/alns/crossover/minfixingrate from 0.4 to 0.3
      · changed default of parameter heuristics/alns/dins/maxfixingrate from 0.5 to 0.9
      · changed default of parameter heuristics/alns/dins/minfixingrate from 0.1 to 0.3
      · changed default of parameter heuristics/alns/localbranching/minfixingrate from 0 to 0.3
      · changed default of parameter heuristics/alns/mutation/minfixingrate from 0.4 to 0.3
      · changed default of parameter heuristics/alns/proximity/minfixingrate from 0 to 0.3
      · changed default of parameter heuristics/alns/rens/maxfixingrate from 0.7 to 0.9
      · changed default of parameter heuristics/alns/rins/maxfixingrate from 0.6 to 0.9
      · changed default of parameter heuristics/alns/rins/minfixingrate from 0.2 to 0.3
      · changed default of parameter heuristics/alns/zeroobjective/minfixingrate from 0 to 0.3
  - New branching rule lookahead that evaluates potential child and grandchild nodes to determine a branching decision
  - LP Solver Interface
    * lp/checkstability is now properly implemented for use with SoPlex
    * new parameter lp/alwaysgetduals

    - new parameter lp/checkfarkas
  - Separation (cutting planes)
    - cuts generated from certain quadratic constraints with convex feasible region are now global
    - new parameter `separating/dircutoffdistfac`
    - new parameter `separating/efficacyfac`
    - removed parameter separating/maxincrounds
    - new parameter separating/zerohalf/dircutoffdistweight
    - new parameter separating/zerohalf/efficacyweight
    - changed default of parameter separating/zerohalf/goodscore from 0.9 to 1
    - new parameter separating/zerohalf/goodmaxparall
    - new parameter separating/zerohalf/initseed
    - new parameter separating/zerohalf/maxparall
    - new parameter separating/zerohalf/objparalweight
  - Symmetry Handling
    - restructured timing of symmetry computation to allow to add symmetry handling components within presolving
    - changed default of parameter constraints/symresack/ppsymresack from 0 to 1
    - new parameter `presolving/symbreak/addconsstiming`
    - changed default of parameter `presolving/symbreak/detectorbitopes` from 1 to 0
    - removed parameter "presolving/symmetry/computepresolved"
    - new parameter `propagating/orbitalfixing/enabledafterrestarts`
    - new parameter `propagating/orbitalfixing/performpresolving`
    - changed default of parameter `propagating/orbitalfixing/presolpriority` from 0 to -1000000
    - changed default of parameter `propagating/orbitalfixing/presoltiming` from 28 to 16
    - new parameter `propagating/orbitalfixing/symcomptiming`
  - improved bound tightening for some quadratic equations
  - new parameter display/relevantstats
  - new parameter misc/scaleobj
  - removed `implfree` presolver
  - See also the `full release notes` and the `release paper`.

SELKIE

The `SELKIE` solver - new to GAMS with this release - is an EMP solver that implements decomposition methods for multi-agent equilibrium and related models.

SoPlex

- New libraries 4.0.1.
  - new simplifier step to perform variable aggregation for equations with two variables
  - new parameter bool:ensureray to re-solve the original problem in case of infeasibility/unboundedness to get a valid proof/ray
  - See also the `full release notes` and the `release paper`.

Xpress

- New libraries 33.01.10.
  - local symmetry re-detection in MIP tree search
  - improved branching for highly symmetric MIPs
  - new or improved cuts, degeneracy handling, crossover, preprocessing
  - FICO does not release patches for all platforms with each patch release, so some platforms lag behind the most-used ones.

3.23.1.6 Tools

CSDP

- See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.

CSV2GDX

- Extended recognition of special values:
  - `N/A` is recognized as `NA`,
  - `TRUE` is recognized as `1`,
  - `FALSE`, `NONE` and `NULL` are recognized as `0`.
- Added new option text to specify the column to get explanatory text from when reading a set.
- Changed handling of bad UELs: Reading a bad UEL (e.g. a UEL exceeding the maximum length) causes an error now. The old behavior can be enabled again using the new option acceptBadUels.

GAMS Studio

- New version 0.10.3
  - Stability and performance improvements.
  - Added a new section Comparing GAMS Studio and GAMSIDE to the documentation.
  - Added a Reference File Viewer to navigate the source code of GAMS models via a reference file, especially when multiple files are involved. A reference file contains all symbol references of GAMS models and is created using the reference parameter when running the model.
  - Added a Tab Browser (Deprecated) to list and search through all open files. It is accessible with a button next to the main tab bar or via the shortcut `Ctrl+K`.
  - GDX Viewer :
    - Added a first, basic version of a Table View : The new button "Table View" switches the view to a table based representation. The button is only enabled for multidimensional symbols and puts the last dimension into the header of the table. Drag and drop of items is not yet supported but will be in the future.
    - Added option to hide specific value columns for variables and equations.
    - Added support for the use of different encodings using the "Edit->Encoding" menu. The default encoding for a GDX file generated in Studio by running a GMS file with automatic GDX creation (`F10`) is the one used by the GMS file.
  - Editor

    - Added smart typing: Automatic insertion of closing character for brackets and quotes.
    - Added ability to select text and press either a opening bracket or quote to surround text with these symbols.
    - Improved backspace behavior, if auto indentation is active.
  - Output Widget
    - Added info, warning, and error messages to GAMS Studio system log.
    - GAMS log files are now written to disk (can be disabled in settings).
    - Added "clear log" context menu entry.
  - Project Explorer
    - Studio automatically adds all files specified through GAMS parameters (.gdx, .ref, .lst) to the Project Explorer.
    - Allow to rename Project Explorer groups.
    - Groups are closed if they are empty.
    - If a group is closed the associated GAMS process will be stopped.
    - Added drag and drop of files between groups.
    - Added shortcut to focus Project Explorer (`Ctrl+J`) and allow to navigate using arrow keys to select a file.
    - Added "Select all", "Expand all" and "Collapse all" to the context menu.
    - Added shortcut to select all files (`Ctrl+A`).
  - Various
    - Added context menu for tabs (offering different close actions).
    - Added animation for groups with running GAMS processes.
    - Studio comes to foreground when double clicking a Studio-associated file.
    - The "About GAMS" dialog contains many important information like used GAMS version, system directory, or license information. It features a "Copy System Information" button which copies all of these information to easily attach them to a support request. The dialog has a second tab which gives an overview about solvers, their license status, and what kinds of capabilities they have.
    - "File -> New..." dialog starts with a default for a new file name.
    - "File -> Open..." (or drag'n'drop) of one file now focuses that file if it is already present in the Project Explorer.
    - New entry "File -> Open... in new group" to force creation of new group (even if the is already present in the Project Explorer).
    - "File -> Open..." (or drag'n'drop) of multiple files puts all files opened at once into a single group instead of individuals.
    - Added shortcut to focus command line parameter edit field (`Ctrl+L`).
    - Added `Ctrl+F` as close shortcut for search dialog.
    - Changed shortcut to open Settings dialog to `F7`.

Note

This version of GAMS Studio does not support the Dark Mode of MacOS Mojave. Please disable this mode to use GAMS Studio.

GDXDUMP

- New options for writing out special values: `NAOut`, `UndfOut`, `PInfOut`, `MInfOut`, and `ZeroOut`.
- Fixed broken generated GAMS code, if `UelTable` option was used, but there were no UELs.
- Lower and upper bounds of equations are dumped always now (even if their values are the default). This fixes a potential problem, that the equation type is unknown for the generated GAMS code.
- When writing CSV files the element text of set elements could not be written. Now with enabled option CSVSetText the element text of a set element is written as the last column in the CSV file.

GDXXRW

- Fixed a bug which could have caused a crash when reading merged cells from Excel using the GDXXXRW options `useRC` and `cMErge=1`.

MODEL2TEX

- The new script `model2tex.sh` sets the required dependencies for the model2tex tool on Linux and macOS. Using this script is the recommended way for using model2tex. Calling the Python script directly is not recommended anymore since the used Python interpreter would need to be configured manually. `model2tex.sh` uses the Python interpreter that comes with the GAMS distribution.

3.23.1.7 Object Oriented APIs

Python

- The `setup.py` script now installs the idx expert-level API.

3.23.1.8 Model Libraries

GAMS Data Library

New models added:

- **csv2gdx2.gms : CSV2GDX Example 2 - Reading CSV Files with CSV2GDX** (111)
- **csv2gdx3.gms : CSV2GDX Example 3 - Reading Semicolon separated Data** (112)
- **csv2gdx4.gms : CSV2GDX Example 4 - Dealing with missing Labels and Duplicates** (113)
- **csv2gdx5.gms : CSV2GDX Example 5 - Reading more than one Parameter from a single Input File** (114)
- **csv2gdx6.gms : CSV2GDX Example 6 - Reading economic Data from the World Bank Data Catalog** (115)
- **csv2gdx7.gms : CSV2GDX Example 7 - Reading special Values with CSV2GDX** (116)
- **csv2gdx8.gms : CSV2GDX Example 8 - Reading an compressed encrypted Input File** (117)
- **csv2gdx9.gms : CSV2GDX Example 9 - Reading Options from an external File** (118)
- **GDXXRWExample17.gms : Reading several Scalars from Spreadsheet** (119)
- **GDXXRWExample18.gms : Reading Sets from Spreadsheet** (120)
- **GDXXRWExample19.gms : Writing Parameter to Spreadsheet including Zero Values** (121)
- **GDXXRWExample20.gms : Reading empty Cells with colMerge and reading merged Excel Ranges with cMerge** (122)
- **GDXXRWExample21.gms : Skipping empty Rows or Columns and Ignoring Rows or Columns** (123)
- **GDXDUMPExample1.gms : GDXDUMP - Adding double Quotes to an user defined Header when writing to CSV** (124)

GAMS EMP Library

New model added:

- **emppython1.gms : Three Simple EMP Models with Indexed EMP Syntax and Python Parser** (104)

GAMS Test Library

New models added:

- **csv2gdx7.gms : CSV2GDX - Checking the Error Messages for incorrect Parameter Input** (752)
- **csv2gdx8.gms : CSV2GDX - Testing the valueDim Option** (753)
- **csv2gdx9.gms : CSV2GDX - Testing the Field Separator Tab** (754)
- **csv2gdx10.gms : CSV2GDX - Testing the Field Separator Semicolon** (755)
- **csv2gdx11.gms : CSV2GDX - Testing the Field Separator Comma** (756)
- selkie01.gms : SELKIE test suite (757)
- selkie02.gms : SELKIE test suite (758)
- selkie03.gms : SELKIE test suite (759)
- selkie04.gms : SELKIE test suite (760)
- selkie05.gms : SELKIE test suite (761)
- selkie06.gms : SELKIE test suite (762)
- selkie07.gms : SELKIE test suite (763)
- selkie08.gms : SELKIE test suite (764)
- selkie09.gms : SELKIE test suite (765)
- selkie10.gms : SELKIE test suite (766)
- selkie11.gms : SELKIE test suite (767)
- selkie12.gms : SELKIE test suite: sub-diagonalization (768)
- selkie13.gms : SELKIE test suite: dualvar (769)
- selkie14.gms : SELKIE test suite: isolated implicit variable (770)
- selkie15.gms : SELKIE test suite: proximal perturbation (771)
- selkie16.gms : SELKIE test suite: obj variable setting (772)
- selkie17.gms : SELKIE test suite: obj variable setting (773)
- selkie18.gms : SELKIE test suite: obj variable not free (774)
- selkie19.gms : SELKIE test suite: equation marginal values (775)
- **put12.gms : Testing put_utility solver** (776)
- **gdxxrw8.gms : GDXXRW - Testing the Option intAsText** (777)

- **gdxxrw9.gms : GDXXRW - Testing the Option checkDate** (778)
- **gdxxrw10.gms : GDXXRW - Testing cMerge when reading Sets with the values Option** (779)
- **gdxxrw11.gms : GDXXRW - Reading and writing special Values** (780)
- **gdxxrw12.gms : GDXXRW - Testing the skipEmpty and cMerge Option** (781)
- **gdxxrw13.gms : GDXXRW - Testing the values Option when reading or writing Set Elements** (782)
- **gdxxrw14.gms : GDXXRW - Testing different Excel Range Specifications** (783)
- **gdxdump2.gms : GDXDUMP - Testing the dumping Functionality on several GDX Files** (784)
- **gdxdump3.gms : GDXDUMP - Dumping special Values of a Parameter from GDX** (785)
- **gdxdump4.gms : GDXDUMP - Dumping special Values of Variable-Subfields from GDX** (786)
- **onmulti8.gms : Test for $onMultiR** (787)
- **implset1.gms : Test for Implicit Set Definition** (788)
- **emppy1.gms : Test an equilibrium model using emp python** (789)
- **emppy2.gms : Formulate the simplevi.gms example using emp python** (790)
- **emppy3.gms : Test a combination of optimization and vi agents using emp python** (791)
- **miqcp04.gms : Test behavior for integer infeasible model** (792)
- **duplic01.gms : Detecting duplicate entries in unsorted data** (793)

3.23.1.9 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	18.5.8	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.9	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.8	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 8.1	7.5	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 11.1	11.0	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 12.0	✓	✓	✓	✓		
LINDOGLOBAL 12.0	✓	✓	✓	✓		
LOCALSOLVER 8.0	✓	✓	✓	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
ODHCPLEX 4		✓	✓			
OQNLP	✓	32bit				
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 6.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 4.0	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 33.01	✓	✓	✓	✓	32.01	29.01

3.24 25.1 Distribution

3.24.1 25.1.1 Major release (May 19, 2018)

3.24.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Eden Huang, Katja Buhrkal Jensen, Jan-Erik Justkowiak, Erwin Kalvelagen, Marko Loparic, Bruce McCarl, Scott McDonald, Edmund Moshammer, Andres Ramos, Tom Rutherford, Wilfredo Sifuentes, and Tapio Westerlund.

3.24.1.2 Platforms

- On Windows, some solvers in the GAMS distribution as well as GAMS Studio have dependencies on certain MS Visual C++ runtime libraries. Most of them are included in the GAMS distribution or are present on most Windows systems. However, we are aware that in in rare cases, some libraries are missing. In such situations, we advise to run the appropriate installer for these libraries:
  - On a 32-bit GAMS version, execute `[GAMS system]\apifiles\C++\lib\vs2013\vcredist_x86.exe`, followed by `[GAMS system]\apifiles\C++\lib\vs2015\vcredist_x86.exe`.
  - On a 64-bit GAMS version, execute `[GAMS system]\apifiles\C++\lib\vs2017\vcredist_x64.exe`.

3.24.1.3 GAMS System

GAMS

- New conditional expressions to be used in conditional compilation, namely `gdxDimension`, `gdxEquType`, `gdxParType`, `gdxSetType`, `gdxSymExist`, `gdxVarType`, and `uelExist`. Details about these can be found here.
- New command line option LstTitleLeftAligned: Write title of LST file left aligned.
- New command line option ShowOSMemory: Show the memory usage reported by the operating system (either resident set size or virtual set size) instead of the internal counting.
- GAMS throws now a compilation error if an unexpected suffix at a symbol is encountered in an option statement involving identifiers. In the past such suffixes were ignored, which could have given the impression that they were actually doing something. So, all the `.dim1` suffixes in the following example will create an error now:

```
Set i(*)
    j(*)
    ij(*,*) / i1.j1, i1.j2, i2.j1, i2.j2 /;

Option i <  ij.dim1;
Option i <= ij.dim1;

Set       k     / k1*k6 /;
Parameter ii(*) / i1 4, i2 5, i3 6 /
          iii(k,*);

Option iii > ii.dim1;
Option shuffle = ii.dim1;
```

- The details of the expansion of multi-line macros were changed with this release. In the past, trailing white space in the macro definition was always removed. Now, in multi-line macros, the white space before the continuation character \ is preserved. This example demonstrates the difference:

```
$macro xAndY(x,y) x and \
                  y
scalar x /1/;
if(xAndY(1,x), display 'true');
```

  In the past, this caused a compilation error, since the macro was expanded to `if(1 andy, display 'true');` ("and" and "y" were concatenated to "andy"). Now that the white space after "and" is preserved, this does not cause a problem anymore.

- The statement `Alias (k,k);` for an unknown symbol `k` causes a compilation error now.

- In rare cases, embedded code recognized indented Python code incorrectly. This has been fixed.
- New property gams.debug for controlling debug information of the Object-oriented API in embedded code sections.
- Fixed a problem related to SSL support in `GMSPython` that prevented certain tools (e.g. pip) from working on Linux and macOS.
- Models of type MCP can now be solved with modifiable parameters in the GUSS framework and as OO-API `GAMSModelInstance`.

3.24.1.4 Solvers

BARON

- New libraries 18.5.8.
  - Updated to Ipopt 3.12.8 and CBC 2.9.9.
  - Replaced Ipopt's linear solver MUMPS by MA57 from HSL, a collection of Fortran codes for large-scale scientific computation. Thus, when using IPOPT as NLP solver, BARON may now be more robust and a little faster, especially for problems without integer variables.
  - New bounds reduction strategies based on optimality conditions can reduce the size of the branch-and-bound tree.
  - Rewrote core memory management routines for speed increase and memory reduction.
  - On some problems, BARON currently provides incorrect marginal values. This problem will be fixed in the near future.
- The use of Ipopt in BARON on macOS is no longer disabled by default.
- There will be no future updates of BARON for Windows 32-bit.

Bonmin(H), CBC, Couenne

- Wall-clock time is now always used to apply a timelimit to Branch-and-Bound. The nww option `clocktype` can be used to switch back to CPU time.

CONOPT

- New libraries 4.05 for CONOPT 4.
  - Improvements for multi-threading.
- New libraries for CONOPT 3.
- Fixed problems with redundancy in preprocessor and an issue related to system error 2024.

ConvertD

- Added options PermuteVars and PermuteEqus to permute the declaration of variables and equations, respectively, in scalar GAMS output.
- Tweaked printing of variable bounds and activity levels in scalar GAMS ouput.
- Fixed handling of infinite upper bounds for integer variables in scalar GAMS output: A line that sets GAMS option intvarup to 0 is now added to the output.

Couenne

- New libraries.

DICOPT

- Improvements to Feasibility Pump:
  - Fixed handling of nonlinear objective function when maximizing.
  - Fixed update of cutoff decrement after a solution with objective value very close to zero has been found.
  - Fixed translation of NLP projection problem optimal value to norm of associated projection. Changed default for option fp_projzerotol from 1e-6 to 1e-4.
  - Added creation of cuts derived from solution of NLP projection problem. Use option fp_projcuts to disable.
  - Added option fp_integercuts to disable integer cuts or enable them for mixed-binary problems only. The latter is the new default.
  - Added option fp_mipgap to specify an optimality tolerance (relative gap limit) for the MIP projection problem.
  - The stall limit is now only applied after a first solution has been found. Changed default for option fp_iterlimit to 20.

GUROBI

- New libraries 8.0.
  - As announced for GAMS 24.9, Gurobi 8 is no longer supported for Windows 32-bit. We continue to ship Gurobi 7.5 for this platform.
- New partition heuristic based on user annotation via the new dot option .partition and enabled with new option PartitionPlace.
- Completely new interface to Gurobi's Remote Services (i.e. compute server, distributed algorithm, and instant cloud). For details see Compute Server and the following sections.

JAMS

- Added capability to handle QVI models.
- The options controlling the reformulation strategy for shared variables in multi-agent EMP models have changed: see ImplVarModel for details.

LocalSolver

- New libraries 7.5 (20180405).
- To use LocalSolver 7.5, a machine-specific LocalSolver license is now required on Linux, too. See the solver manual on how to obtain such a license. Alternatively, it is still possible to use LocalSolver 7.0 by choosing LOCALSOLVER70 instead of LOCALSOLVER as solver.

Mosek

- New libraries 8.1.0.53.

ODHCPLEX

- A new solver GAMS/ODHCPLEX from Optimization Direct Inc. has been added to the GAMS solver portfolio.
- The solver implements a set of heuristic methods (named ODHeuristics) for finding feasible solutions to Mixed Integer Programming (MIP and MIQCP) models and uses IBM CPLEX as its underlying solver engine.
- The heuristics decomposition method works in an automatic fashion or can be guided by user specified selections.
- Users will need a GAMS/CPLEX or GAMS/Cplex link license for this solver to work. Commercial users will also need a GAMS/ODHCPLEX license.
- Currently, the solver is available for Windows 64-bit only.

SCIP

- New libraries 5.0 (09c736f).
  - changed default for parameter `constraints/quadratic/gaugecuts` from 1 to 0
  - changed default for parameter heuristics/completesol/freq from 1 to 0
  - changed default for parameter separating/cmir/freq from -1 to 10
  - changed default for parameter separating/cmir/freq from 1 to 0
  - changed default for parameter separating/flowcover/freq from -1 to 10
  - changed default for parameter separating/flowcover/freq from 1 to 0

Xpress

- New libraries 32.01.10 containing several minor bug fixes and performance improvements.

3.24.1.5 Tools

GAMS Studio

- This release contains the first preview version of GAMS Studio - a completely new integrated development environment for GAMS. GAMS Studio is still in a very early stage of development, but we believe it is already mature enough to be a productive tool. GAMS Studio is available for Linux, macOS, and Windows only.
- GAMS Studio combines many (but not all) features of the classic GAMS IDE with some new elements:
  - a code editor to write GAMS models (including basic syntax coloring, block edit mode, etc.),
  - execution of GAMS models,
  - an output panel that presents the progressing GAMS log,

  - a listing viewer (including a tree view to navigate through the file) to inspect the listing file,
  - an interactive option editor to set GAMS parameters,
  - a GDX viewer that shows the table of content and data of GDX files and offers useful details like sorting and filtering of data,
  - a project explorer helps to manage different projects in one session, and
  - an integrated help system to make it easier to find additional information, e.g., by pressing F1 while the cursor is on a GAMS keyword in the code editor.

- We encourage our users to weigh in, help prioritize GAMS Studio's future development directions, or provide any other feedback. If you find problems, miss features, or have comments, please send a note to studio@gams.com.
- It is also possible to contribute directly or build on top of GAMS Studio, since the source code is available on GitHub (https://github.com/GAMS-dev/studio) under the GNU GPL license (see https://github.com/GAMS-dev/studio).
- We plan to provide updates for GAMS Studio frequently in the near future, also separate from new releases of the GAMS distribution. To facilitate the update process, Studio includes an interactive check (requiring an Internet connection) for the availability of newer versions. This "Check for Update" button can be found below the menu "Help".
- A short overview about GAMS Studio can be found in the YouTube channel GAMS Lessons.

Note

-
  - New options to define the association for GAMS files (`.gms`) have been added to the installer for GAMS on Windows. For now, the GAMS IDE is still the default choice, but this may change for future releases.
  - On Linux, Studio is installed in the form of an AppImage as part of the GAMS system directory.
  - On macOS, Studio is part of the DMG installer only. It is installed as an additional application in the `Applications` directory next to the GAMS Terminal application and is available through the Launchpad. GAMS files (`.gms`) are associated with GAMS Studio.

Attention

-
  - In rare cases, GAMS Studio crashes when the `Open` or `Save` dialog gets opened. This is mostly related to third party software, e.g. older versions of "Dell Backup and Recovery". It is recommend to update, switch off or uninstall the software.
  - We are aware of some performance issues at this stage of development. So while things work well, for example, with the models from our model libraries, you might experience some delay when working with huge files, e.g., while searching or when the log is processed for very long output.

XLSDump

- Sometimes, Excel Spreadsheets contain links to pictures, which cannot be found. This situation caused an error in XLSDump in previous versions. Now, such an error is ignored.

3.24.1.6 Object Oriented APIs

- New implementation of `GAMSJob.Interrupt()` on Linux and macOS that removes dependency to the command line tool `pstree`.

C++

- Added support for Microsoft Visual Studio 2015 on Windows 32-bit.

Java

- Changed the naming scheme of a temporary working directory to be created from `yyyyMMdd_HHmmss` to the prefixed `gams_` (defined by `GAMSGlobals.workingDirectoryPrefix`, in case no working directory has been specified.
- Fixed the behavior when a GAMSDatabase is added with a name that already exists. A GAMSException will be raised now (see **GAMSWorkspace.addDatabase(String databaseName)** and **GAMSWorkspace.addDatabaseFromGDX(String gdxFileName, String databaseName)**).
- Calls on `GAMSWorkspace.finalize` and `GAMSSymbolIterator.finalize` are no longer available, because calling a finalizer method can arbitrarily delay the reclamation of object instances and potentially create unpredictable outcome. Whenever the object is no longer needed it is recommened to explicitly dispose the object rather than to rely on the Java garbage collector to do the job. See **GAMSDatabase.dispose**, **GAMSModelInstance.dispose**, and GAMSOptions.dispose.

3.24.1.7 Expert Level APIs

- All Java native interfaces to expert-level APIs are now included in `[Path/To/GAMS]/apifiles/Java/api/GAMSJava/

GMO

- The constant `MAXEVALTHREADS` (i.e. the number of parallel threads supported for function and derivative evaluations in the solver interface library GMO) has been increased from 16 to 64. This effectively increases the same limit in the CONOPT4 solver.

3.24.1.8 Model Libraries

PSOPTLIB - Power System Optimization Modelling in GAMS

- This new library by Alireza Soroudi has been added to the GAMS system. This is a collection of the models based on the book Power System Optimization Modelling in GAMS by Alireza Soroudi. The library contains a selection of 32 models from various areas of power system optimization expressed in GAMS. Book and library describe how the General Algebraic Modeling System (GAMS) can be used to solve various power system operation and planning optimization problems. The book is the first of its kind to provide readers with a comprehensive reference that includes the solution codes for basic/advanced power system optimization problems in GAMS, a computationally efficient tool for analyzing optimization problems in power and energy systems. The book covers theoretical background as well as the application examples and test case studies. It is a suitable reference for dedicated and general audiences including power system professionals as well as researchers and developers from the energy sector and electrical power engineering community and will be helpful to undergraduate and graduate students.
- You can retrieve the individual models through the IDE and Studio model library browser, via the command line utility `psoptlib` or through calls in the Object Oriented APIs.

GAMS EMP Library

- **simpleqvi1.gms : Simple Quasi-Variational Inequality** (101)
- **simpleqvi2.gms : Simple Quasi-Variational Inequality** (102)
- **simpequil3.gms : Simple Generalized Nash Equilibrium Problem** (103)

GAMS Model Library

- **guss2dim.gms : Two dimensional scenario GUSS Example** (423)
- **obstacle.gms : An Obstacle Problem** (424)
- **csched**: Added two more formulations for a related problem. Contributed by Tapio Westerlund.

3.24.1.9 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	✓	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.9	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.8	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		
GUROBI 7.5	✓	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 10.3	✓	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 11.0	✓	✓	✓	✓		
LINDOGLOBAL 11.0	✓	✓	✓	✓		
LOCALSOLVER 7.5	✓	✓	7.0	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
OQNLP	✓	32bit				
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 5.0	✓	✓	✓	✓		

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 3.1	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 32.01	✓	✓	✓	✓	✓	29.01

3.24.2 25.1.2 Minor release (August 01, 2018)

3.24.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Marcel Adenauer, Shaoyan Guo, Eden Huang, David Laudy, Evangelos Panos, Wilfredo Sifuentes and Larissa de Vries.

3.24.2.2 Platforms

- Fixed problem using solvers with Fortran dependencies on a Mac OS X 10.11 system.

3.24.2.3 GAMS System

GAMS

- Fixed a bug which caused an unexpected error when using the `+` operator in a model statement with the following symbol being a model itself, like this:

```
variable x;
equation e1, e2;
e1.. x =L= 5;
e2.. x =G= 3;
model m2 /e2/;
model m /e1+m2/;
solve m min x use lp;
```

3.24.2.4 Solvers

BARON

- Fixed translation of branching priority values when specified in BARON options file and GAMS option PriorOpt not enabled.
- Specifying `maxdouble` as branching priority in a BARON options file now behaves as if specifying `inf` as branching priority in a GAMS model.

BONMIN

- The default for option `number_cpx_threads` is now initialized according to the value of the GAMS option threads.

CONOPT

- New libraries 3.17I for CONOPT 3.
- New libraries 4.06 for CONOPT 4.

GUSS

- Models with variable parameters and model attribute `holdFixed=1` could not be solved with 25.1.1. In this release the holdFixed option is automatically turned off.

Ipopt

- New libraries.
  - Fixed problems in handling of time limit (reslim).

LocalSolver

- New libraries 7.5 (20180601).

ODHCPLEX

- New libraries 3.4.3.
- ODHCPLEX is now available for Linux, too.

SoPlex

- New libraries 3.1 (b0e0048).

SCIP

- New libraries 5.0 (1d9c207).

3.24.2.5 Tools

MODEL2TEX

- Backslashes in the explanatory text of symbols are replaced automatically with `\textbackslash`.

GAMS Studio

- New version 0.9.2
  - Stability improvements
  - Allow only one instance of Studio to run at the same time (when file association for Studio is active, a double click on a gms file will open that file in a running Studio instance and does not open a new Studio)
  - Added startup parameter `--gams-dir` to specify a non-default GAMS system folder to be used
  - Added shortcuts for `Interrupt` and `Stop` (`F12` and `Shift+F12`)
  - Restructured the `File -> Encoding` menu
  - About dialog: added button to copy product information to clipboard
  - GDX Viewer: added facility to search for symbols
  - Option Editor: allow to use F1 to open help
  - Project Explorer:
    - "Add Existing Files" allows to add multiple files at once
    - Added context menu entry to open the log for a group-node

    Note
  - When migrating from an older version of GAMS Studio, one might have an empty project explorer and no recent files listed. This is expected behavior because of a bug fix within the GAMS Studio settings file.

3.24.2.6 Object Oriented APIs

Python

- Fixed a bug that prevented the setup.py file to be called with parameter `-noCheck` for turning off the version check.
- The setup.py script can be used from an arbitrary location. It is not required anymore to switch to the files location before installing.

3.24.3 25.1.3 Minor release (October 30, 2018)

3.24.3.1 Acknowledgments

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

3.24.3.2 GAMS System

GAMS

- Fixed a bug which caused the sysIdent property not being set correctly after a solve statement.
- Fixed a bug which could have caused a problem with `execute_load` being called after `endEmbeddedCode` in a loop.

3.24.3.3 Solvers

ANTIGONE

- If interrupting ANTIGONE (using Ctrl+C, for example), the final trydual NLP solve does no longer stop immediately in the first iteration.

ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2018.4 on macOS.

CONVERTD

- When using options `Jacobian` or `Hessian`, new symbols `iobj`, `objConst`, and `objJacVal` are written to the GDX file. If the model can be reformulated with a true objection function, the symbol `iobj` contains the label of the objective defining equation, while the symbols `objConst` and `objJacVal` constain the objective constant and the coefficient of the objective variable. If the model cannot be reformulated with a true objective, `iobj` will be empty and `objConst` and `objJacVal` will be 0.

GUROBI

- New libraries 8.1.0.
- Option GUROBIpreqlinearize allows the value of 2 (Force Linearization and get compact relaxation).

JAMS

- Added some experimental indexed EMP syntax. See example model **emppython1.gms : Three Simple EMP Models with Indexed EMP Syntax and Python Parser**.

Lindo/LindoGlobal

- New libraries 11.0.338.

MOSEK

- New libraries 8.1.0.64.

3.24.3.4 Tools

CSDP

- See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.

3.24.3.5 Object Oriented APIs

Python

- The `UpdateAction Primal` and `Dual` in a `GamsModifier` were not recognized and triggered an exception. This has been fixed.

3.24.3.6 Model Libraries

GAMS EMP Library

- **emppython1.gms : Three Simple EMP Models with Indexed EMP Syntax and Python Parser** (104)

3.25 25.0 Distribution

3.25.1 25.0.1 Major release (January 17, 2018)

3.25.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Jeff Dischler, Michael Ferris, Dhruv Gupta, Erwin Kalvelagen, Florian Landis, Andre Lemelin, Erfan Mohagheghi, Anthony Paul, and Nathan Sudermann-Merx.

3.25.1.2 Platforms

- Due to the discontinuation of Cplex on Windows 32bit (with Cplex 12.7 in 2016) we are faced with potentially phasing out/dropping solvers depending on Cplex on Windows 32bit with the next major release. The most likely candidate for this is ANTIGONE. These solvers will still be available for 64-bit Windows and other supported platforms. Other solver vendors (e.g. Gurobi and FICO-Xpress) have also announced the discontinuation of their product on the Windows 32bit platform. If you rely heavily on the availability and support of the Windows 32bit platform please contact `support@gams.com` to discuss your options.
- As announced, dropped support for x86-64 Solaris.
- As announced, increased the minimal required MacOS X version to 10.11.
- As announced, increased the minimal required GLIBC version on Linux to 2.12.

3.25.1.3 GAMS System

GAMS

- Extended the `$offEmbeddedCode` facility to allow the use of a "projection operator": `$offEmbeddedCode {symbol[<[=]embSymbol[.dimX]]}`
  More information can be found here.
- Added support for the Object-oriented GAMS Python API in the Embedded Code Facility for Python. The method `gams.set()` accepts instances of type `GamsParameter`, `GamsSet`, etc. as data. Instances of `GamsWorkspace` and `GamsDatabase` can be retrieved using the properties `gams.ws` and `gams.db`. The property `gams.wsWorkingDir` can be used to specify the working directory of the created GamsWorkspace.
- Added `$libinclude pyEmbMI` to conveniently work with Python OO-API `GamsModelInstance` from embedded code. See **embmiex1.gms : Simple Embedded Code ModelInstance example**.
- Fixed a problem in embedded code when reading an empty scalar symbol.
- Fixed a problem when solving a model with solveLink 5, 6, or 7 with communicating scales without scaleOpt active.
- The EPS value in the Embedded Code Facility for Python has been changed to `4.94066E-324`. This is the same value that is used for EPS in the Object-oriented Python API.
- New Put_Utility `save`: Writes a save file of the current state of execution.
- Improvement for the Put_Utility Statement: It is no longer required to define a file and activate it, just to use a `put_utility`.
- New command line option fileStemApFromEnv: Append a string read from an environment variable to the fileStem.
- Change to command line option fileStem: Create an immediate error when the value contains a \ or / character to avoid problems later on.
- Fixed a bug with `execute_unload`, which could have caused a `set` to be exported mistakenly as `alias` to a different `set`, if symbol renaming was used.
- Fixed a bug which prevented to open more than 65,000 put files.
- Solves with scaleOpt=1 using GUSS or instantiations of OO-API GAMSModelInstances resulted in some circumstances in the report of scaled solutions or other erroneous behavior. Hence GAMS will now reset the modelname.scaleOpt attribute to NA before such a solve or GAMSModelInstance generation. As a consequence, models solved via GUSS or as a GAMSModelInstance in the OO-APIs will not be scaled by GAMS.
- Solving MCP models with scaleOpt=1 using the solvers AMPL or PYOMO creates an execution error now. In the past this lead to wrong results potentially.

3.25.1.4 Solvers

ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- Updated Intel MKL libraries to version 2018.0 for Linux and Mac OS X and to version 2018.1 on Windows 64-bit.

CBC

- New libraries.

CONOPT

- New libraries 3.17G (Conopt3).
- New libraries 4.04 (Conopt4).
  - Fixed a system error 65666.

Couenne

- New libraries.
  - Bugfixes and tuning for fixed point bound tightening.
- If Couenne does not find a feasible solution, but the initial point is feasible, then the initial point is now reported back to GAMS.

CONVERT

- If the filename provided for the `Jacobian` or `Hessian` option contains the string `novenames` ConvertD will not export the original equation and variable names as set text to elements of sets i and j. Having the original equation and variable names can make the GDX file significantly larger and slow to write and read.
- Fixed a problem when trying to write scaled MCP models.

CPLEX

- New libraries 12.8.0.
- New parameters
  - SubMIPScale: Parameter to scale the problem matrix when CPLEX solves a subMIP during MIP optimization
  - SubMIPStartAlg: Starting algorithm for subMIP of a MIP
  - SubMIPSubAlg: Algorithm for subproblems of a subMIP of a MIP
  - DynamicRows: Switch for dynamic management of rows
  - Sifting: Switch for sifting from simplex optimization
- In CplexD allow the BendersPartition to be set via the .stage variable suffix together with option BendersPartitionInStage.

Gurobi

- Fixed a problem for model with SOS variables but no constraints.

KESTREL

- New options `neos_username` and `neos_user_password` can be used in the option file in order to submit authenticated jobs using a NEOS user account.

LocalSolver

- New libraries 7.5 (20171117) for Mac OS X and Windows.
  - Preprocessing entirely rewritten: size reduction by a factor up to 10 on some huge instances.
  - Combinatorial models based on booleans and integers: performance improvement and increased ability to prove optimality.
  - Continuous linear and nonlinear models: performance improvement through the integration of state-of-the-art algorithms.
- Note, that to use LocalSolver 7.5, a machine-specific LocalSolver license is now required. See the solver manual on how to obtain such a license. Alternatively, it is still possible to use LocalSolver 7.0 by choosing LOCALSOLVER70 instead of LOCALSOLVER as solver.

Mosek

- New libraries 8.1.0.34.

SBB

- Fixed a bug introduced in 24.8 that prevents the infeasSeq option to work.

SCIP

- New libraries 5.0 (2b35b18).
  - SCIP can now automatically detect and exploit symmetries in MIPs (Linux and Mac OS X only)
    - added parameter misc/usesymmetry to determine which symmetry handling should be used
    - added parameter groups `presolving/symmetry`, `presolving/symbreak`, constraints/symresack, constraints/orbisack, constraints/orbitope
  - Presolving
    - new presolver sparsify that tries to cancel nonzero coefficients in linear constraints by adding multiples of linear equalities
    - disabled reformulation of products of a binary variable with a linear term that does not solely involve binary variables
      - added parameter `constraints/quadratic/binreformbinaryonly` to disable reformulation of products of binary and non-binary variables
    - revised disaggregation of quadratic constraints: the number of created constraints can now be controlled and the disaggregated constraints are scaled in order to increase numerical accuracy
      - replaced `constraints/quadratic/disaggregate` by `constraints/quadratic/maxdisaggrsize` to bound the total number of created constraints when disaggregating a quadratic constraint
      - added parameter `constraints/quadratic/disaggrmergemethod` to change the strategy of how to merge independent blocks of quadratic constraints
  - Primal Heuristics
    - new primal heuristic ALNS that orchestrates eight different LNS heuristics adaptively using algorithms for the multi-armed bandit problem

  * new primal heuristic MPEC that solves a MPEC reformulation of a mixed-binary nonlinear problem by regularized NLP reformulations
  * improved the clique and variable bound pre-root heuristics, which are now often able to fix many more variables
    - removed parameters heuristics/clique/{multiplier,initseed}
    - replaced parameter heuristics/{clique,vbounds}/minfixingrate by heuristics/clique/minintfixingrate, heuristics/vbounds/minintfixingrate, heuristics/clique/minmipfixingrate, and heuristics/vbounds/minmipfixingrate, which check the fixing rate before LP solving and after sub-MIP presolve
    - added parameters heuristics/clique/maxbacktracks and heuristics/vbounds/maxbacktracks to limit the number of backtracks in the fix-and-propagate phase
    - added parameters heuristics/clique/uselockfixings and heuristics/vbounds/uselockfixings to enable fixing of additional variables based on variable locks
    - added parameters heuristics/vbounds/feasvariant and heuristics/vbounds/tightenvariant to specify the fixing variants used by the vbounds heuristic
    - changed default for parameters heuristics/clique/freq and heuristics/vbounds/freq from -1 to 0
    - changed default for parameter heuristics/clique/priority from -1000500 to 5000
    - changed default for parameter heuristics/vbounds/priority from -1100600 to 2501
  * added parameters heuristics/completesol/beforepresol, heuristics/completesol/maxlpiter, and heuristics/completesol/maxcontvars
  * changed default for parameter heuristics/indicator/oneopt from 1 to 0
  * changed default for parameter heuristics/locks/minfixingrate from 0.25 to 0.65
  * changed default for parameter heuristics/locks/priority from 2000 to 3000

- Separation (cutting planes)
  * utilizing linear inequalities (computed in the OBBT propagator) to compute stronger linearizations for bilinear terms
    - added parameters `constraints/quadratic/usebilinineqbranch`, `constraints/quadratic/minscor` and `constraints/quadratic/bilinineqmaxseparounds`
    - added parameter `constraints/quadratic/mincurvcollectbilinterms` to change the minimal curvature of constraints to be considered when returning bilinear terms to other plugins
    - added parameters propagating/obbt/itlimitfactorbilin, propagating/obbt/minnonconvexity, and propagating/obbt/createbilinineqs
  * improved cut post-processing (apply coefficient tightening, enforce maximal dynamism), selection, and management
    - added parameters separating/maxlocalbounddist, separating/maxcoefratio, and `separating/intsupportfac`
    - removed parameter separating/orthofac
    - changed default for parameter separating/cutagelimit from 100 to 80
    - changed default for parameter separating/minefficacy from 0.05 to 0.0001
    - changed default for parameter separating/minefficacyroot from 0.001 to 0.0001
    - changed defaults for parameters `separating/minortho` and `separating/minorthoroot` from 0.5 to 0.9
    - changed default for parameter `separating/objparalfac` from 0.0001 to 0.1
    - changed default for parameter separating/poolfreq from 0 to 10
  * MIP cutting planes are now separated within the tree search, too
    - parameter separating/maxstallrounds only applies to nodes in the tree (not the root node, anymore); use the new parameter separating/maxstallroundsroot for the root node
    - added parameters separating/*/expbackoff to all separators, which increases the frequency exponentially over the depth in the tree
    - added parameter separating/maxincrounds

- changed default for parameter separating/maxstallrounds from 5 to 1
- changed default for parameter separating/maxrounds from 5 to -1
- changed default for parameter separating/maxroundsrootsubrun from 1 to -1
- changed default for parameter separating/gomory/delayedcuts from 1 to 0
- changed default for parameter separating/gomory/freq from 0 to 10
- changed default for parameter separating/gomory/maxbounddist from 0 to 1
- changed default for parameter separating/impliedbounds/freq from 0 to 10
- changed default for parameter separating/impliedbounds/maxbounddist from 0 to 1
- changed default for parameter separating/strongcg/freq from 0 to 10
- changed default for parameter separating/strongcg/maxbounddist from 0 to 1
- changed default for parameter `separating/strongcg/maxsepacuts` from 50 to 20
- changed default for parameter separating/zerohalf/freq from -1 to 4
- changed default for parameter separating/zerohalf/maxbounddist from 0 to 1

* new implementation of zerohalf separator
  - added parameters separating/zerohalf/badscore, separating/zerohalf/densityoffset, separating/zerohalf/goodscore, separating/zerohalf/maxcutcands, separating/zerohalf/maxrowdensity separating/zerohalf/maxslack, separating/zerohalf/maxslackroot, and separating/zerohalf/minviol
  - removed parameters separating/zerohalf/delayedcuts, separating/zerohalf/ignoreprevzhcuts, separating/zerohalf/maxcutsfound, separating/zerohalf/maxcutsfoundroot separating/zerohalf/maxdepth, separating/zerohalf/maxncalls, separating/zerohalf/maxtestdelta, separating/zerohalf/onlyorigrows, separating/zerohalf/relaxcontvars, separating/zerohalf/scalefraccoeffs, separating/zerohalf/trynegscaling, separating/zerohalf/usezhcutpool, separating/zerohalf/preprocessing/∗, and separating/zerohalf/separating/∗
  - changed default for parameter separating/zerohalf/maxroundsroot from 10 to 20
  - changed default for parameter separating/zerohalf/maxsepacuts from 50 to 20
  - changed default for parameter separating/zerohalf/maxsepacutsroot from 500 to 100
* faster implementation of CMIR cut generation heuristic
  - moved many parameters for flowcover and cmir separators to new parameter group separating/aggregation
  - changed default for parameters separating/cmir/freq and separating/flowcover/freq from 0 to -1
  - changed default for parameters separating/cmir/maxbounddist and separating/flowcover/maxbounddist from 0 to 1
  - changed default for parameter separating/cmir/priority from -3000 to -100000
  - changed default for parameter separating/flowcover/priority from -4000 to -100000
* additional parameter changes
  - removed parameter separating/feastolfac
  - removed parameter separating/cgmip/allowlocal
  - removed parameters separating/{gomory,strongcg}/maxweightrange
  - changed default for parameter separating/gomory/makeintegral from 1 to 0
  - changed default for parameter separating/gomory/maxrank from 3 to -1
  - changed default for parameter separating/gomory/sidetypebasis from 0 to 1
  - added parameter constraints/indicator/maxsepanonviolated to stop separation after separation of non violated cuts
  - removed parameters constraints/{abspower,bivariate,quadratic,nonlinear}/mincutefficacysepa, constraints/{abspower,bivariate,quadratic,nonlinear}/mincutefficacyenfofac, and constraints/soc/minefficacy

– Propagation

* use disjoint set to reduce peak memory usage and time to compute clique table connectedness information
* added analysis of the clique table which identifies possible aggregations via the search for strongly connected components and may detect infeasible assignments on the way

            - added parameter propagating/vbounds/minnewcliques to specify the minimum number of new cliques to trigger another clique table analysis
            - added parameters propagating/vbounds/maxcliquesmedium and propagating/vbounds/maxcliquesexha to limit the number of cliques relative to the number of binary variable for performing clique table analysis
            - changed default for parameter propagating/vbounds/presolpriority from 0 to -90000
            - changed default for parameter propagating/vbounds/presoltiming from 28 to 24
        * extended conflict analysis by analyzing dual solutions of boundexceeding LPs and improved dual ray analysis
            - removed parameters conflict/usemir and conflict/prefermir
            - added parameter conflict/sepaaltproofs
            - added parameter conflict/prefinfproof to determine whether to prefer infeasibility proof to boundexceeding proof
            - changed default for parameter conflict/useboundlp to 'b'
            - changed default for parameter conflict/maxvarsfac from 0.1 to 0.15
            - changed default for parameter conflict/maxvarsfac from 30 to 0
    - LP Relaxation
        * use LP solution polishing during probing and diving mode to activate it during many primal heuristics; remains disabled during strong branching and OBBT
            - only effective if using SoPlex as LP solver
            - added value 3 for parameter lp/solutionpolishing to enable LP polishing only during probing and diving mode
        * added parameter lp/refactorinterval to change the refactorization interval of the LP solver
    - removed parameters constraints/{abspower,bivariate,nonlinear,quadratic,soc}/scaling
    - added parameter groups table/* to control statistic output
    - See also the `full release notes`, the `changelog`, and the `release paper`.

- Fixed a problem when solving a model with solveLink 6 or 7 with scaleOpt active.

SolveEngine

- The GAMS time limit (reslim) is now passed to SolveEngine. Added solver option hardtimelimit to specify a time limit that is enforced on the GAMS side.

SoPlex

- New libraries 3.1.0 (876e6e8).
    - New scaling method that combines geometric and equilibrium scaling. Use new value 6 for option int:scaler to activate this.
    - See also the `release paper`.

Xpress

- New libraries 32.01.05.

3.25.1.5 Tools

CDSP

- See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.

GDXXRW

- Sometimes, Excel is not ready (e.g. because some data needs to be refreshed, when a worksheet is opened), when a read request is sent by GDXXRW. This could cause an exception. With this release we changed the behavior of GDXXRW to wait a second and resend the request in this case. This basically mimics the behavior of setting RWait to 1000 in case of the mentioned exception.

3.25.1.6 Object Oriented APIs

C++

- Changed the compiler from GCC to Clang on Mac OS X.
- Added support for Microsoft Visual Studio 2015 and Microsoft Visual Studio 2017 on Windows 64-bit.
- API binaries have been moved to `apifiles/C++/lib`. On Windows, there are further subdirectories for different compiler versions.
- Update of Visual Studio solutions for the examples on Windows. Three different solutions reflecting the supported versions of Microsoft Visual Studio are available (e.g. examples-vs2013.sln, examples-vs2015.sln, examples-vs2017.sln).
- The GAMS C++ API tutorial has been reworked. Information about building the C++ API examples via cmake, qmake, and Microsoft Visual Studio has been added.
- New example `TransportGDX` that shows how to import and export GDX files. A description has been added to the tutorial.

.NET

- Changes for `GAMSSymbol` and `GAMSSymbolRecord`: Both classes got the `IEquatable<T>` Interface. As a result the behavior of the `Equals` function as well as the `==` and `!=` operators were modified. For both classes `Equals` and `==` returns now true, if the internal data reference is the same. Here is an example:

```
GAMSVariable x1 = db.GetVariable("x");
GAMSVariable x2 = db.GetVariable("x");
if(x1 == x2)
    Console.WriteLine("x1 == x2");
else
    Console.WriteLine("x1 != x2");
if (x1.Equals(x2))
    Console.WriteLine("x1 equals x2");
else
    Console.WriteLine("x1 does not equal x2");
```

  In previous versions we got this output:

```
x1 != x2
x1 does not equal x2
```

  Now we get:

```
x1 == x2
x1 equals x2
```

- New example `TransportGDX` that shows how to import and export GDX files.

Java

- The minimum version requirement of the Java Runtime Environment for using the GAMS Java API is now Java SE 7.
- New `TransportGDX` example to demonstrate how to import and export GDX files.
- Removed method `GAMSSymbol.compact`, deprecated since 24.8.1 (December 2016).
- Changed equivalence behavior of `GAMSSymbol` and `GAMSSymbolRecord` objects. As a result, two symbol objects with the same internal reference are now equivalent, similar to symbol record objects:
  - Two symbols are equivalent if and only if they have the same internal reference.
  - Two symbol records are equivalent if and only if they have the same internal reference.

  The behavior of operator `==` remains unchanged. The following exmaple illustrates the new equivalence behavior:

```
GAMSVariable x1 = db.getVariable("x");
GAMSVariable x2 = db.getVariable("x");
GAMSVariable x3 = x1;
assertTrue(x1.equals(x2));   // true,  previously false
assertFalse(x1 == x2);       // false, previously false
assertTrue(x1.equals(x3));   // true,  previously true
assertTrue(x1 == x3);        // true,  previously true
```

Python

- Added implementation of `__eq__()` and `__ne__()` to classes `_GamsSymbol` and `_GamsSymbolRecord` and its derived classes. As a result the behavior of the operators `==` and `!=` has changed. `==` now returns `True`, if the internal data reference is the same. The behavior of `is` remains unchanged. The following example illustrates the change:

```
x1 = db["x"]
x2 = db["x"]
print(x1 == x2) # now: True,  before: False
print(x1 is x2) # now: False, before: False
```

- New example `transport_gdx.py` that shows how to import and export GDX files. A description has been added to the tutorial.

3.25.1.7 Expert Level APIs

- As announced, the expert-level C++ API files were removed from the distribution. Users should switch to the expert-level C API files. The object-oriented C++ API introduced in the last major release could also be a good alternative for replacement if the user C++ code exchanges data and runs a GAMS model.
- The expert-level Java API files now ensure the load of jni libraries in the similar order as those employed by the Object Oriented Java API: first load from `java.library.path`, if neither specified nor found then from the directory where the API classes are located.

GDX

- Do not allow empty filename as argument for `gdxOpenAppend`, `gdxOpenRead`, `gdxOpenWrite`, and `gdxOpenWriteEx`. This will create an error right away now.

3.25.1.8 Model Libraries

GAMS Model Library

- **embmiex1.gms : Simple Embedded Code ModelInstance example** (417)
- **spbenders1.gms : Stochastic Benders - Sequential GAMS Loop** (418)
- **spbenders2.gms : Stochastic Benders - Async Subsolve GAMS Loop** (419)
- **spbenders3.gms : Stochastic Benders - Sequential GamsModelInstance** (420)
- **spbenders4.gms : Stochastic Benders - Parallel MPI** (421)
- **spbenders5.gms : Stochastic Benders - Parallel MPI with GAMSModelInstance** (422)
- Most models in the GAMS Model library have been refurbished to reflect a common syntax style and features introduced since the model was added.
- **prime**: Make use of the `break` statement for a little nicer and faster formulation.

GAMS Test Library

- **scensol7.gms : Test GUSS Option ReportLastScen** (737)
- **emp27.gms : Test LOGMIP/EMP on x.fx=0 handling** (738)
- **embpy03.gms : Test projection operator when loading data from embedded code** (739)
- **qcp11.gms : Test dual solution of SOCP** (740)
- **put11.gms : Put_Utility without file handle** (741)

3.25.1.9 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓		
BARON	✓	✓	✓	✓		
BDMLP	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓		
CBC 2.9	✓	✓	✓	✓		
CONOPT 3	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓		
CPLEX 12.8	12.6	✓	✓	✓	12.6	✓
DECIS	✓	✓	✓	✓	✓	
DICOPT	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓		

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	Sparc 64bit SOLARIS	IBM Power 64bit AIX
GUROBI 7.5	✓	✓	✓	✓		✓
GUSS	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓
KNITRO 10.3	✓	✓	✓	✓		
LGO	✓	✓	✓	✓	✓	
LINDO 11.0	✓	✓	✓	✓		
LINDOGLOBAL 11.0	✓	✓	✓	✓		
LOCALSOLVER 7.5	✓	✓	7.0	✓		
MILES	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓		
MSNLP	✓	✓	✓	✓	✓	
NLPEC	✓	✓	✓	✓	✓	✓
OQNLP	✓	32bit				
PATH	✓	✓	✓	✓	✓	✓
SBB	✓	✓	✓	✓	✓	✓
SCIP 5.0	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓		
SOPLEX 3.1	✓	✓	✓	✓		
XA	✓	✓	✓			
XPRESS 32.01	✓	✓	✓	✓	✓	29.01

3.25.2 25.0.2 Maintenance release (January 31, 2018)

3.25.2.1 Acknowledgments

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

3.25.2.2 GAMS System

GAMS

- Bugfix: $offEmbeddedCode does not ignore $onUNDF anymore.
- Bugfix: Special values (NA, EPS, INF, ...) are correctly communicated from embedded code back to GAMS.
- Bugfix: Took care about a problem, which caused an unexpected execution error for certain uses of Put_Utility.
- Fixed a bug in GMSPython on Mac OS X which prevented the Python interpreter from working. This problem occurred with the DMG installer only.

3.25.2.3 Solvers

ANTIGONE, Bonmin, Couenne, Ipopt, SCIP

- On Linux, removed the MKL libraries that were optimized to certain x86 instruction sets (AVX, etc.) as this resulted in errors when running ANTIGONE or SCIP, probably due to library conflicts.

CPLEX

- Option Tuning can be repeated in a GAMS/Cplex option file to provide a larger number of model instances for tuning. Before this change the number was restricted by the maxium line length of an option line (256 characters).

LocalSolver

- New libraries 7.5 (20180119) for Mac OS X and Windows.

3.25.2.4 Tools

CSDP

- See ANTIGONE, Bonmin, Couenne, Ipopt, SCIP.

CSV2GDX

- Bugfix: Reading sets (no Value or Values option) with the option AutoRow works again.

3.25.3 25.0.3 Minor release (March 21, 2018)

3.25.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Dominik Bongartz, Wolfgang Britz, Erwin Kalvelagen, Maria Kannavou, Hans Mittelmann, Christoph Pahmeyer, Gilles Scouvart, and Hermann Westerholt.

3.25.3.2 GAMS System

GAMS

- Bugfix for Put_Utility: If a `put_utility` statement was used as the first statement to mark a file as active (so no statement like `put fx;` was used before) and following `put` or `put_utility` statements did not mention this `file` explicitly, then the `file` association got lost.
- Bugfix: Avoid a crash when one of the following attributes was used on an empty Singleton Set: `.len`, `.uel`, `.val` (an execution error is generated instead now)
- Bugfix for command line option fileStemApFromEnv: In the past, if the input file was defined including a file extension, the value of this option was extended to the extension and not to the file stem as intended. This is fixed now.
- Bugfix for $if exist File on Windows: In the past, if a file was specified as <`Drive`>`:/`<`File`>, e.g., `C:/t.txt` (note the forward slash '/' after the colon ':'), this always returned false, even if the file exists. That has been fixed now.

3.25.3.3 Solvers

BARON

- Fixed use of .EquClass option.

BONMIN(H), CBC, Couenne

- Branch-and-bound now checks wallclock-time, if running with multiple threads.

CONOPT4

- New libraries 4.05.
  - Improved the selection of post-triangular variables when there are multiple candidates.

CPLEX

- Bugfix: If the Cplex conflict refiner (triggered by option iis) could not identify a conflict the behavior was erratic. This has been fixed.
- Bugfix: Value 6 for option MIPStart was documented but got rejected. This has been fixed.

DICOPT

- Fixed setup of NLP projection problem in feasibility pump.
- Fixed stopping criterion when using stop on crossover: DICOPT was stopping as soon as the relative gap between the objective value of the best known solution and the bound provided by the MIP relaxation was below 0.001, thus might have declared suboptimal solutions as optimal. With this release, DICOPT will use the value of GAMS option optcr as optimality tolerance. Note, that the default for this option is 0.1 (!).

Ipopt(H)

- New libraries.

MOSEK

- No update, but be aware that `Mosek announced` to drop the convex nonlinear optimizer in their next major release (Mosek 9). Thus, in a future GAMS release, GAMS/Mosek will not accept models with model types NLP or DNLP anymore. Note, that linear or quadratic problems (LP, MIP, (MI)QCP) are not affected.

SCIP

- New libraries 5.0.1 (227c4c7).
  - New option `presolving/symmetry/displaynorbitvars`.

SoPlex

- New libraries 3.1.1 (ab921a5).

3.25.3.4 Tools

MODEL2TEX

- Added support for singleton sets.
- Fixed a dependency bug on Mac OS X which prevented model2tex from working.
- Fixed a bug regarding SOS variables.
- Fixed a bug regarding domain jumps.
- Added support for Python 3 on Linux and Mac OS X.

3.26 24.9 Distribution

3.26.1 24.9.1 Major release (August 30, 2017)

3.26.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Jan Abrell, Etienne Ayotte-Sauvé, Wolfgang Britz, Florian Habermacher, Florian Häberlein, Maximilian Held, Ignacio Herrero, Hanspeter Höschle, Erwin Kalvelagen, Toni Lastusilta, John Ross, Tom Rutherford, and Linus Schrage.

3.26.1.2 Platforms

- The set of supported platforms has not changed, but we've divided it into the *core* platforms (Windows 32-bit, Windows 64-bit, Linux, and Mac OS X) and the *peripheral* platforms (AIX, x86-64 Solaris, and Sparc64 Solaris). See Supported Platforms for more details. Please note, however, that we are *not* changing the content or behavior of GAMS based on this division: GAMS models will continue to work in the same cross-platform way for both core and peripheral platforms.
- We will drop support for x86-64 Solaris with GAMS 25.0.
- We may increase the minimal required GLIBC version on Linux to 2.12 with GAMS 25.0.
- We may increase the minimal required MacOS X version to 10.11 with GAMS 25.0.

3.26.1.3 GAMS System

GAMS

- New feature, the Embedded Code Facility: This extends the connectivity of GAMS to other programming languages. It allows the use of Python code during compile and execution time. GAMS symbols are shared with the external code, so no communication via disk is necessary.
  The embedded code feature is available on Linux, MacOS X, and Windows. For these platforms, a Python 3.6 installation is included with the GAMS distribution. If the user wants to work with a different Python 3.6, installed separately, for models with embedded code the new command line option `pySetup` needs to be set to 0.

  Note

  This feature is currently in beta status. Any feedback to `support@gams.com` is appreciated.
- New command line option procDirPath: Specifies the directory where the process directory should be created.
- New compile time constants to set the model attribute `solPrint`:
  - 0 = %`solPrint.off%`
  - 1 = %`solPrint.on%`
  - 2 = %`solPrint.silent%`
- New compile time constants to set the model attribute `solveOpt`:
  - 0 = %`solveOpt.Replace%`
  - 1 = %`solveOpt.Merge%`
  - 2 = %`solveOpt.Clear%`
- Allow new string synonyms to set the integer values of the following command line options:
  - solPrint: `Off` (=0); `On` (=1); `Silent` (=2)
  - sysOut: `Off` (=0); `On` (=1);
- Fixed a bug where the following dollar control options did not get written to the dump file when the command line parameter dumpOpt was used. Sometimes this caused an error in the generated file.
  - $[on|off]DotScale
  - $[on|off]Embedded
  - $[on|off]Expand
  - $[on|off]Local
  - $[on|off]Macro
  - $[on|off]Margin
  - $[on|off]Order
  - $[on|off]StrictSingleton
  - $[on|off]UNDF
- Fixed an error which caused the loss of equation information when using the Grid facility in certain cases. This could have caused problems when a hot start should be performed.
- A parameter error is now created if the key of a 'double dash' GAMS parameter exceeds 63 characters (in the past, the key was truncated silently in this case).
- Fixed a crash which happened if the `.tl` suffix was used on an empty Singleton Set in a put statement.
- Fixed a bug which could have caused a crash if nested loops where used in a particular combination.
- Fixed a bug with `$onPut ... $offPut` in a `for` and `repeat` structure: When `$onPut` was the first statement inside one of these programming flow control features, it was executed just once and not repeatedly.

Special Functions

- Several of the more exotic GAMS intrinsic functions (aka "special functions") were re-implemented to address some bug reports and to pass a more rigorous set of tests. The updated functions include the loggamma, gamma, logbeta, beta, and binomial functions. In general, the updated functions offer improved precision and a more consistent behavior in exceptional cases (e.g. overflows, singularities, and domain violations) compared to the previous versions. In addition, the gamma and beta functions are now classified as smooth (NLP) instead of nonsmooth (DNLP) and the domains of the beta and binomial functions have been changed.

Documentation

- The contents of GAMS User's Guide and the McCarl (Expanded) User's Guide have been merged, revised, and reorganized as User's Guide as well as A GAMS Tutorial by Richard E. Rosenthal. Also other parts of the documentation has been reorganized and are now more closely integrated.
- The McCarl GAMS User Guide (CHM and PDF) can now be found in the `mccarl/` subdirectory in the distribution.
- The PDFs containing the solver manuals and the GAMS User's Guide have been replaced by a single PDF containing large parts of the current documentation. However, the main format for the documentation is HTML.

3.26.1.4 Solvers

BARON

- New libraries 17.8.7.
  - New range reduction techniques and relaxations for quadratic constraints.
  - New heuristics for finding feasible solutions of integer programs.
  - Improvements in interfaces to local NLP solvers, including the use of second-order derivatives.
  - FilterSQP added to the list of local solvers. New option AllowFilterSQP and added new possible value 14 for option NLPSol.
  - Bugfixes.

CBC

- New libraries.
  - Fixed some problems with the handling of SOS type 2 in presolve.
- Added option dumpsolutionsmerged to write all found alternate solutions into a single GDX file.

Conopt

- New libraries 3.17E and 4.03.
  - Fixed a serious error in Conopt4 for exactly threads=8. Moreover, improved multi-threading performance.
  - Major revisions to Conopt4 for reliability and performance.
- The option Rtzern is now user settable for Conopt3.

CONVERT

- The GAMS equation and variable scale values (suffix `.scale`) will be communicated to CONVERT independent of the ScaleOpt model attribute.

CPLEX

- Added option workerAlgorithm to select the method for optimizing Benders subproblems.
- Added option writeAnnotation to create Cplex annotation file.
- The header of MIP/solve trace files contains now the option number and the name of the model.

DECIS

- DecisC, DecisM, and the EMP-SP solver Decis are now available for MacOS X.

GAMSCHK

- The procedures that generate output in the listing file are summarized at the end of the GAMSCHK execution. If you use the IDE, these summary lines are clickable and locate the cursor to the corresponding subsection in the IDE. Moreover, the procedures are also entered in the listing file index after the Solution Report.
- The lower bound for options LevelFilt and MargFilt has been reset from 1 to -5.

GUROBI

- New libraries 7.5.1.
  - Fewer constraint violations in MIQP solutions: Gurobi has tightened the internal tolerances for MIQP models to reduce the number of cases where the solution exhibits small constraint violations.
- New parameter startNodeLimit provides additional control over how much is performed to complete a partial MIP start.
- Gurobi 7.5 will be the last Gurobi release that supports 32-bit Windows. You should plan to migrate your applications to 64-bit Windows in the future.
- The header of MIP/solve trace files contain now the active option number and the name of the model.

GUSS/Scenario Solver

- Fixed a problem when initializing variables bounds to 0 (`updateType=0`) that have scenario update parameters for lower and upper bounds.

Lindo/LindoGlobal

- New libraries 11.0
  - LP Solver Improvements:
    - With new enhancements made to the simplex solvers, the average performance on large instances has increased by 20% for the primal simplex and 15% for the dual simplex compared to the previous version.
  - MIP Solver Improvements:
    - New symmetry detection capabilities to reduce overall branch-bound effort. This may dramatically reduce the time needed to prove optimality on some models with integer variables.
    - Perspective and soft-clique cuts effective on difficult MIQP and models with assignment constraints.
  - Global Solver Improvements:
    - Improved stability and robustness through several enhancements to quadratic recognition and range reduction.
  - Nonlinear Solver Improvements:
    - New major release of the nonlinear solver.
    - Improved preprocessor.
    - Use of interval function and derivative computations.
    - Advanced scaling leading to improved solution quality.

LocalSolver

- New libraries 7.0 (20170728).
  - Reinforcement of mixed-integer linear programming (LP & MIP) techniques for combinatorial optimization.
  - Reinforcement of nonlinear programming techniques (NLP) for numerical optimization.

MINOS/QUADMINOS

- New libraries 5.6 (dated July 2016).
  - Improved handling of singularites and empty cols in pivoting code.
  - Further bug fixes and improvements.
- Fix improper handling of scaled CNS models.
- Fix handling of logging frequency.
- QUADMINOS, the quadruple-precision version of MINOS, has been available in previous releases (since 24.4). This release includes the library model **[DQQ]**, an example of how to use MINOS and QUADMINOS together to compute greatly improved solutions at moderate cost.

MOSEK

- New libraries 8.1.0.23.
  - Performance of the presolve has been improved slightly.
  - Multi-thread performance of the conic optimizer has been improved for certain large models.
  - Changed scaling for quadratic and quadratically constraint optimization problems.
  - Bugfixes.

MPSGE

- The MPSGE $sysInclude `mpsgeset` allows now for an optional argument `-mt=0` or `1` after the model name. The default value for the argument `mt` can be controlled via the double dash option `--MPSGEMT=0` or `1`. If the `mt` option is set to 1 the `MODEL.GEN` file is created in the GAMS scratch directory. Hence the `$include` before the solve needs to read `$include "%gams.scrdir%MODEL.GEN"`. This allows to run multiple MPSGE jobs with the same model in the same working directory. The default of this option is 0. The model **hansmge** demonstrates the use.

SBB

- The header of MIP/solve trace files contains now the option number and the name of the model.

SCIP

- New libraries 4.0 (9d3c1b1).
  - Improved conflict analysis through central conflict pool and dual ray analysis for primal infeasible LPs.
  - New solution polishing to improve integrality of LP solutions when using SoPlex as LP solver.
  - Added adaptive solving behavior of SCIP based on solving phases and heuristic transitions.
  - Revised pseudo random number generation and introduced central random seed for all plugins.
  - Randomized tie-breaking in different parts of the code to reduce performance variability.
  - New primal heuristics GINS, LP face, Complete Sol, Locks, Repair, and Multistart.
  - The 1-opt heuristic is now iterated as long as new incumbents are found.
  - Improved tuning of heuristic timings.
  - Reduced memory usage of primal heuristics that use problem copies.
  - New presolving steps that disaggregate SOC constraints, reformulate QP's by adding KKT conditions, and treat variables appearing only in a single quadratic constraint with proper square coefficients.
  - New separators for gauge cuts, convex projection cuts, and perspective cuts for indicator constraints.
  - Improved knapsack approximation algorithms, greedy knapsack solution for the flow cover separation, clique partitioning, and clique separation.
  - New propagator for OBBT on convex NLP relaxation.
  - Tuned propagation methods of several constraint handlers and propagation timings.
  - Improved and extended stuffing for linear constraints.
  - Changed handling of coupling constraints for indicator constraints.
  - See also the `full release notes`, the `changelog`, and the `release paper`.
  - Changed parameters:
    - `presolving/components/*` moved to `constraints/components/*`
    - `conflict/depthscorefac` renamed to `conflict/graph/depthscorefac`
    - `misc/permutationseed` renamed to `randomization/permutationseed`
    - `misc/permuteconss` renamed to `randomization/permuteconss`
    - `misc/permutevars` renamed to `randomization/permutevars`
    - `branching/random/seed`: default changed from 0 to 41
    - `constraints/indicator/sepacouplingcuts`: default changed from 0 to 1
    - `constraints/SOS1/perfimplanalysis`: default changed from 1 to 0
    - `heuristics/ofins/freq`: default changed from -1 to 0

  - `heuristics/reoptsols/freq`: default changed from -1 to 0
  - `heuristics/trivialnegation/freq`: default changed from -1 to 0
  - `heuristics/clique/initseed`: default changed from 0 to 61
  - `lp/solver`: default changed from soplex to soplex2, if CPLEX is not licensed
  - `presolving/abortfac`: default changed from 0.001 to 0.0008
  - `separating/clique/cliquedensity`: default changed from 0.05 to 0
  - `conflict/usesb`: default changed from 0 to 1

- Added option gams/dumpsolutionsmerged to write all found alternate solutions into a single GDX file.

- Changed default for timing/clocktype to wallclock time.

- Initial variable levels can now be passed as partial solution to SCIP. To control the various possibilities, the type of option gams/mipstart has changed from bool to integer. See also subsection Starting point in the GAMS/SCIP solver manual.

SolveEngine

- New solver SolveEngine to solve LP and MIP problems remotely via the Satalia SolveEngine. The SolveEngine aggregates different solution algorithms for optimization problems and automatically selects an algorithm that seems to suite best for a given model instance.

- GAMS/SolveEngine comes free of charge with any licensed GAMS system. Users must have an API key for the Satalia SolveEngine to submit jobs.

SoPlex

- New libraries 3.0 (c32c55a).
  - Added a new scaling implementation Least squares (Curtis-Reid scaling).
  - Added persistent scaling to keep scaled LP for multiple reoptimizations.
  - Added an experimental version of a decomposition based approach to avoid degeneracy in the dual simplex method. This feature is activated by setting the parameter bool:decompositiondualsimplex to true, which sets the basis representation to 'row' and the algorithm to 'dual'.
  - New parameter bool:computedegen to enable computation of the degeneracy of the basis in each iteration.
  - New parameter `int:printcondition` to enable printing the condition number of the basis during solve.
  - Automatically use the row representation for problems with more than 20% more constraints than variables.
  - Changed default for parameter int:factor_update_max from 200 to new value 0.
  - See also the `full release notes` and the `release paper`.

- Changed default type of timer to wallclock time.

XPRESS

- New libraries: Optimizer 31.01.09 (aka XPRESS 8.3). There are many improvements and additions:
  - The parallel MIP code has been completely rewritten to improve performance and scalability.
    - Reduced overhead for small, easy MIPs.
    - Reduced the memory usage for very large MIPs, especially those that are significantly reduced during the initial preprocessing.
    - Heuristics can now be run in parallel with cutting in deterministic mode. Previously, heuristics would only be run in parallel after cutting.
    - Improved implementation of zero-half cutting.
    - Aggregated Mixed Integer Rounding cuts have been improved for network-type problems.
  - New presolve reductions, strengthenings and reformulations for convex quadratic problems.
  - Improved performance and numerical stability of crossover.
  - Code support for AVX2 in the barrier solver: use option cpuPlatform to select the target instruction set.
  - Crossover after a barrier solve is now multi-threaded: see option crossoverThreads for details.
- The header of MIP/solve trace files contains now the option number and the name of the model.

3.26.1.5 Tools

CSV2GDX

- New option `ValueDim`: Indicate if an extra dimension for values is added even if there is just one value column. This is ignored, if there is no value column.

GDXDUMP

- Fixed a problem with writing a scalar variable or equation in CSV format with all fields.

GDXMERGE

- Improved feedback about problems when processing input files.

3.26.1.6 Object Oriented APIs

- Fixed a bug regarding `SymbolUpdateType.Zero` that prevented records from being updated in `GAMSModelInstance.Solve()`.

C++

- This release contains a beta version of the object-oriented C++ API that can be used to control GAMS from within C++11 and later. It allows the seamless integration of GAMS into C++ applications by providing appropriate classes for the interaction with GAMS. The GAMSDatabase class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the GAMSJob class and by using the GAMSModelInstance class a sequence of closely related model instances can be solved in an efficient way.
- The API is available in the distributions for Linux, MacOS X, and Windows.
- Futhermore, the C++ API is published under MIT license and is hosted at the GAMS GitHub organization.
- To use this API please check the GAMS API documentation.

.NET

- New functions `GAMSWorkspace.AddJobFromApiLib`, `GAMSWorkspace.AddJobFromNoaLib` to create GAMSJob from models from the GAMS API Library and the Nonlinear Optimization Applications Library.

Python

- New functions `GamsWorkspace.add_job_from_apilib`, `GamsWorkspace.add_job_from_noalib` to create GamsJob from models from the GAMS API Library and the Nonlinear Optimization Applications Library.
- Added a version check for the setup.py scripts to avoid unintentional installation of wrong versions.

3.26.1.7 Expert Level APIs

- With GAMS 25.0, the expert-level C++ API files will be removed from the distribution. Users should switch to the expert-level C API files. The object-oriented C++ API introduced in this release could also be a good alternative for replacement if the user C++ code exchanges data and runs a GAMS model.

GAMS Modeling Object

- Changed the function `gmoGetModelTypeTxt`: Added argument for model type number instead of using the model type of the stored model.

3.26.1.8 Model Libraries

GAMS Data Library

- **embeddedSort.gms : Sorting of numerical data using the embedded code facility** (108)
- **embeddedSplit.gms : Splitting of labels using the embedded code facility** (109)
- embeddedMultiInstance.gms : Handling multiple instances of the embedded code facility at once (110)

GAMS Model Library

- **dqq.gms : Warm-starting quad-precision MINOS** (414)
- **cbenders.gms : Cplex Benders for a Simple Facility Location Problem** (415)
- **robustlp.gms : Robust linear programming as an SOCP** (416)

GAMS Test Library

- **mpsge14.gms : MPSGE sync test: .GEN/integer1/nsolves** (713)
- **fnsincosintrv.gms : Test sin/cos func/grad interval evals** (714)
- **procdir1.gms : Test correct behavior of procDir, scrDir and procDirPath** (715)
- **empbp06.gms : Bilevel model with phantom vars owned by leader** (716)
- **fnslexp.gms : Test correctness of slexp intrinsic** (717)
- **fnslexp2.gms : Test correctness of slexp intrinsic** (718)
- **fnsqexp.gms : Test correctness of sqexp intrinsic** (719)
- **fnsqexp2.gms : Test correctness of sqexp intrinsic** (720)
- **fnsllog10.gms : Test correctness of sllog10 intrinsic** (721)
- **fnsllog102.gms : Test correctness of sllog10 intrinsic** (722)
- **fnsqlog10.gms : Test correctness of sqlog10 intrinsic** (723)
- **fnsqlog102.gms : Test correctness of sqlog10 intrinsic** (724)
- **fnslrec.gms : Test correctness of slrec intrinsic** (725)
- **fnslrec2.gms : Test correctness of slrec intrinsic** (726)
- **fnsqrec.gms : Test correctness of sqrec intrinsic** (727)
- **fnsqrec2.gms : Test correctness of sqrec intrinsic** (728)
- **fnloggamma.gms : Test correctness of loggamma intrinsic** (729)
- **fnbinomial.gms : Test correctness of binomial intrinsic** (730)
- **breakcont2.gms : Advanced test for break and continue statements** (731)
- **fnlogbeta.gms : Test correctness of logbeta intrinsic** (732)
- **fnbeta.gms : Test correctness of beta intrinsic** (733)
- **ssuffix.gms : List of all System Suffixes** (734)
- **embpy01.gms : Test for embedded code facility** (735)
- **embpy02.gms : Test for embedded code facility** (736)

3.26.1.9 Solver/Platform availability

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	x86 64bit SOLARIS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
ALPHAECP	✓	✓	✓	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓			
BARON	✓	✓	✓	✓			
BDMLP	✓	✓	✓	✓	✓	✓	✓
BONMIN 1.8	✓	✓	✓	✓	✓		
CBC 2.9	✓	✓	✓	✓	✓		

	x86 32bit MS Windows	x86 64bit MS Windows	x86 64bit Linux	x86 64bit Mac OS X	x86 64bit SOLARIS	Sparc 64bit SOLARIS	IBM Power 64bit AIX
CONOPT 3	✓	✓	✓	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓	✓	✓	✓
COUENNE 0.5	✓	✓	✓	✓	✓		
CPLEX 12.7	12.6	✓	✓	✓	12.6	12.6	✓
LINDO 11.0	✓	✓	✓	✓			
DECIS	✓	✓	✓	✓		✓	
DICOPT	✓	✓	✓	✓	✓	✓	✓
GLOMIQO 2.3	✓	✓	✓	✓			
GUROBI 7.5	✓	✓	✓	✓			✓
GUSS	✓	✓	✓	✓	✓	✓	✓
IPOPT 3.12	✓	✓	✓	✓	✓		
KESTREL	✓	✓	✓	✓	✓	✓	✓
KNITRO 10.2	✓	✓	✓	✓			
LGO	✓	✓	✓	✓	✓	✓	
SBB	✓	✓	✓	✓	✓	✓	✓
LINDOGLOBAL 11.0	✓	✓	✓	✓			
LOCALSOLVER 7.0	✓	✓	✓	✓			
MILES	✓	✓	✓	✓	✓	✓	✓
MINOS	✓	✓	✓	✓	✓	✓	✓
MOSEK 8	✓	✓	✓	✓			
MSNLP	✓	✓	✓	✓		✓	
NLPEC	✓	✓	✓	✓	✓	✓	✓
OQNLP	✓	32bit					
PATH	✓	✓	✓	✓	✓	✓	✓
SCIP 4.0	✓	✓	✓	✓	✓		
SNOPT	✓	✓	✓	✓	✓	✓	✓
SOLVEENGINE	✓	✓	✓	✓			
SOPLEX 3.0	✓	✓	✓	✓	✓		
XA	✓	✓	✓				
XPRESS 31.01	✓	✓	✓	✓	✓	✓	29.01

3.26.2 24.9.2 Minor release (November 14, 2017)

3.26.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Paul Buckland, William N. Caballero, Xin Fang, Christophe Gouel, Fabricio Porras-Ortiz, Shigeru Tsubakitani, and Haoshui Yu.

3.26.2.2 GAMS System

GAMS

- Added prefixing of `PYTHONPATH` with the site-packages directory of the GAMS Python 3.6 installation in `GMSPython`.
- Added support for Python sets in the Embedded Python Code method `gams.set()`.
- Speed up detection/extraction of quadratic terms for QCP models.
- Do not allow the use of Embedded Code if execMode is set to 2 or higher.

Installer

- Removed the execution of a Python script during the Windows installation process that patches the path of certain Python programs in `GMSPython\Scripts`.

3.26.2.3 Solvers

BARON

- New libraries 17.10.16.
- Due to a problem with the Ipopt interface in BARON on Mac OS X, the use of Ipopt in BARON is currently disabled on Mac OS X.
- GAMS can now pass a term like `(negativeConstant)**negativeIntegerVariable` to BARON.

  Note

  By default, GAMS correctly rejects such a formulation, though. To work around this, `MaxExecError` needs to be set to a positive number and the option sys12 needs to be set to 1. Here is an example, how this could be used:

```
scalar a /-2/;
integer variable x;
x.lo = -3;
x.up = 3;

variable z;
equation e;

e.. z =e=a**x;

model m /e/;
x.l=1;
MaxExecError = 10;
option sys12 = 1;
solve m min z use minlp;
```

CPLEX

- Removed any limits on the number of Threads in a GAMS/CPLEX option file. Previously, the maximum for option Threads was 128.

CONOPT

- New libraries 3.17F.
  - Fixed an infinite loop when using option LMMXSF.

CONVERT

- Fix for variables with negative lower and infinite upper bound in LINGO output. The LINGO output has been changed to always use the `@Bnd` keyword to write variable bounds.

GUROBI

- New libraries 7.5.2.

GUSS/Scenario Solver

- Added the GUSS option `ReportLastScen`. If this is set to 1 the solution of the last scenario will be reported back to GAMS rather than solution of the base case. This is particularly useful when GUSS is used to implement a homotopy approach where the shock to a parameter is sliced in several small shocks and hence the last scenario represents the solution to the shocked system.

Lindo/LindoGlobal

- New libraries 11.0.300.

LogMIP

- Fixed improper handling of variables fixed at zero that occurred if they were used in nonlinear equations in a disjunction reformulated via the convex hull.

Mosek

- New libraries 8.1.0.31.
- Fixed dual solution for conic quadratic problems.

SCIP

- New libraries 4.0 (#22b4564).

SoPlex

- New libraries 3.0 (#3bfa247).

3.26.2.4 Object Oriented APIs

C++

- Added a try/catch block around the code that uses the classes from the GAMS C++ OO-API in all examples. Without this, the exceptions thrown by the methods of the GAMS classes will not be reported and *harmless* exceptions, like a compilation or execution error in the GAMSJob.run method result in application crashes.

Python

- Fixed a bug in the setup script that prevented files from being installed in the correct location in certain cases.

3.27 24.8 Distribution

3.27.1 24.8.1 Major release (December 21, 2016)

3.27.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Etienne Ayotte-Sauvé, Wolfgang Britz, Göran Bylund, Wietse Dol, Gregory Dourbois, Katja Jensen, Josef Kallrath, Toni Lastusilta, Renger van Nieuwkoop, and Tom Rutherford.

3.27.1.2 Platforms

- On Linux, the minimal required GLIBC version is now 2.7.
- On MacOS X, for some solvers (ANTIGONE, BARON, CPLEX, COIN-OR, SOPLEX, SCIP) the minimal required Mac OS X version is now 10.10. We will drop support for older Mac OS X versions for the complete GAMS system with the next major release.
- A generic license (with platform code GEN) is now limited to the platforms Windows, Linux, and Mac OS X. If you have a generic license and need a license for one of the other platforms (e.g. AIX), please contact sales or support to receive an additional license file for that platform free of charge.

3.27.1.3 GAMS System

GAMS

- The statements `Break` and `Continue` were added to allow more control over the execution of control structures (loop/while/repeat/for):
  - `Break [n];`: Terminate the n inner most control structures (n is optional, if it is omitted, it is set to 1).
  - `Continue;`: Jump to the end of the inner most control structure.

- Extend the report summary of the solution report to print the number of variables and equations where the level was projected to one of the bounds (compare model attribute tolProj), if that number is greater than 0.
- The model attribute tolProj to project levels to bounds is no longer ignored when the GAMS Grid Facility (solveLink = 3 or 4) is used.
- New command line option `fileStem`: Sets the file stem for output files which use the input file name as stem by default, see fileStem for more details.
- New option `MCPRHoldfx` (can be set using the option statement, a command line parameter, or model attribute):
  - Prints list of rows that are perpendicular to variables removed due to the holdfixed setting if set to 1.
- New option `AsyncSolLst` (can be set using the option statement, or a command line parameter):
  - If set to 1, GAMS prints a solution listing to the lst file also when an asynchronous solve (see The Grid and Multi-Threading Solve Facility) is used.
  - The default value is 0.
- New variant for the solveLink option and model attribute: 7 (compile time constant %`solveLink.threadsSimulate%`) - The problem is passed to the solver in core without use of temporary files, GAMS waits for the solver to come back but uses same submission process as 6 (see Multi-threading Submission Testing)
- New function numCores (available at both compile and execution time): Returns the number of logical cores in the system.
- Allow macro expansion in the domain list of a symbol declaration.
- New dollar control splitOption that splits a string representing a option/value pair into option name and option value.
- Fixed a problem with the optional `maxWait` parameter of the function readyCollect.
- Fixed potential error in file created by dumpOpt, if there were symbols loaded from GDX.
- Fixed an error which could lead to wrong results when assigning to a symbol while the symbol's alias was used on the right hand side of the assignment.
- Fixed an error which could lead to a crash if a phantom set element was used to control an assignment inside a loop.
- Fixed a problem with unscaling solutions for MCP models that resulted in incorrect dual values for scaled MCPs. For an example, compare the new test library model **mcp11** when run with 24.8 and with something previous.

Documentation

- The offline documentation now provides `search` and `keyword` indexing functionalities in addition to navigating the documentation.
- The Microsoft Compiled HTML help file `gams.chm` is no longer available.
- The table of contents for browsing the GAMS model libraries is now available offline in HTML format.

3.27.1.4 Solvers

BARON

- New libraries 16.12.7.

CPLEX

- New libraries 12.7.0.0.
- Support for the platfrom MS Windows 32 bit, Solaris i86pc, and Solaris SPARC 64bit has been dropped by IBM for Cplex 12.7. The GAMS system for these platforms contains Cplex 12.6.3.
- Cplex 12.7 implements Benders Algorithm. This is available in CplexD only.
- The IIS option triggers now the conflict refiner. The IIS function in Cplex 12.7 has been replaced by the conflict refiner. The IIS option now also works on infeasible models with model status `Infeasible No Solution` as well as on problems with discrete variables.
- New parameters
  - BendersFeasCutTol: Tolerance for whether a feasibility cut has been violated in Benders decomposition
  - BendersOptCutTol: Tolerance for optimality cuts in Benders decomposition
  - BendersStrategy: Benders decomposition algorithm as a strategy
  - DataCheck: Data consistency checking and modeling assistance
  - RLTCuts: Reformulation Linearization Technique (RLT) cuts
- Modified parameters:
  - MipStart: A value of 6 accepts the (partical) MipStart without any checks

CONOPT

- This distribution includes the first official release of the CONOPT4 solver. The chapter in the solver manual has an interesting subsection on when you should use CONOPT4 for all existing CONOPT users.
- The current solver alias `CONOPT` still points to the CONOPT3 solver. This might change in the next, or next but one major release. We invite you to share your experience with this new version of CONOPT via support (`support@gams.com`) with us.

ConvertD

- Write proper scale and prior information for equations and variables with option `Jacobian`.

DICOPT

- An implementation of a Feasibility Pump primal heuristic (for convex MINLP) has been added to DICOPT. This heuristic can be run before the actual DICOPT algorithm. Outer approximation cuts from the MIP subproblem of the Feasibility Pump are transferred to initialize the MIP outer approximation of the main DICOPT algorithm. Option feaspump can be used to enable the Feasibility Pump and various other options are available to tune the feasibility pump.
- Option convex has been added to indicate the presence of convex MINLP. If this option is set, the defaults for various other options are modified.

GUROBI

- New libraries 7.0.1.
- Explore alternative solutions via the Solution Pool.
- New parameters
  - BestObjStop: Objective value to stop optimization
  - BestBdStop: Objective bound to stop optimization
  - InfProofCuts: Infeasibility proof cut generation
  - StrongCGCuts: Strong-CG cut generation
  - DegenMoves: Degenerate simplex moves
  - TuneCriterion: Tuning criterion
  - SolNPool: Activate export of alternative solution
  - PoolSolutions, PoolSearchMode, and PoolGap to control generation of alternative solutions
- Gurobi 7 supports general constraints. With the help of the dot-option `.GenConstrType` the user can classify a constraint to be of type `Max`, `Min`, `Abs`, `And`, and `Or`.
- Gurobi 7 supports indicator constraints.
- Gurobi 7 supports multi objective hierarchical optimization. Details can be found in the GAMS/Gurobi manual, in subsection Multiple Objectives.

JAMS

- New option ZipDebug=xxx.zip to specify that, in the event of abnormal termination or behavior, a zip file of debugging info be created.
- New reformulation options to handle shared or duplicated equations and variables in equilibrium models.

Knitro

- New libraries 10.2.0.
  - Significant speed and robustness improvements when using BFGS (hessopt=2) or L-BFGS (hessopt=6) Hessian approximations with the default Knitro interior-point method.
  - General performance improvement on mixed-integer models.
  - Minor bug fixes.
  - New mixed-integer SQP (MISQP) algorithm for nonlinear mixed-integer models (mip_method=3). This new algorithm is intended for small, potentially non-convex models with possibly expensive function evaluations. It can be used even when integer variables are not relaxable (i.e. functions can only be evaluated with integer variables at integer points) by setting mip_relaxable=0, and can be used with parallel multistart.
  - New option mip_nodealg to control algorithm used at B&B nodes: overrides the generic algorithm option.
- The Knitro Tuner is now available from the GAMS/Knitro link.

Lindo/LindoGlobal

- New libraries 10.0.131.
  - LP Solver Improvements:
    - With new enhancements made to the simplex solvers, the average performance on large instances has increased by 35% for the primal simplex and by 20% for the dual simplex compared to the previous version.
  - MIP Solver Improvements:
    - New heuristic algorithms help to find significantly better solutions for many models with knapsack constraints and block structures.
    - New MIP preprocessing level devoted to tightening variable bounds for some nonlinear models.
  - Stochastic Solver Improvements:
    - Improved cut management for Nested Benders Decomposition Method leading to speed improvements over 60% for large linear multistage SP instances.
    - Better handling of multistage SP models which do not have full-recourse.
  - Global Solver Improvements:
    - Incorporates bound tightening process to the linearization procedure and improve solvability of linearized model.

LocalSolver

- New libraries 6.5 (20160729).
  - For near-linear (discrete or continuous) problems, new moves have been introduced based on linear programming and mixed-integer linear programming techniques. These moves allow to intensify the search on near-linear models by exploring optimally larger neighborhoods.
- Added possibility for "Hierarchical Optimization of Multiple Objective Functions".
- Removed option `origlog`. GAMS/LocalSolver will now always print the original LocalSolver log.
- Fixed level values of aggregated variables in solution.

Mosek

- New libraries 8.0.0.48 (`Mosek release notes`)
  - Presolve performance has been improved.
  - The eliminator in the presolve has been reimplemented, is usually faster, and requires much less memory.
  - Presolve has been improved on conic quadratic problems.
  - The numerical stability of conic optimizer has been improved significantly, particularly for semidefinite optimization problems.
  - The scaling routine for the conic optimizer is more aggressive.
  - Quadratic and quadratically constrained problems are now internally converted to conic form and are solved using the conic optimizer. Nevertheless full primal and dual information to the original problems is available.
  - A dualizer for conic quadratic problems is now available. By default it dualize the problems before optimizing if deemed worthwhile. The dualization is transparent to the user and can be turned off.

- The conic optimizer linear algebra is now parallelized using Cilk Plus and scales better when the number of threads is increased for large problems. Moreover, for smallish problems using too many threads does not hurt performance.
- The computational efficiency graph partitioning based ordering method in the interior-point optimizer has been improved.
- It is now possible to force the interior-point optimizer to run in the calling thread.
- Only one mixed integer optimizer is available now, which corresponds to the mixed integer conic optimizer that was introduced with version 7.
- The primal network simplex optimizer has been removed. It is suggested to use the dual simplex optimizer instead.
- The primal-dual simplex optimizer has been removed. It is suggested to use the dual simplex optimizer instead.
- The concurrent optimizer has been removed. It is suggested to use the interior-point optimizer instead.
- The following GAMS/Mosek options have been removed:
  * MSK_IPAR_ANA_SOL_BASIS
  * MSK_IPAR_ANA_SOL_PRINT_VIOLATED
  * MSK_IPAR_CONCURRENT_NUM_OPTIMIZERS
  * MSK_IPAR_CONCURRENT_PRIORITY_DUAL_SIMPLEX
  * MSK_IPAR_CONCURRENT_PRIORITY_FREE_SIMPLEX
  * MSK_IPAR_CONCURRENT_PRIORITY_INTPNT
  * MSK_IPAR_CONCURRENT_PRIORITY_PRIMAL_SIMPLEX
  * MSK_DPAR_FEASREPAIR_TOL
  * MSK_IPAR_FEASREPAIR_OPTIMIZE
  * MSK_IPAR_INTPNT_FACTOR_DEBUG_LVL
  * MSK_IPAR_INTPNT_FACTOR_METHOD
  * MSK_IPAR_LOG_CONCURRENT
  * MSK_IPAR_LOG_NONCONVEX
  * MSK_IPAR_LOG_PARAM
  * MSK_IPAR_LOG_SENSITIVITY
  * MSK_IPAR_LOG_SENSITIVITY_OPT
  * MSK_IPAR_MAX_NUM_WARNINGS
  * MSK_IPAR_MIO_CONT_SOL
  * MSK_IPAR_MIO_CUT_CG
  * MSK_IPAR_MIO_CUT_LEVEL_ROOT
  * MSK_IPAR_MIO_CUT_LEVEL_TREE
  * MSK_IPAR_MIO_FEASPUMP_LEVEL
  * MSK_DPAR_MIO_HEURISTIC_TIME
  * MSK_IPAR_MIO_HOTSTART
  * MSK_IPAR_MIO_KEEP_BASIS
  * MSK_IPAR_MIO_LOCAL_BRANCH_NUMBER
  * MSK_DPAR_MIO_MAX_TIME_APRX_OPT
  * MSK_IPAR_MIO_PRESOLVE_AGGREGATE
  * MSK_IPAR_MIO_PRESOLVE_PROBING
  * MSK_IPAR_MIO_PRESOLVE_USE
  * MSK_DPAR_MIO_REL_ADD_CUT_LIMITED
  * MSK_IPAR_MIO_STRONG_BRANCH
  * MSK_DPAR_MIO_TOL_MAX_CUT_FRAC_RHS
  * MSK_DPAR_MIO_TOL_MIN_CUT_FRAC_RHS
  * MSK_DPAR_MIO_TOL_REL_RELAX_INT

  - MSK_DPAR_MIO_TOL_X
  - MSK_IPAR_MIO_USE_MULTITHREADED_OPTIMIZER
  - MSK_IPAR_NONCONVEX_MAX_ITERATIONS
  - MSK_DPAR_NONCONVEX_TOL_FEAS
  - MSK_DPAR_NONCONVEX_TOL_OPT
  - MSK_IPAR_PRESOLVE_ELIM_FILL (use MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL instead)
  - MSK_IPAR_PRESOLVE_ELIMINATOR_USE
  - MSK_IPAR_PRIMAL_REPAIR_OPTIMIZER
  - MSK_IPAR_QO_SEPARABLE_REFORMULATION
  - MSK_IPAR_WARNING_LEVEL
  - MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_CONIC_ITEMS (use MSK_IPAR_WRITE_IGNORE_INC instead)
  - MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_NL_ITEMS (use MSK_IPAR_WRITE_IGNORE_INCOM instead)
  - MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_PSD_ITEMS (use MSK_IPAR_WRITE_IGNORE_INCOM instead)

- The following GAMS/Mosek options have been added:
  - MSK_DPAR_DATA_SYM_MAT_TOL: Absolute zero tolerance for elements in in symmetric matrices.
  - MSK_DPAR_DATA_SYM_MAT_TOL_HUGE: An element in a symmetric matrix which is larger than this value in absolute size causes an error.
  - MSK_DPAR_DATA_SYM_MAT_TOL_LARGE: An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.
  - MSK_DPAR_INTPNT_QO_TOL_DFEAS: Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
  - MSK_DPAR_INTPNT_QO_TOL_INFEAS: Controls when the conic interior-point optimizer declares the model primal or dual infeasible.
  - MSK_DPAR_INTPNT_QO_TOL_MU_RED: Relative complementarity gap feasibility tolerance used when interior-point optimizer is applied to a quadratic optimization problem.
  - MSK_DPAR_INTPNT_QO_TOL_NEAR_REL: Termination tolerance multiplier that is used if no accurate solution can be found.
  - MSK_DPAR_INTPNT_QO_TOL_PFEAS: Primal feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
  - MSK_DPAR_INTPNT_QO_TOL_REL_GAP: Relative gap termination tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.
  - `MSK_IPAR_INTPNT_MULTI_THREAD`: Controls whether the interior-point optimizers can employ multiple threads if available.
  - MSK_IPAR_MIO_CUT_CLIQUE: Controls whether clique cuts should be generated.
  - MSK_IPAR_MIO_CUT_GMI: Controls whether GMI cuts should be generated.
  - MSK_IPAR_MIO_CUT_IMPLIED_BOUND: Controls whether implied bound cuts should be generated.
  - MSK_IPAR_MIO_CUT_KNAPSACK_COVER: Controls whether knapsack cover cuts should be generated.
  - MSK_IPAR_MIO_CUT_SELECTION_LEVEL: Controls how aggressively generated cuts are selected to be included in the relaxation.
  - MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE: Enables or disables perspective reformulation in presolve.
  - MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL: Controls whether presolve can be repeated at root node.
  - MSK_IPAR_MIO_VB_DETECTION_LEVEL: Controls how much effort is put into detecting variable bounds.
  - MSK_IPAR_OPF_WRITE_HEADER: Write a text header with date and MOSEK version in an OPF file.

    - MSK_IPAR_OPF_WRITE_HINTS: Write a hint subsection with problem dimensions in the beginning of an OPF file.
    - MSK_IPAR_OPF_WRITE_PROBLEM: Write objective, constraints, bounds etc.
    - MSK_IPAR_OPF_WRITE_SOL_BAS: Whether to include basic solution in OPF files.
    - MSK_IPAR_OPF_WRITE_SOL_ITG: Whether to include integer solution in OPF files.
    - MSK_IPAR_OPF_WRITE_SOL_ITR: Whether to include interior solution in OPF files.
    - MSK_IPAR_OPF_WRITE_SOLUTIONS: Enable inclusion of solutions in the OPF files.
    - MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL: Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of presolve.
    - MSK_DPAR_SEMIDEFINITE_TOL_APPROX: Tolerance to define a matrix to be positive semidefinite.
    - MSK_IPAR_WRITE_DATA_PARAM: If this option is turned on the parameter settings are written to the data file as parameters.
    - MSK_IPAR_WRITE_LP_FULL_OBJ: Write all variables, including the ones with 0-coefficients, in the objective.
    - MSK_IPAR_WRITE_MPS_FORMAT: Controls in which format the MPS is written.
    - MSK_IPAR_WRITE_TASK_INC_SOL: Controls whether the solutions are stored in the task file too.
    - MSK_SPAR_WRITE_LP_GEN_VAR_NAME: Sometimes when an LP file is written additional variables must be inserted.
  - For the following options, the default value has changed:
    - MSK_DPAR_MIO_TOL_FEAS from 1e-7 to 1e-6
    - MSK_IPAR_LOG_MIO_FREQ from 1000 to 10
    - `MSK_IPAR_WRITE_PRECISION` from 8 to 15
- Fixed selection of optimizer for "fixed solve" of a mixed-integer conic problem.

Pyomo

- Fixed compatibility issues with different versions of Pyomo.

SCIP

- New libraries 3.2 (#0d4fc08).
- Changed the default LP solver to SoPlex also for the case where a CPLEX license is available, due to problems when using CPLEX 12.7.0.0 as LP solver in SCIP.

SoPlex

- The GAMS/SoPlex interface has been rewritten and does not use the OsiSpx layer anymore. The solver OSISOPLEX is now an alias for the solver SoPlex.
- SoPlex parameter files can now be used.
- SoPlex can now be warmstarted when only the model instance data changes (e.g., via GUSS).
- New libraries 2.2 (df190de).

3.27.1.5 Tools

CSV2GDX

- Improved error reporting.

GDXDUMP

- New command line option `CSVAllFields` to get all fields (level, marginal, lower, upper, and scale) when writing a variable or equation symbol in CSV format.

MODEL2TEX

- Increased the page width of the txt file generated by the GAMS command line option `docfile` to 32767 (max. value).
- Changed the default encoding to `latin` and added a new command line parameter that allows to change the encoding (`-e=ENCODING`)
- Changed the default format of the symbol tables.
- The JSON style file contains a new property called `columnSetting` that allows to adjust the columns.
- Avoid some unnecessary parentheses in sum and product operators.
- Minor change in the equations subsection that removes several warnings.
- Several minor bug fixes.

GDXXRW

- Fixed a problem when writing a symbol with option merge or clear to a range with `CDim=0` or `RDim=0`.

GMSZIP/GMSUNZIP

- New versions of `Info-ZIP's` tools `zip` (version 3.1c02) and `unzip` (version 6.00). The executable names have been prefixed with "gms" for clear identification.

3.27.1.6 Object Oriented APIs

- New option GAMSOptions.ErrorLog: Maximal number of error message lines written to the log for each error.

.NET

- New examples that demonstrate the use of the API in a graphical environment: TransportGUI, CutstockGUI, FarmGUI.
- Distribute compiled GUI examples in directory <`GAMSDir`>\`apifiles\GUIexamples` on Windows.
- `GAMSWorkspace.AddJobFromFile`: Throw an exception if a given file does not exist.

Python

- The option `GamsOptions._errorlog` has been renamed to `GamsOptions.errorlog` in order to indicate it as public.

3.27.1.7 Model Libraries

GAMS API Library

- **PInterrupt.gms : Test GamsJob interrupt mechanism in object oriented Python API** (54)
- **CSInterrupt.gms : Test GamsJob interrupt mechanism in object oriented .Net API** (55)

GAMS Model Library

- **asyncloop.gms : Transportation Problem with async loop body execution** (411)
- **trnsindic.gms : Fixed Charge Transportation Problem with Indicator Constraints** (412)
- **timesteps.gms : Accessing previous (or next) time steps in an equation fast** (413)
- Modified models:
  - Make use of the new `break` statement to formulate the following models a little nicer: **tsp1**, **tsp4**, **cutstock**, **awktsp**, **sddp**, **sipres**, **allbases**, **qfilter**
  - **tgridmix**: The of logic of submission and collection has been reworked so that the scenario sets do not have to be one dimensional and ordered (contributed by Tom Rutherford).
  - **asyncjobs**: The logic for putting out the log line in time intervals was flawed and has been corrected.

GAMS Test Library

- **emp17.gms : Simple test of one optimizing agent** (693)
- **breakcont1.gms : Test break and continue statements** (694)
- **lindgl04.gms : Test non-convex quadartic program with Lindo(Global)** (695)
- **mcp11.gms : Test marginals for a scaled MCP problem** (696)
- **csv2gdx1.gms : Test csv2gdx on input containing spaces** (697)
- **gdxxrw7.gms : Test merge and clear option for special data layout** (698)
- **scale02.gms : Test that an MCP with scales is rejected when appropriate** (699)
- **emp18.gms : Test JAMS/EMP on implicit variable handling** (700)
- **emp19.gms : Test JAMS/EMP on implicit variable handling** (701)
- **emp20.gms : Test JAMS/EMP on implicit variable handling** (702)
- **emp21.gms : Test JAMS/EMP on implicit variable handling** (703)
- **emp22.gms : Test JAMS/EMP on implicit variable handling** (704)

- **emp23.gms : Test JAMS/EMP on implicit variable handling** (705)
- **emp24.gms : Test JAMS/EMP on implicit variable handling** (706)
- **emp25.gms : Test JAMS/EMP on implicit variable handling** (707)
- **emp26.gms : Test JAMS/EMP on implicit variable handling** (708)
- **gurobi02.gms : GUROBI test suite - general constraints max,min,abs** (709)
- **gurobi03.gms : GUROBI test suite - general constraints and,or** (710)
- **slx01.gms : run tests for different solvelink values** (711)
- **gurobi04.gms : GUROBI test suite - multi objective** (712)

3.27.2 24.8.2 Maintenance release (January 03, 2017)

3.27.2.1 Acknowledgments

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

3.27.2.2 GAMS System

GAMS

- Change for command line parameter MultiPass: If it is set to 2, als errors from `$gdxIn` are ignored now
- Fixed a bug which caused wrong results in some assignments which use symbols that were used out-of-order in previous assignments.

3.27.2.3 Solvers

SCIP

- Added a workaround that allows for using CPLEX 12.7.0.0 as LP solver in SCIP again. For the moment, the default is still to use SoPlex.

3.27.3 24.8.3 Minor release (January 28, 2017)

3.27.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Mohammad R. Hesamzadeh, and Katja Jensen.

3.27.3.2 GAMS System

GAMS

- Fixed potentially wrong values for model attribute `etSolver` for cases where the command line option `solveLink` was set to 1, 2 or 5.

3.27.3.3 Solvers

BARON

- New libraries 17.1.2.
  - More robust links with COIN-OR solvers and a better link with FICO Xpress.
  - Some new RLT cuts are included in relaxations.

DE

- Create a capability error if DE is called to solve a model of type EMP without stochastic information.

DECIS

- Create a capability error if DECIS is called to solve a model of type EMP without stochastic information.

Examiner2

- Fix behavior when the subsolver returns a model status like `19 Infeasible - No Solution`. The model status can be passed unchanged back to GAMS in such a case.

Kestrel

- Adjusted the default URL to https using port 3333.
- Fixed a bug that prevented to specify the protocol in the `neos_server` parameter in an option file. The complete format of the parameter is now `protocol://host:port`.

LINDO

- Create a capability error if LINDO is called to solve a model of type EMP without stochastic information.

MOSEK

- New libraries 8.0.0.53.

SCIP

- Changed the default LP solver back to CPLEX, if available (see also `24.8.1` and `24.8.2` release notes).

3.27.3.4 Tools

GDXDUMP

- Fixed wrong output in case of nested quotes in symbol text.

3.27.3.5 Model Libraries

GAMS Model Library

- **linearne**: Minor fix to model formulation (contributed by Mohammad R. Hesamzadeh).

3.27.4 24.8.4 Minor release (April 10, 2017)

3.27.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Ivan Leung, Tara Rengarajan, and Sajjad Shafiei.

3.27.4.2 GAMS System

GAMS

- Fixed an issue which caused unnecessary memory consumption if many indexed symbols with explicit labels were used.

3.27.4.3 Solvers

BARON

- New libraries 17.4.1.
  - Bug fixes and an enhanced convexity detector.

CPLEX

- New libraries 12.7.1.0.
  - Note that with this update the log generated by Cplex changed in a way that the Cplex options set (either by the user or by GAMS defaults) are displayed. This is intentional and no sign of a problem. So one could see something like this in the log when running GAMS/Cplex:

```
CPXPARAM_Simplex_Limits_Iterations              2000000000
CPXPARAM_TimeLimit                              1000
CPXPARAM_Threads                                1
```

DICOPT

- Fixed serious bug in feasibility pump implementation.

Examiner, Examiner2

- The list of checks to perform for solved models was set incorrectly when the trace option was used.

GUROBI

- New libraries Gurobi 7.0.2.
- Added options MultiObjMethod and MultiObjPre.
- Fixed a problem that lead to wrong error messages when setting the ObjNAbsTol and ObjNRelTol parameters.
- Fixed a problem that reported back the wrong alternative solution.

JAMS

- Fix problem with bilevel models having variables owned by the leader but not appearing in the leader objective or constraints.

Lindo/LindoGlobal

- New libraries 10.0.179.

MOSEK

- New libraries 8.0.0.60.

NLPEC

- Fix problem handling empty constraints (e.g. `f.. eps*x =G= 0`) that appear in MPEC models. Note that such models can easily be produced by JAMS/EMP.

3.27.4.4 Object Oriented APIs

Python

- Added support for Python 3.6.
- Fixed a bug in `GamsJob.run()` that prevented the underlying GAMS model from terminating, if executables spawned by GAMS generate log output that is not captured.

3.27.5 24.8.5 Maintenance release (May 10, 2017)

3.27.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Ana Carolina, Gary Goldstein, Erwin Kalvelagen, Amit Kanudia, Toni Lastusilta, Antti Lehtila, Bruce McCarl, and Hans Kristian Ringkjob.

3.27.5.2 Platforms

- The AIX system is now build on AIX 7.1.

3.27.5.3 GAMS System

GAMS

- Fixed a bug which could cause a crash in particular assignments involving singleton sets or explicit labels. This was introduced with GAMS 24.8.4.
- Fixed problems with the `break` statement:
  - There was a potential crash if break was used in a "sparse loop", e.g., `loop(j$x(j), ...)`.
  - If break was used in a loop with more than one index, e.g., `loop((i,j), ...)`, that loop was treated as multiple loops for the `break` statement, one for each index. Now it is treated as just one loop as intended.
  - Fixed a problem where a loop was not correctly recognized as a loop if it runs over just one fixed element, e.g., `loop(i('i2'), ...)`.

3.27.5.4 Solvers

BARON

- Initialize BARON option Threads with value of GAMS option Threads, if the latter is at least 1.

CBC

- New libraries.
  - Fixed a bug in presolve that caused problems with fixed discrete variables.

GUROBI

- Suboptimal solutions were not reported back to GAMS. This has been fixed.

Lindo/LindoGlobal

- New libraries 10.0.182.

MOSEK

- New libraries 8.0.0.69.

3.28 24.7 Distribution

3.28.1 24.7.1 Major release (March 14, 2016)

3.28.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Alex Dowling, Johannes Hedtrich, Austin Milt, and Salvador Pineda.

3.28.1.2 GAMS System

GAMS

- Be more strict when the syntax `$LOAD par=var.L` is used: This creates an error now, if `par` is not a parameter (if `par` was a variable before, the complete variable `var` was loaded as if the suffix was omitted)
- GAMS Grid Facility (solveLink = 3 or 4)
  - Fixed a bug for the model attribute `objval`: This could have been wrong when the objective variable was scaled.
  - Fixed a bug for the marginals: These could have been wrong when the objective variable was scaled.
- Solving in parallel threads (solveLink = 6)
  - Allow to solve scaled models (model attribute `scaleOpt` is set to 1) as well (this generated an execution error before).

3.28.1.3 Solvers

BARON

- New libraries 16.3.4.
  - Faster LP interfaces.
  - Additional checks on solution reliability of LP and NLP subsolvers.
  - New automatic differentiation routines.
  - Improved handling of memory-intensive problems.
  - Enhanced presolving for continuous and integer programs.
  - New primal heuristic for integer programs.
  - New facilities for solving constraint (non)linear systems (CNS).

CBC

- New libraries.

ConvertD

- The GDX file created with option `Jacobian` contains now a set ANl that indicates the non-linear Jacobian elements.

CPLEX

- For an unbounded model Cplex did not mark the unbounded variables correctly. This has been fixed.

GUROBI

- Gurobi Optimization, Inc. has decided to end the current arrangement that allows us to offer GAMS/Gurobi integrated licenses. GAMS will continue to offer the latest version of Gurobi but users will need to get a Gurobi license from Gurobi Optimization, Inc. plus a GAMS/Gurobi link license from GAMS or use the free OsiGurobi link. This includes evaluation and demo licenses.
  For existing GAMS/Gurobi customers the following transition arrangements have been negotiated. The GAMS/Gurobi integrated license will continue to work as long as the license stays under maintenance. Academic GAMS/Gurobi customers will not be able to renew maintenance on their GAMS/Gurobi integrated license, their license will be changed into a GAMS/Gurobi link license. Academic users can get free Gurobi licenses from www.gurobi.com.
  In the past, academic GAMS/Gurobi link or GAMS/OsiGurobi licenses did not require a Gurobi license installed. This has changed with this release
- New libraries 6.5.1
- The option PreSOS1BigM and PreSOS2BigM were incorrectly classified as integer options. They have been reclassified as real option.

IPOPTH

- Fixed a memory access issue in the linear solver HSL MA86 in case of very large models.

LocalSolver

- New libraries 6.0 (20160308).

MOSEK

- New libraries 7.1.0.49.

SBB

- Fixed a problem with option `UserHeurCall`. This option required that all discrete variables were declared first in the model. This is not necessary anymore.

SCIP

- New libraries 3.2 (#f69c505).
  - The timing mask for parameters `constraints/.../presoltiming`, `presolving/.../timing`, and `propagating/.../presoltiming` changed from 2/4/8 for fast/medium/exhaustive to 4/8/16.
  - Parameter `constraints/SOS1/updateconflpresol` has been renamed to `constraints/SOS1/perfimplanalys`
  - Detailed Changelog.

SOPLEX

- New libraries 2.2 (#12d3858).

3.28.1.4 Tools

GAMS IDE

- Fixed a bug which caused an error when opening the FINLIB (Practical Financial Optimization Models).

MODEL2TEX

- Fixed a bug regarding negative variables.
- Fixed a bug regarding subscripts of symbol names with underscores.
- Fixed a bug that prevented alternative names from being replaced in Ord constructs.
- New parameter -o (–OUTPUT) that allows to specify an alternative output file.
- Use UTF8 encoding.
- The columns of the symbol table have a fixed (equal) width.
- The original name of a symbol that is changed using the JSON style file is displayed in the symbol table.
- The JSON style file is alphabetically sorted by keys.

3.28.1.5 Object Oriented APIs

- Fixed a bug with the property `GAMSOptions.Defines` (.NET), `GAMSOptions.defines` (Java), and `GamsOptions.defines` (Python): When too many entries were added, all of them were ignored.

Python

- Fixed a memory leak in `GamsDatabase`.

3.28.1.6 Model Libraries

GAMS Test Library

- **scale01.gms : Test results of scaled model** (692)

3.28.2 24.7.2 Minor release (July 07, 2016)

3.28.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Peter Alexander, Daniel Balistrieri, Paul Buckland, Denis Hess, Eden Huang, Josef Kallrath, Sherman Robinson, Tom Rutherford, Jhih-Shyang Shih, and Loic Ventre.

3.28.2.2 Platforms

- With the next major release, the minimal glibc version requirement for the GAMS distribution on Linux will be increased to 2.7.

3.28.2.3 GAMS System

GAMS

- Improved the performance for assignments from one large symbol to itself inside a loop, when only one element gets changed, e.g.:

  ```
  loop((j,cty,yrs,sim), ...; par(j,cty,'Total',yrs,sim) = sum(lnd, par(j,cty,lnd,yrs,sim));)
  ```

- Fixed a performance issue which caused very long model generation time for large scalar models.

3.28.2.4 Solvers

BARON

- New libraries 16.5.16.
  - Added option WantDual to indicate whether BARON should make a final call to an NLP solver to try to compute duals if an inexpensive method of calculating them failed.
- CompIIS option did not print IIS into listing file. This has been fixed.

CBC

- New libraries.

CPLEX

- New libraries 12.6.3.0.2.
- A memory leak in Cplex has been fixed that showed up in combination with the OO-API GAMSModelInstance class.
- Fixed a problem with CplexD option FreeGamsModel.
- Fixed a problem with `MipStart` in combination with SemiInt or SemiCont variables with lower bound 0.

GUROBI

- New libraries 6.5.2.

JAMS

- Fix handling of objectives in optimizing agents of equilibrium systems in cases where the objective coefficient was not 1 or -1, and in the equation level returned for the objective row(s).

LocalSolver

- New libraries 6.0 (20160625).

Mosek

- New libraries 7.1.0.52.

OsiMosek

- Fixed handling of gap tolerances and node limit of MIP solves.

SBB

- Fixed a problem with models that have domain violations in the root NLP.
- Clearly identify text written to the listing file as output from the root solver.

SCIP

- New libraries 3.2 (#e99d344).

SoPlex

- New libraries 2.2 (#3c5e86f).

PATH

- Fixed a problem with bogus report of empty rows/cols when benchmarking a model, i.e. solving with iterlim=0.

XPRESS

- Maintenance update of Optimizer libraries for Windows, Linux, and Mac to 28.01.14.
- Several minor bug fixes that affect correctness and performance in some uncommon cases.

3.28.2.5 Tools

- Increase maximum available memory for GDXVIEWER, MDB2GMS, SQL2GMS, XLS2GMS, GDX2HAR, and HAR2GDX to 3GB.

CSV2GDX

- Fixed a problem with labels with trailing blanks. Labels with trailing blanks potentially resulted in a broken GDX file.
- Introduced the symbolic constant `LastCol` for the Values parameter.
- Report duplicate keys.

GDXXRW

- Fixed a problem with unnecessary memory consumption when writing with option clear or merge.
- Symbols and text were not written in sequence as instructed. This has been fixed.
- Added command line parameter ReCalc that by default prevents frequent recalculations in Excel while writing to the spreadsheet.

3.28.2.6 Object Oriented APIs

- Fixed a bug which did not clear a pending Ctrl-C event, so that it might have been applied to a following GAMSModelinstance.Solve by accident.
- Fixed a problem, that caused a crash instead of a GAMSException in some rare cases.

Python

- Fixed a minor bug regarding the names of GamsJob listing files.

3.28.2.7 Model Libraries

- Some URLs in various models have been updated.

3.28.3 24.7.3 Maintenance release (July 11, 2016)

3.28.3.1 Acknowledgments

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

3.28.3.2 Tools

CSV2GDX

- Fixed a bug which caused an error when reading sets from a csv file.

MODEL2TEX

- Fixed a bug regarding wrongly generated equations in the generated LaTeX file.

3.28.4 24.7.4 Minor release (September 19, 2016)

3.28.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Manuel Alvarez, Felix Färber, Ricardo Manuel Pinto de Lima, and Tara Rengarajan.

3.28.4.2 GAMS System

GAMS

- Fixed a problem when using extrinsic function libraries using API version 1.
- Extend possible range of `n` in `fact(n)` at compile time. It was limited to 20 in the past. Now it can be up to 166 (as in assignments at execution time).

Installer

- New default group name (`GAMS xx.x (xx bit)`) for the start menu on Windows 10.

3.28.4.3 Solvers

BARON

- New libraries 16.8.24.
  - Small performance improvements and robustification of LP solver interfaces and bugfixes.
  - The starting point is now utilized before invoking projection and other initialization strategies.
  - The option DeltaTerm is now recognized again.

Couenne

- New libraries.

CPLEXD

- The option MipStart accepts now values other than 0 and 1. The value describes the effort level Cplex uses to determine a MipStart from the starting variable levels. The value 2 is interesting because it just checks for feasibility of the MipStart. In this case the level of all variables, not only the discrete ones, are passed on to Cplex.

DE

- DE crashed if multiple joint variables were present in one stage. This has been fixed.

Lindo/LindoGlobal

- New libraries 9.0.293.

Mosek

- New libraries 7.1.0.55.

SCIP

- New libraries 3.1 (#6b9196f).

SoPlex

- New libraries 2.2 (#074950a).

3.28.4.4 Tools

CSV2GDX

- Fixed a bug which lead to a rejection of unquoted labels with spaces. This bug was introduced in 24.7.2.

GDXXRW

- Fixed a bug which caused an error message when reading a sheet with a non-empty range that contained no data.

3.28.4.5 Object Oriented APIs

- `GAMSModelInstance.Instantiate`: Skip creation of GDX file, which was unreachable from within the APIs anyway.

Python

- Added support for Python 3.4 on Mac OS X.
- Fixed a problems with Exceptions in `GamsWorkspace`.

3.29 24.6 Distribution

3.29.1 24.6.1 Major release (January 18, 2016)

3.29.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Etienne Ayotte-Sauvé, Michael Ferris, Per Ivar Helgesen, Erwin Kalvelagen, Martha Loewe, Geoff Moore, and Renger van Nieuwkoop.

3.29.1.2 Platforms

- The minimal supported Mac OS X version has been increased to 10.9 (Mavericks). COIN-OR solvers, Gurobi, SCIP, and SoPlex now require a system with at least Mac OS X 10.9. The GAMS base system, tools, and other solvers are still working on Mac OS X 10.7 and 10.8, but may stop working in a future release without extra announcement.
- The installer for Wine on Linux has been dropped.

3.29.1.3 GAMS System

GAMS

- Increased controlled set nesting limit from 120 to 300.
- New option SHUFFLE.
- New option for command line parameter MultiPass:
  - 0: standard compilation
  - 1: check-out compilation
  - 2: as 1 and in addition skip `$call` and ignore errors because of missing files with `$include` (NEW)
- Allow to load variable and equation attributes into a parameter at compile time, e.g. `$LOAD par=var.L` (par being a parameter and var a variable)

- Solving in parallel threads (beta feature)

  A new variant for the solveLink option and model attribute has been added. If `solveLink` is set to 6 (compile time constant `%solveLink.asyncThreads%`), GAMS does not wait for the solver to return and does not collect the result when a solve statement is executed. Instead, the model is generated and then passed to the solver in a separate thread while GAMS continues the execution. This way, multiple models can be solved in parallel and the results can be collected later.

  This is similar to the Grid Facility (`solveLink = %solveLink.asyncGrid%=3`) with the difference, that the solver does not operate in its own process space but in a separate thread, which allows efficient in-memory communication between GAMS and the solver (like it is done with `solveLink = %solveLink.loadLibrary%=5`). After the solve statement, one can store a handle of the model instance (using the model attribute mymodel.handle) and use the same functions that are used for the Grid Facility to collect the solution and deal with the model instance: `HandleCollect(handle)`, `HandleStatus(handle)`, and `HandleDelete(handle)`.

  The new function `ReadyCollect(handleParameter [, maxWait])` can be used to wait until a model is ready to be collected. It can be used for both `solveLink = %solveLink.asyncThreads%` and `solveLink = %solveLink.asyncGrid%`. The arguments and return codes are:

  - Arguments
    - `handleParameter`: parameter holding handles of model instances to wait for
    - `maxWait`: maximum time to wait (`+INF` if omitted)
  - Return Codes
    - 0: (one of) the requested job(s) is ready
    - 1: no active job to wait for
    - 2: no handle provided
    - 3: invalid handle
    - 4: user specified time-out when using a `solveLink = %solveLink.asyncThreads%` handle
    - 5: user specified time-out when using a `solveLink = %solveLink.asyncGrid%` handle
    - 8: unknown error (should not happen)

  The new option threadsAsync (available on the command line and with the option statement) sets the maximum number of threads that should be used for the asynchronous solves. If a negative number is set, this specifies how many of the available processors on the host machine should not be used. The default setting is -1.

  Currently, the following solvers can be used with `solveLink = %solveLink.asyncThreads%`:

  - CONOPT
  - CPLEXD
  - GUROBI
  - MOSEK
  - OSICPLEX
  - OSIGUROBI
  - SCIP

  If another solver is selected, `solveLink = %solveLink.asyncGrid%` will be used instead (which is noted in the log).

  An example of how this new feature can be used, can be seen in the GAMS Model Library model **tgridmix**.

Stochastic Programming with EMP

- Fixed a bug which caused wrong expected values for parametric distribution using the `RandVar` keyword.

3.29.1.4 Solvers

ANTIGONE

- New libraries.

BARON

- The handling of branching priorities in a GAMS/BARON options file has changed. Priorities should now always be given in the GAMS convention.

Convert

- GAMS branching priorities are now converted to BARON branching priorities when writing a BARON input file.

CPLEX

- New libraries 12.6.3.
- Cplex and CplexD now report the deterministic time spend (in `ticks` not in seconds) in the model attribute `ETAlg`.
- IBM's Cplex cloud offering "DOcloud" can be accessed via the "Kestrel solver".

DICOPT

- New option usexinit instructs DICOPT to start the NLP sub-solves from the user supplied input point.

GUROBI

- New libraries 6.5.
- New option PreMIQCPForm that determines the format of the presolved version of an MIQCP model.
- New option `WorkerPort` specifies a non-default port number for the distributed worker machines.
- New option VarHint. The variable hints communicated through level and tryint will affect the heuristics that Gurobi uses to find feasible solutions, and the branching decisions that Gurobi makes to explore the MIP search tree.
- GAMS/Gurobi supports solving models in the Gurobi Instant Cloud.
- Fixed a problem with option IIS for models with SOS variables.

IPOPT

- MKL Pardiso is now available as linear solver on Mac OS X, too.

KESTREL

- The Kestrel client provides experimental access to "IBM's DOcloud" offering.

PATH

- Minor fix for zero tolerance in the basis reset routine of the Lemke method.

Sulum

- Sulum will be dropped from the distribution with GAMS 24.8.

XPRESS

- Updated Optimizer libraries for Windows and Linux: 28.01.05 → 28.01.10.
- Several minor bug fixes that affect correctness and performance in some corner cases.

3.29.1.5 Tools

ASK

- New feature SelectDirectory.

GAMS IDE

- Sorting by symbol name is no longer case sensitive.
- GDX viewer can now show numbers with full precision.
- The option editor no longer shows the dot options.
- Fixed a bug where the cursor was not shown after double-clicking on a red line.

GDX2SQLITE

- New version 0.7.
  - Added option varchar to export character columns as `VARCHAR(255)` instead of `TEXT`.
  - Better names of columns when option small is used.
  - Adding timing info.

GDXMERGE

- Protect against very large symbols causing memory errors.
- Added option EXCLUDE to exclude symbols from being merged.

GDXXRW

- An empty range is no longer an error.

SQL2GMS

- In the old version double quotes were removed when reading a command file. In this version we keep double quotes to be able to escape SQL names (table names, column names). E.g. we now can handle non-standard names by double quoting them in a query. Depending on the database you can do:

  ```
  q=select "Some COLUMN" from "This Table"
  ```

  Note that some databases (such as SQL Server) use [ ] for this goal.

  In the special case where you write:

  ```
  q="select c from t"
  ```

  the surrounding double quotes are removed before passing the query on to the database.

MODEL2TEX

- If an identifier is changed using the specifications in the JSON style file, underscores are no longer changed from "_" to "_". This makes it possible to use subscripts when replacing an identifier.

3.29.1.6 Object Oriented APIs

Java

- Fixed a bug in `GAMSModelInstance.copyModelInstance` method when duplicating scratch directory.

Python

- New example `transport8a.py`.

3.29.1.7 Model Libraries

GAMS EMP Library

- **nbcontindep**: use sampling for continuous distributions if another solver than Lindo is selected
- **nbcontjoint**: use sampling for continuous distributions if another solver than Lindo is selected
- **nbsimple**: use discrete distribution

GAMS Test Library

- **shuffle1.gms : Test for option shuffle** (690)
- **sl601.gms : Check correct behavior when using solveLink=6** (691)

3.30 24.5 Distribution

3.30.1 24.5.1 Major release (September 23, 2015)

3.30.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Guillaume Erbs, Michael Ferris, Mahbube Habibian, Josef Kallrath, Jean Mercenier, Stan Peter, Ingmar Schlecht, and Mark Thissen.

3.30.1.2 Platforms

- Support for Windows XP has been dropped completely with this release (as announced).

3.30.1.3 GAMS System

GAMS

- New model attributes
  - `maxInfes`: Maximum of infeasibilities
  - `meanInfes`: Arithmetic mean of infeasibilities
- New option solver. This simplifies the selection of the (default) solver for multiple model types.
  - The command line option `solver=abc` initializes the default solver for the model types solver `abc` is capable of to `abc`. This initialization is done before the default solvers of individual model types are set via command line options. So a command line with `lp=conopt solver=bdmlp` will first set BDMLP as the default solver for model types LP, RMIP, and MIP (these are the model types BDMLP can handle) and then reset Conopt as the default solver for LP. The order of these parameters on the command line has no impact (i.e. `lp=conopt solver=bdmlp` behaves identically to `solver=bdmlp lp=conopt`). If multiple occurrences of option `solver` appear, the last one sets the option as it is with other options, including `LP`, `MIP`, ...
  - The solver for multiple model types can be set via the `Option solver=abc;` in the GAMS model source code. This sets the solver for model types `abc` can handle to `abc`. With the `option solver=abc;` the order among other solver setting options is significant. For example, `option lp=conopt, solver=bdmlp;` will first set the solver for LP to Conopt and in the next step to BDMLP because BDMLP is capable of handling model type LP. Setting `solver` twice can also make sense: `option solver=conopt, solver=cbc;` will result into setting the solver for model types CNS, DNLP, NLP, QCP, RMIQCP, and RMINLP to Conopt and the solver for model types LP, RMIP, and MIP to CBC.
- When using $LOADDC the reporting of domain errors has been improved.
- This release features several improvements of the execution system. The following lists some (extreme) examples:
  - Improved performance when indices of accessed symbol are in "wrong" order, e.g.:

```
$if not set size $set size 10000
set v  / v1*v%size% /
e  / e1*e%size% /;
alias(v,w);
set evw(e,v,w) /#e:#v:#w/
    vw(v,w);
vw(v,w) = sum(e, evw(e,v,w));
```

The final assignment can be done in less than 1% of the time required by GAMS 24.4.

– Improved performance when assigning to non-empty symbols when the assignment is driven by their domain, e.g.:

```
$if not set size $set size 5000
set i / i1*i%size% /; alias (i,j);
parameter a(i,j), b(i,j);
a(i,j)$(uniform(0,1)<0.95) = 1;
b(i,j) = not a(i,j);
a(i,j) = b(i,j);
```

The final assignment can be done in just ∼5% of the time required by GAMS 24.4.

– Improved performance when executing certain combinations of mappings, e.g.:

```
$if not set size1 $set size1 8000
$if not set size2 $set size2 24
set hi          / 1*%size1% /
    ti          / 1*%size2% /
    h           / h1*h%size1% /
    t           / t1*t%size2% /
    hmapx(h,hi) / #h:#hi /
    tmapx(t,ti) / #t:#ti /;
parameter xi(hi,ti), x1(h,t);
xi(hi,ti) = uniform(0,1);
x1(h,t) = sum((hmapx(h,hi),tmapx(t,ti)), xi(hi,ti));
```

The final assignment can be done in just ∼2% of the time required by GAMS 24.4.

– Improved performance when "searching" a lot in a large symbol, e.g.:

```
$if not set size1 $set size1 50
$if not set size2 $set size2 200000
$if not set size3 $set size3 150000
set f /f1*f%size1%/
    j /j1*j%size2%/
    l /l1*l%size3%/
    flmap(f,l);
flmap(f,l)=uniform(0,1)<0.25;
parameter jlpar(j,l) /#j:#l 10/
          fjpar(f,j);
fjpar(f,j) = sum(l$flmap(f,l), jlpar(j,l));
```

The final assignment can be done in just ∼1% of the time required by GAMS 24.4.

Documentation

- The complete GAMS technical documentation (release and installation notes, user's guides, solver and tools manuals, API reference manuals) is now available in HTML format with a unified table of contents. The documentation is available in the following forms:
  - The online GAMS documentation provides search and keyword indexing functionalities in addition to navigating the GAMS documentation.
  - The offline documentation allows access without a network connection:
    * The main navigation page can be found at `[GAMS system directory]/docs/index.html` (not linked from the GAMS IDE).
    * For Windows, additionally a Microsoft Compiled HTML Help file is available at `[GAMS system directory]/docs/gams.chm` and linked from the GAMS IDE Menu: `Help -> GAMS Documentation`.
- The following documents are still available in PDF format: GAMS User's guide, McCarl Expanded GAMS User's guide, all solver manuals, and GAMS GDX API.

Installer

- New way to install GAMS on Mac OS X using a DMG file.

3.30.1.4 Solvers

ANTIGONE

- Added option conopt_optfile to set name of options file to be used for CONOPT calls in ANTIGONE.

BARON

- Now also available for Mac OS X.
- New libraries 15.9.22.
- New NLP solver for local search: FilterSD.
- Options:
  - New option Threads: Number of cores used for solution of MIP subproblems.
  - New option IISOrder: Order in which constraints are considered in the search for an IIS.
  - Options `ConTol` and `IntTol` removed. Tolerances are now set by AbsConFeasTol, RelConFeasTol, AbsIntFeasTol, and RelIntFeasTol.
  - Options `MultMSize`, `MultRel`, `NLPDoLin`, `MipGap`, `MipNodes`, `MipRel`, `NoutIterMip`, `BilRel`, `Cvxbt`, `CvxInitOA`, `CvxRel`, `MipCuts`, and `RLTRel`: Algorithmic features that have been parameterized are now handle in an automatic way.
- The time limit is enforced based on CPU time for single-threaded jobs and based on wall-clock time for multi-threaded jobs.
- Simplified the screen output (eliminated some details and simplified time to a real number in seconds).
- Updated CBC to 2.9.5.
- Updated Ipopt to 3.12.3.

CBC

- New libraries 2.9.
  - Introduced specialized branching methods for dealing with "big Ms".
  - Introduction of conflict cuts (off by default, enable with option conflictcuts).
  - Introduced new methods for dealing with symmetry.

Conopt

- New libraries 3.17A.
  - Corrected problem with options Ls2ndi and Lmusdf.
  - Systems errors related to very tight memory (984) and the inversion routine (2027) have been removed.
  - Three new options have been added to allow the modeler to see the order of the pre-triangular and post-triangular part of the model and the definitional equations:
    - PRPRET: Flag for printing the pre-triangular part of the model.
    - PRDEF: Flag for printing the defined variables and their defining constraints.
    - PRPOST: Flag for printing the post-triangular part of the model.

Couenne

- New libraries 0.5 (major update).
- Better handling of function `signpower(x,k)` for positive integer k≤10.
- The default values for the following options have changed:
  - `cont_var_priority` from 2000 to 99.
  - `int_var_priority` from 1000 to 98.
  - `feas_pump_milpmethod` from -1 to 0.
  - `feas_pump_poolcomp` from 0 to 4.
  - `output_level` from 0 to 4.

CPLEX

- New option FreeGamsModel. This option preserves memory by dumping the GAMS model instance representation temporarily to disk. This option is available in CplexD only.

GUROBI

- New libraries 6.0.5 (technical release).
- New option FreeGamsModel. This option preserves memory by dumping the GAMS model instance representation temporarily to disk.

GUSS/ScenarioSolver

- The set of GUSS model attributes (`Set ma "GUSS Model Attributes" / System.GUSSModelAttributes /;`) has changed:
  - The attribute `NumNOpt` has been removed.
  - The attributes `MaxInfes` and `MeanInfes` have been added.

IPOPT

- New libraries 3.12 (minor changes).

JAMS

- SubSolver option fixed: it was sometimes ignored.

Kestrel

- The option file can be omitted. In this case, the Kestrel call is done with default settings.

Lindo/LindoGlobal

- Dropped Lindo/LindoGlobal libraries for Intel Solaris as announced previously.
- New libraries 9.0.225.

LocalSolver

- New libraries 5.5.
  - Improved accuracy and performance on numerical or mixed-variable optimization problems.

MINOS

- Adjusted default upper bound on the superbasics limit from 500 to 5000 and consider the number of nonlinear constraint variables, not just the nonlinear objective variables, when computing the superbasics limit: memory is plentiful and cheap.
- Fixed case of memory corruption that resulted when using an initial point containing very many superbasic variables. If the initial point contains more than 5000 superbasics, limit the Hessian dimension to 5000 even though the superbasic limit is larger.

MOSEK

- New libraries 7.1.0.33.
- The GAMS option iterlim now sets the iteration limit for both simplex and interior point algorithms.
- More MOSEK options are now available in GAMS/Mosek:
  - MSK_IPAR_MIO_CUT_CG: Controls whether CG (Chvatal-Gomory) cuts should be generated.
  - MSK_IPAR_MIO_CUT_CMIR: Controls whether mixed integer rounding cuts should be generated.
  - MSK_IPAR_MIO_PROBING_LEVEL: Controls the amount of probing employed by the mixed-integer optimizer in presolve.
  - MSK_IPAR_MIO_RINS_MAX_NODES: Controls the maximum number of nodes allowed in each call to the RINS heuristic.
  - MSK_DPAR_MIO_TOL_MAX_CUT_FRAC_RHS: Maximum value of fractional part of right hand side to generate CMIR and CG cuts for.
  - MSK_DPAR_MIO_TOL_MIN_CUT_FRAC_RHS: Minimum value of fractional part of right hand side to generate CMIR and CG cuts for.
  - MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT: If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation.

OS

- Now a hidden solver. Will be dropped from the distribution at some time (without further notice).

PATH

- New libraries 4.07.03.
- Adjusted to use wall-clock time on all platforms.
- Time limit checked more frequently, e.g. at each pivot.
- The presolve has been extended to find more reductions.
- Minor bug fixes.

SCIP

- New libraries 3.2 #c9c4375 (major update):
  - New presolvers "tworowbnd", "dualagg", "implfree", "redvub", and "stuffing", and improved presolve for ranged- and equality-type linear constraints.
  - Presolving levels FAST, MEDIUM, and EXHAUSTIVE are now used to better coordinate the various presolvers.
  - Generalized upgrade from (SOC-representable) quadratic constraints to SOC constraints.
  - New primal heuristics "distribution diving", "indicator", and "bound", improved clique and variable bound heuristics, and adjusted diving heuristics to solve fewer LPs.
  - New branching rules "distribution", "multaggr", and a new rule for SOS1 constraints.
  - New reliability notions and improved treatment of nonlinearities in hybrid reliability pseudo cost branching.
  - New separator "eccuts" for generating edge-concave cuts for quadratic constraints and improved separation for convex quadratic constraints.
  - Decreased total memory usage by using more buffer data structures.
  - Improved propagation and separation for SOS of type 1 by using information from a conflict graph.
  - See also the `full release notes`, the `change log`, and the `technical report`.
- The following options were removed or replaced:
  - `constraints/.../delaypresol` and `constraints/.../timingmask` replaced by `constraints/.../presoltim` and `constraints/.../proptiming`.
  - `presolving/domcol/singcolstuffing` replaced by `presolving/stuffing/...`.
  - `presolving/.../delay` replaced by `presolving/.../timing`.
  - `propagating/.../presoldelay` replaced by `propagating/.../presoltiming`.
  - `propagating/obbt/maxlookahead` removed.
- For the following options, the default value changed:
  - constraints/SOS1/sepafreq from 0 to 10.
  - "heuristics/clique/minfixingrate" from 0.5 to 0.25.
  - "heuristics/vbounds/minfixingrate" from 0.5 to 0.25.
  - heuristics/actconsdiving/maxdiveavgquotnosol from 0 to 1.
  - heuristics/actconsdiving/maxdiveubquotnosol from 0.1 to 1.
  - heuristics/dins/nwaitingnodes from 0 to 200.
  - lp/rowrepswitch from -1 to 1.2.
  - presolving/abortfac from 0.0001 to 0.001.
  - presolving/restartfac from 0.05 to 0.025.
  - presolving/immrestartfac from 0.2 to 0.1.
  - presolving/dualinfer/priority from 20010000 to -200.
  - propagating/obbt/itlimitfactor from 5 to 10.

SoPlex

- New libraries 2.2.0 (major update).

SULUM

- New libraries 4.3.892.
  - Several bug fixes in both the MIP solver and the LP solver.
  - Improved numeric stability, degeneracy handling, and perturbation scheme in LP optimizer.
  - Added a new combined pricing scheme to the dual simplex optimizer.
  - Improved presolve and restart in MIP optimizer.

XPRESS

- New libraries for XPRESS v7.9: Optimizer 28.01.05.
  - Significantly improved linear algebra routines for the simplex solvers improving efficiency of a wide range of problems.
  - Improvement heuristics called more often.
  - The MIP log now provides information which heuristic finds a solution.
  - Improved linear dependency checker for large problems.
  - Improved scaling, including scaling of big-M type rows and of the Curtis-Reid scaling option.
  - Improved inference-learning from infeasible subproblem during the MIP search.
  - Improved presolver for quadratic instances.
  - Improved handling of Special Ordered Sets.
  - Improved propagation of conflict cuts.

3.30.1.5 Tools

GAMS IDE

- In the model library browser the IDE may provide hints about the content of the column when hovering over the headers.
- Option to specify file extensions for files that will be reloaded without a confirmation dialog.

GDXDIFF

- When renaming the temporary file to `gdxdiff.gdx` fails, issue a ViewClose command in case the file is open in the GAMSIDE and try to rename again.
- If no difference was found, issue a message in the log indicating this.
- Add a set with two elements with explanatory text of the two files compared.

GDXMERGE

- When a filename cannot be used as a UEL, use a generated name instead.

GDXXRW

- New options `IgnoreRows` and `IgnoreColumns` to ignore a set of rows or columns for a symbol. Rows can be specified as `IgnoreRows=1,4:5` and columns `IgnoreColumns=A,D:F` or `IgnoreColumns=1,4:6`.

MODEL2TEX

- The beta version of this tool allows the automatic generation of LaTeX code that documents a given GAMS model.

MPSGE

- The documentation of MPSGE has moved again. It can now be found in the User's Guide.

3.30.1.6 Object Oriented APIs

- New examples `SpecialValues` and `Clad` (Java: `specialvalues/SpecialValues.java`, `clad/Clad.java`; Python: `special_values.py`, `clad.py`).
- New functions to retrieve models from the GAMS API Library and the Nonlinear Optimization Applications Library:
  - .NET: `GAMSWorkspace.ApiLib` and `GAMSWorkspace.NoaLib`.
  - Java: `GAMSWorkspace.addJobFromApiLib` and `GAMSWorkspace.addJobFromNoaLib`.
  - Python: `GamsWorkspace.apilib` and `GamsWorkspace.noalib`.

Python

- Added support for Python 3.4 (Windows and Linux only). The examples have been changed to be compatible with all supported Python versions.

3.30.1.7 Model Libraries

NOALIB - Nonlinear Optimization Applications Using the GAMS Technology

- This new library by Neculai Andrei has been added to the GAMS system. This is a collection of the models based on the book Nonlinear Optimization Applications Using the GAMS Technology by Neculai Andrei. The library contains a wide spectrum of nonlinear optimization applications expressed in GAMS. The book and library emphasize the local solutions of the large-scale, complex, continuous nonlinear optimization applications, and the abundant examples in GAMS are highlighted by those involving ODEs, PDEs, and optimal control. The collection of these examples will be useful for software developers and testers.
- You can retrieve the individual models through the IDE model library browser or via the command line utility `noalib`.

GAMS API Library

- **CSSpecialValues.gms : Test handling of Special Values in object oriented C# API** (47)
- **PSpecialValues.gms : Test handling of Special Values in object oriented Python API** (48)
- **JSpecialValues.gms : Test handling of Special Values in object oriented Java API** (49)
- **CSNUnit.gms : Compiles and runs NUnit tests for object oriented C# API** (50)
- **CSClad.gms : Test changing solver options while running using the interrupt method** (51)
- **PClad.gms : Test changing solver options while running using the interrupt method** (52)
- **JClad.gms : Test changing solver options while running using the interrupt method** (53)

GAMS Data Library

- **invert1**: Pass `gams.sysdir%` to R script to make linkage between GAMS and R explicit.
- **gdxmrw_tr3**: Canonical form LP created in Matlab and solved via `gams()` Mex-function.
- **gdxmrw_tr4**: Better example of `gams()` usage.
- **gdxmrw_tr5**: Better example of `gams()` usage.

GAMS EMP Library

- **transecs**: Fix formulation as embedded complementarity system and provide equivalent alternative as single-agent equilibrium system.
- **farmnbd.gms : The Farmer's Problem - Stochastic with NBD** (100)

GAMS Model Library

- In the IDE the GAMS Model Library browser has now a `Lic` column indicating the license requirement of a model. The letters **D** and **G** indicate that the model does not require a license. Models with **G** can even be solved by a global solver without a license (the demo limit for global solvers is 10 variables and 10 equations). A letter **L** indicates that a license is required.
  - **tgridmix**: Fix the logic for sleeping. Only do that if no more jobs to be scheduled or all cores are busy.

GAMS Test Library

- **asynntrp**: Make this model work under Unix that have the `pstree` utility available.
- **call6.gms : Call GAMS in a folder containing a %** (674)
- **cmexrc01.gms : Trigger unexpected cmexRC error** (675)
- **rs02.gms : Solving Three-dimensional Noughts and Crosses using Cplex and Gurobi distributed MIP** (676)
- **single04.gms : Check handling of singleton sets assigned and referenced in a loop** (677)
- **model2tex1.gms : Test that model2tex produces a tex file** (678)
- **exmcp6.gms : External Equation - Example MCP 6** (679)
- **scensol6.gms : Test execute_loadhandle for GUSS/GRID** (680)
- **idxperm1.gms : Check correct behavior when permuting indices in model generation** (684)
- **idxperm2.gms : Check correct behavior when permuting indices in loop etc** (685)

3.30.2 24.5.2 Maintenance release (September 29, 2015)

3.30.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Kevin Doran and Renger van Nieuwkoop.

3.30.2.2 GAMS System

GAMS

- Fixed a bug which caused a crash for certain assignments, in particular it had to be an assignment to a symbol that was also referenced on the RHS, with permuted indices and we actually set some of the records to zero which were non-zero before.

3.30.2.3 Object Oriented APIs

.NET

- Fixed a bug that lead to a crash on Linux when the GAMS system directory is a symbolic link. (This is for Mono only.)

Python

- Fixed a bug that lead to a crash on Linux when the GAMS system directory is a symbolic link. The property `GamsWorkspace.system_directory` now always returns the canonical path with all symbolic links resolved.

3.30.2.4 Model Libraries

GAMS Test Library

- **idxperm3.gms : Check correct behavior when permuting indices of symbol used on LHS and RHS** (686)

3.30.3 24.5.3 Maintenance release (October 01, 2015)

3.30.3.1 Acknowledgments

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

3.30.3.2 GAMS System

GAMS

- Fixed a bug which caused wrong results in some cases of out-of-order assignments

3.30.4 24.5.4 Maintenance release (October 15, 2015)

3.30.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Per Ivar Helgesen and Jan Imhof.

3.30.4.2 GAMS System

GAMS

- Fixed a bug which caused wrong results in some cases of out-of-order assignments, in particular it had to be an assignment using the same controlling set more than once in one symbol, e.g. `p(i,j,i)`.
- Fixed a bug when handling suffixes (e.g. .val) of Singleton Sets in Equations.

3.30.4.3 Object Oriented APIs

.NET

- Fixed a bug with the property `GAMSOptions.IDir`.

3.30.4.4 Model Libraries

GAMS Test Library

- **single05.gms : Check correctness of set attributes for singleton sets** (687)
- **idxperm4.gms : Check correct behavior when permuting indices of symbol using same controlling set multiple times** (688)

3.30.5 24.5.5 Maintenance release (November 25, 2015)

3.30.5.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Albrecht, Stefan Kemnitz, Martha Loewe, Thomas Maindl, Bruce McCarl, and Andres Ramos.

3.30.5.2 GAMS System

GAMS

- GAMS now always reports the upper bound that is sent to the solver in the column listing and solution report. Hence, with IntVarUp=1 GAMS prints the value of 100 for integer and semiint variables that are at default bound value +INF.
- Fixed a bug which could cause a crash when referencing attributes of equations (e.g. marginals) after a model was solved with a non-default solveLink setting and there was no solution returned.

Stochastic Programming with EMP

- Fixed a bug which caused wrong sample sizes if the sample keyword was used more than once and with different sample sizes.

3.30.5.3 Solvers

Cbc

- New libraries.

CplexD

- Fixed a problem with the computeserver option. The value for the computeserver option is limited to 255 characters. In order to specify a longer list of workers, one can now use multiple lines with the computeserver option.

Ipopt

- New libraries.

LocalSolver

- New libraries 5.5 (20151028).

Mosek

- New libraries 7.1.0.41.

SCIP

- New libraries 3.2 (#e9a5ca7).
  - Removed options `constraints/SOS1/bipbranch`, `constraints/SOS1/neighbranch`, and `constraints/SOS1/sos1branch`.

SoPlex

- New libraries 2.2 (#f17b9e7).

3.30.5.4 Tools

put_toexcel, put_tohtml

- Fixed a problem with `put_toexcel` and `put_tohtml`.

3.30.6 24.5.6 Maintenance release (November 27, 2015)

3.30.6.1 Solvers

COUENNE

- New libraries.

IPOPT, BONMIN, SCIP

- Fixed issue that Ipopt and Bonmin always read `ipopt.opt`, despite of the `optfile` setting in GAMS.

3.31 24.4 Distribution

3.31.1 24.4.1 Major release (December 20, 2014)

3.31.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Erwin Kalvelagen, Jeff Linderoth, and Erkka Rinne.

3.31.1.2 GAMS System

GAMS

- Fixed a bug causing a potential crash when solving an EMP model having an open file handle without an external file name at the same time.
- The `execMode` setting is not ignored by `put_utility` anymore.
- Added `gbin/md5sum.exe` to Windows distribution. This utility allows users to verify the file integrity of newly downloaded GAMS installation files.
- The `gbin_new` directory will be removed with the next distribution.

Installer

- Windows installer removes the GAMS entry in the current user hive of the registry, if GAMS is installed for all users. Previously, the hive for the current user remained unchanged in this situation.

Extrinsic Function libraries

- The new extrinsic function library `parcclib` was added to the system. This library demonstrates how to access the GAMS parameter file and use it's information through the GAMS Option Object. Further information can be found in the GAMS User's Guide, Appendix J.
- Allow up to 20 arguments (used to be 10).
- Support for extrinsic function libraries that do not provide gradient and/or Hessian values. GAMS uses finite differences (same method as for `.gradn` and `.hessn`) to approximate the derivatives, even inside a solver. The example function library `trilib` now implements function `Sine` without any derivatives and leaves their calculation to GAMS.
- New GAMS options have been introduced to parameterize the numerical derivative calculations. While these are most important for extrinsic functions, they also apply to the `.gradn`/`.hessn` calculations of intrinsic functions.
  - `FDDelta`: step size in the numeric gradient and Hessian calculation. For single argument functions, GAMS evaluates the function at f(x-d) and f(x+d) for the numerical gradient. If function values are used for the numerical Hessian, GAMS evaluates at f(x-2d), f(x), and f(x+2d). For multi argument functions, the same is done for the components of the input argument vector. The default for `FDDelta` is 1e-5. This option can be set from the command line, in an option statement, and as a model attribute.
  - **Note:** In previous releases it was possible to set the step size for `.gradn`/`.hessn` using the option `Real1`. This does not work anymore.
  - `FDOpt`: The option value packs two option in the different digits: *ij*. The *i* digit indicates if scaling of the step size (`FDDelta`) by the value of the input argument should be turned off. If *i*=0 (scaling on) the following step size is used: `max(1,|x|)*FDDelta`. The *j* digit is mostly for testing, but has one setting that might be relevant when the extrinsic function provides gradient but no Hessian values. The numerical derivatives routine in this case uses the gradient calculation from the extrinsic function to approximate the Hessian. If the gradient is expensive to calculate compared to a function evaluation, it could be beneficial to use multiple function values to approximate the Hessian. In this case set the *j* digit to 1. Here are all possible values for this option:
    - 0: All derivatives analytically if available, for numerical Hessian use gradient values, scale delta

    - 1: All derivatives analytically if available, for numerical Hessian use function values, scale delta
    - 2: Gradient analytically, force Hessian numerically using gradient values, scale delta (testing only)
    - 3: Gradient analytically, force Hessian numerically using function values, scale delta (testing only)
    - 4: Force gradient and Hessian numerically, scale delta (testing only)
    - 10: Same as 0, but no scale of delta
    - 11: Same as 1, but no scale of delta
    - 12: Same as 2, but no scale of delta (testing only)
    - 13: Same as 3, but no scale of delta (testing only)
    - 14: Same as 4, but no scale of delta (testing only)
  - **Note:** In previous releases it was possible to deactivate the scaling for `.gradn`/`.hessn` using the option `Integer1`. This does not work anymore.

- Renamed and better documented the `EXTRFUNC_ERROR` enum to `EXTRFUNC_EVALERROR` in the C header `extrfunc.h`. The old values are deprecated and will be removed in a future release.

Documentation

- The manuals of all solver and several tools are now available in HTML format. A PDF that contains all solver manuals is still available (`docs/solvers/allsolvers.pdf`).

3.31.1.3 Solvers

AlphaECP

- Alpha-ECP v2.10.06.
  - Added support for solver trace file, i.e. new option `solvetrace`.

ANTIGONE, GloMIQO

- Now also available for Mac OS X.

BARON

- New libraries 14.4.0.
  - Added reliability branching for integer variables.
  - Bug fixes in CBC interface and IIS module.
  - Improved performance of problem reading and reformulation.
  - Updated Ipopt to version 3.11.9.

BENCH

- Removed option `cumulative`. This option makes no sense anymore since all solvers are spawned asynchronously and all get the same limits.

CBC

- Fixed a race-condition in log output when using multiple threads.
- New libraries.

Couenne

- New libraries.

CONOPT

- New libraries 3.16D.
  - Tolerance adjustments that make sense if NaN appears in intermediate calculations.

CPLEX

- Don't rerun primal simplex in cases where there is already an infeasibility status.
- New libraries 12.6.1.
- New parameters:
  - `qtolin` controls the linearization of the quadratic terms in the objective function of a QP or MIQP model
  - `localimplied` controls the generation of locally valid implied bound cuts

DICOPT

- Added support for `=N=` rows.

GUROBI

- New libraries 6.0.0.
- Fixed problem in tuning. Previous version did not write out link options in the tuned option file.
- Gurobi 6.0 supports a distributed MIP algorithm (option `DistributedMIPJobs`). This requires the Gurobi Compute Server to be licensed.
- Explicit lazy constraint: Users can use the "dot" option `.lazy` to mark constraints as lazy. Possible values are 0, 1, 2, and 3. See option `.lazy` for details.
- Option `ConcurrentMIPJobs` has been renamed to `ConcurrentJobs`. The distributed concurrent optimizer now also handles LP models.
- Option `ScaleFlag` allows now value 2 which enables aggressive scaling.
- New parameters:
  - `DistributedMIPJobs` controls the number of workers of the distributed MIP algorithm
  - `.lazy` marks constraints as lazy constraints
  - `LazyConstraints` controls the use of lazy constraints

Knitro

- New libraries 9.1.0.
  - Overall speed and robustness improvements on NLP and MINLP models.
  - New algorithm choice introduced: active-set SQP.
  - Optional barrier solution refinement procedure: option `bar_refinement = NO/yes`.
  - Deterministic parallel multi-start: option `ms_deterministic = no/YES`.
- Dropped Knitro for Solaris on Intel CPUs (as announced).

LocalSolver

- Added the hybrid mathematical programming solver `LocalSolver` 5.0 (Beta) from Innovation 24 to the GAMS solvers portfolio.
- LocalSolver can be applied to large-scale, mixed-variable, non-convex optimization problems (GAMS model types MIP, (MI)QCP, (MI/D)NLP). It combines local search techniques, constraint propagation and inference techniques, linear and mixed-integer programming techniques, and nonlinear programming techniques in a unique hybrid neighborhood search approach to find high-quality feasible solutions. Hence, LocalSolver offers an alternative for problems where conventional branch-and-bound and/or outer-approximation based solution methods do not provide satisfactory results.
- It is suggested to set the GAMS options for iteration or timelimit (iterlim, reslim) appropriately to limit the effort that LocalSolver spends on the problem.

Lindo/LindoGlobal

- We will drop Lindo/LindoGlobal libraries for Intel Solaris with the next major release (24.5).
- New libraries 9.0.142 for Linux, Mac OS X, and Windows:
  - Support for semi-continuous variables.
  - Simplex LP algorithm implementation has been improved for speed and robustness. The performance improvements compared to previous version are 90% for primal simplex and 45% for the dual simplex.
  - Knapsack related cuts improvements. Significantly faster solve times on models with certain knapsack-like constraints.
  - Improved default node selection rules improves performance on most MIPs.
  - New branching variable rule options: maximum coefficients and neighborhood branching. Can reduce number of branches on certain MIPs.
  - Perspective reformulation capability gives improved performance on quadratic portfolio models with semi-continuous variables, e.g. min-buy quantities.
  - Improved default settings for NLPs gives 5% average speed improvement.
  - New preprocessing for LP/MIP significantly reduces coefficient density of certain dense matrices.

Mosek

- New libraries 7.1.0.12.
  - Improved performance of the mixed-integer conic optimizer.

MSNLP/OQNLP

- These solvers now use Conopt as their default solver if Conopt is licensed. Otherwise they will use lsgrg as before.

OS

- We plan to drop the OS solver with GAMS 24.5. The capability of convert to write OSiL files will be kept.

SCIP

- New libraries 3.1 #020d055.

SoPlex

- New libraries 2.0 #d67b17b.

Sulum

- New libraries 4.0.665.
  - Improvements in presolve, leading to more reductions.
  - General improvements in obtaining a feasible solution faster, especially with focus on the root node to obtain a good bound. Rewrite and improvements of feasibility pump, objective diving.
  - The simplex algorithm is now exchanging more information with the branch and cut method. Improvement of SINS heuristic (finding a better basis after the node solve).
  - Added a MIP root restart feature based on reductions.
  - Cutting planes generation was both improved and extended.
  - Changed default for option `mipmaxrestarts` from 5 to 1.

XPRESS

- New libraries for XPRESS v7.8: Optimizer 27.01.02.
  - Automatic solution refinement for LP and MIP models.
  - Improved deterministic concurrent LP.

3.31.1.4 Tools

GAMSIDE

- Added option to suppress trailing zeroes in GDX viewer.
- Improved ability to open files from the Windows Shell.
- Cutoff for number of lines to be syntax colored or not.
- When saving a file, the Undo buffer is no longer cleared.

GDXDUMP

- Text quoted with a single quote did not have a separator when writing `SymbolAsSet`.
- Added option `SymbolAsSetDI`.
- Avoid string overflow when quoting text.
- Added more types for sets and equations.

GDXXRW

- Added option `values=All` which is the new default when `CDim=0` or `RDim=0`.

GDXVIEWER

- Added `ACCDB` format as an output option for MSAccess.

GDX2SQLite, Scenred, Scenred2

- These tools are now also available for AIX.
- Moved documentation of Scenred and Scenred2 from docs/solvers to docs/tools.

IDECMDS

- Allow a pattern to be specified to close files.

MPS2GMS

- The `mps2gms` tool now produces proper generic GAMS source for models with quadratic terms.
- Bug fix for MPS files written by Cplex that contain SOS constraints.

MPSGE

- Moved manual to ''docs/tools''.

3.31.1.5 Expert Level APIs

- There are new API files for the "Indexed GDX" (IDX) library in <`GAMS Dir`>\`apifiles`\<`Language`>\`api`:
  - The indexed GDX library can be used to read and write indexed GDX files.
  - Each symbol in such a GDX file must be a parameter.
  - Each parameter must have a domain consisting of a UELs which forms an integer sequence starting at 1.
  - This way the data is provided in a format convenient to store in arrays in the target language.
  - In GAMS such a file can be read using `$LoadIDX` during compilation.
  - In GAMS such a file can be written using `execute_unloadIDX` during execution.
  - The new example <`GAMS Dir`>\`apifiles`\`CSharp`\`xp_CalcInverseIDX` makes use of this API, this example is also used in the APILib model **CSCalcInverse**.

Python

- Fixed a bug regarding lists of strings in Python 3 (e.g. `gdxDataWriteStr()`).

3.31.1.6 Object Oriented APIs

- We changed the handling of GAMS Aliases in the object oriented APIs:
  - If we ask for the number of GAMSSymbols in a GAMSDatabase, the Aliases will be excluded.
  - If we iterate over all GAMSSymbols in a GAMSDatabase, Aliases will be skipped.
  - If we ask explicitly for an Alias in a GAMSDatabase (`GAMSDatabase.GetSet("a")` with `a` being an Alias) we will get a reference to the GAMSSet referenced by the Alias, not the Alias itself.
  - Note: Aliases can appear in a GAMSDatabase only, if it was initialized by a GDX file containing an Alias.
  - The new examples `Alias` demonstrate this new behavior for the different OO API languages.

Python

- Fixed a bug in `GamsDatabase.merge_record` that prevented the function from creating a record if none was found.

3.31.1.7 Model Libraries

GAMS API Library

This is a new collection of GAMS models. It can be accessed in the GAMS IDE at `Model Libraries` -> `GAMS API Library` or through the command line tool `apilib`. The models in this collection can be used as scripts to compile and execute the example applications using the GAMS object oriented APIs as well as the expert level APIs, which can be found in <`GAMS Dir`>`/apifiles`.

- **apiutil.gms : Generates the API Model Library files** (01)
- **testapi.gms : API Quality Assurance Test** (02)
- **Cex1.gms : Test expert level C API to read and write GDX** (03)
- **CSex1.gms : Test expert level C# API to read and write GDX** (04)
- **CPPex1.gms : Test expert level C API to read and write GDX in C++** (05)
- **DCex1.gms : Test expert level Delphi (function) API to read and write GDX** (06)
- **DOex1.gms : Test expert level Delphi (object) API to read and write GDX** (07)
- **DPex1.gms : Test expert level Delphi (pure) API to read and write GDX** (08)
- **Fex1.gms : Test expert level Fortran API to read and write GDX** (09)
- **Jex1.gms : Test expert level Java API to read and write GDX** (10)
- **Pex1.gms : Test Python core API to read and write GDX** (11)
- **VBex1.gms : Test expert level VB.Net API to read and write GDX** (12)

- **Cex2.gms : Test expert level C API to read and write GDX, set options and execute GAMS** (13)
- **CSex2.gms : Test expert level C# API to read and write GDX, set options and execute GAMS** (14)
- **CPPex2.gms : Test expert level C API to read and write GDX, set options, and execute GAMS in C++** (15)
- **DOex2.gms : Test expert level Delphi (object) API to read and write GDX, set options and execute GAMS** (16)
- **Fex2.gms : Test expert level Fortran API to read and write GDX, set options and execute GAMS** (17)
- **Jex2.gms : Test expert level Java API to read and write GDX, set options and execute GAMS** (18)
- **Pex2.gms : Test Python core API to read and write GDX, set options and execute GAMS** (19)
- **VBex2.gms : Test expert level VB.Net API to read and write GDX, set options and execute GAMS** (20)
- **CStrseq.gms : Test object oriented C# API running a sequence of examples based on a transport model** (21)
- **CPPtrseq.gms : Test object oriented C++ API running a sequence of examples based on a transport model** (22)
- **Jtrseq.gms : Test object oriented Java API running a sequence of examples based on a transport model** (23)
- **Ptrseq.gms : Test object oriented Python API running a sequence of examples based on a transport model** (24)
- **VBtrseq.gms : Test object oriented VB.Net API running a sequence of examples based on a transport model** (25)
- **CSBenders.gms : Test object oriented C# API using a (multi-threaded) Benders Decomposition Algorithm** (26)
- **JBenders.gms : Test object oriented Java API using a (multi-threaded) Benders Decomposition Algorithm** (27)
- **PBenders.gms : Test object oriented Python API using a (multi-threaded) Benders Decomposition Algorithm** (28)
- **CSCutstock.gms : Test object oriented C# API using a cutting stock example** (29)
- **JCutstock.gms : Test object oriented Java API using a cutting stock example** (30)
- **PCutstock.gms : Test object oriented Python API using a cutting stock example** (31)
- **CSDomainChecking.gms : Test object oriented C# API for domain checks** (32)
- **JDomainCheck.gms : Test object oriented Java API for domain checks** (33)
- **PDomainChecking.gms : Test object oriented Python API for domain checks** (34)
- **JInterrupt.gms : Test object oriented Java API for interrupting running GAMS jobs** (35)
- **CSTsp.gms : Test object oriented C# API using a Traveling Salesman Problem** (36)
- **JTsp.gms : Test object oriented Java API using a Traveling Salesman Problem** (37)

- **PTsp.gms : Test object oriented Python API using a Traveling Salesman Problem** (38)
- **CSWarehouse.gms : Test object oriented C# API using a warehouse location problem** (39)
- **JWarehouse.gms : Test object oriented Java API using a warehouse location problem** (40)
- **PWarehouse.gms : Test object oriented Python API using a warehouse location problem** (41)
- **CSAlias.gms : Test handling of Aliases in object oriented C# API** (42)
- **JAlias.gms : Test handling of Aliases in object oriented Java API** (43)
- **PAlias.gms : Test handling of Aliases in object oriented Python API** (44)
- **apihtm.gms : Generates HTM apilib library files** (45)
- **CSCalcInverse.gms : Test expert level C# API to read and write indexed GDX** (46)

GAMS Data Library

- Note: Opening an Excel file (.xls) in protected view may work improperly due to Microsoft issue 2745652. Fix: Enable editing of the Excel file and reopen the file.
- **gdxmrw_qp1_starter.gms : Portfolio Analysis with Matlab and GAMS** (91)
- **gdxmrw_qp2_starter.gms : Portfolio Analysis with Matlab and GAMS** (92)
- **gdxmrw_tr1.gms : Transport LP with non-indexed GDX data interface** (93)
- **gdxmrw_tr2.gms : Transport LP with indexed GDX data interface** (94)
- **gdxmrw_qp3.gms : QP solver M-file using GAMS and GDXMRW** (95)
- **gdxxrw_autoopen.gms : Tests that gdxxrw calls the auto_open macro facility** (96)
- **gdxmrw_qp4.gms : Calling GAMS model from Matlab** (97)
- **gdxmrw_intro01_init.gms : Introduction to data transfer between Matlab and GAMS** (98)
- **gdxmrw_intro02_init.gms : Introduction to calling GAMS from Matlab** (99)
- **gdxmrw_irgdx01_init.gms : Reading data from a indexed GDX file with IRGDX** (100)
- **gdxmrw_iwgdx01_init.gms : Writing data into a indexed GDX file with IWGDX** (101)
- **gdxmrw_rgdx01_init.gms : Reading data from a GDX file into a structure with RGDX** (102)
- **gdxmrw_wgdx01_init.gms : Writing structured data into a GDX file with WGDX** (103)
- **gdxmrw_ext01_init.gms : Extended use of GDXMRW** (104)

GAMS Model Library

- Modified models
  - **tsp1** and **tsp4** now use `Singleton Sets`
  - **flowchan** had incorrect boundary conditions
  - `licememo`: give a full solver/model type matrix independent of the actual license
- **partssupply.gms : Parts Supply Problem** (404) (contains ps2_f_s .. ps5_s_mn, which are still available, as submodels)
- **qfilter.gms : Audio filter design using quad-precision MINOS** (405)
- **derivtst.gms : How to test derivatives of functions** (406)
- **carseq.gms : Car Sequencing** (407)
- **pmedian.gms : P-Median problem** (408)
- **sgolfer.gms : Social Golfer Problem** (409)

GAMS Test Library

- Modified Models
  - **trilib01**, **trilib02**, and **trilib03**: Compare numeric gradient and Hessian values in the models. Function `Sine` in the library does not provide derivatives anymore. Lower the tolerances for acceptance due to numerical derivatives of function `Sine`
- **indic04.gms : Test of indicator constraints with explicit labels** (663)
- parlib01.gms : Test extrinsic functions in parcclib (667)
- **convert10.gms : CONVERT test suite - check interval evaluator in Convert** (668)
- **convert11.gms : CONVERT test suite - check interval evaluator in Convert** (669)
- **convert12.gms : CONVERT test suite - check interval evaluator in Convert** (670)
- **lazy01.gms : Test lazy constraints** (671)
- **mps2gms1.gms : Test mps2gms** (672)
- **execmode01.gms : Test execmode behavior** (673)

3.31.2 24.4.2 Minor release (March 15, 2015)

3.31.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Sergey Kuznetsov, Ricardo M. De Lima, and Corey Noone.

3.31.2.2 GAMS System

GAMS

- Fixed potential problem when running on Windows in a workDir containing a `%`.
- Fixed problem with EMP solves with `solvelink=solveLink.asyncGrid%`.
- Fixed problem with GUSS/Grid when using `execute_loadhandle` instead of `handlecollect`.

3.31.2.3 Solvers

ANTIGONE

- New libraries.

CONOPT

- Introduced new boolean option `PreTri2Log` that send message from Conopt's pre-triangular analyzer that go to the listing file also to the GAMS log. The option requires the generation of the model dictionary, so `mymodel.DictFile=1;` has to be added before the `Solve` statement.

CONVERT

- Convert and ConvertD: Fix for writing scalar MCP models that contained fixed variables removed by `holdfixed=1`.
- ConvertD: Add support for external equations in scalar GAMS models.
- ConvertD: Propagate EPS to scalar GAMS models.

Couenne

- New libraries.

Examiner2

- Support added for custom trace files.

GUROBI

- New libraries 6.0.2.

Lindo/LindoGlobal

- New libraries 9.0.157 for Linux, Mac OS X, and Windows.

LocalSolver

- New libraries 5.0 (20150119).

Mosek

- New libraries 7.1.0.24.

SCIP

- New libraries 3.1 #67d713c.
- Fixed overwriting of solvetrace file by final NLP resolving.

SoPlex

- New libraries 2.0 #8381aa4.

3.31.2.4 Tools

GDXMRW

- `gdxInfo`: fix output of domains in symbol declaration
- `gdxInfo`: handle aliases properly

3.31.2.5 Object Oriented APIs

- Fixed overwriting of the default value for the `integer1` option when running with `GAMSModelInstance`, which lead to unexpected solver behavior.
- Fixed potential problem with `GAMSModelInstance` used with BARON.

.NET

- New property `GAMSSymbol.DomainsAsStrings`: Domains of Symbol, each element is a string, if the domain is an alias in GAMS, this call will return the name of the Alias, not the name of the aliased Set.

Java

- Changed naming scheme of GDX output scratch file to sequence number.

Python

- Fixed a bug in the constructor of all subclasses of `_GamsSymbol` that occurred when the explanatory text was omitted.
- New property `_GamsSymbol.domains_as_strings`: Domains of Symbol, each element is a string. If the domain is an alias in GAMS, this call will return the name of the alias, not the name of the aliased set.

3.31.2.6 Expert Level APIs

GMO

- Fixes to `gmoGetRowJacInfoOne` and `gmoGetColJacInfoOne`: In case of an empty row/column, now return -1 in colidx/rowidx if index base is 0.

3.31.2.7 Model Libraries

GAMS Model Library

- **dyncge.gms : A Recursive-Dynamic Standard CGE Model** (410)

3.31.3 24.4.3 Maintenance release (April 02, 2015)

3.31.3.1 Acknowledgments

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

3.31.3.2 Solvers

Examiner2

- Fixed incapability to continue on a trace file that already has trace records in it.

LocalSolver

- Corrected computation of values for free variables that appeared (linearly) in one equation only.

3.31.3.3 Tools

GDXXRW

- In 24.4.2 we quietly introduced a new way to determine the content of a sheet. This resulted in a bug for empty sheets and sheets that have been saved with an active filter. This has been fixed.

3.31.4 24.4.4 Maintenance release (May 12, 2015)

3.31.4.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Thomas Counsell, Michael Ferris, Jean Mercenier, and Loïc Ventre.

3.31.4.2 Solvers

Gurobi

- New libraries 6.0.4.
- The new libraries do not work on Windows XP anymore. As the library for COIN-OR solvers, SCIP, and SoPlex link to the Gurobi library, these solvers also do not work on Windows XP anymore.

Ipopt

- New libraries for Linux and Mac OS X.
  - Fixed an issue in the MA97 interface that lead to convergence problems.

Minos

- Consider the number of nonlinear constraint variables, not just the nonlinear objective variables, when computing the superbasics limit: memory is plentiful and cheap.
- Fixed case of memory corruption that resulted when using an initial point containing very many superbasic variables. If the initial point contains more than 500 superbasics, limit the Hessian dimension to 500 even though the superbasic limit is larger.

Mosek

- New libraries 7.1.0.30.

Xpress

- New libraries Optimizer 27.01.08 (was 27.01.02 before).

3.31.4.3 Tools

GDXDUMP

- Fixed a problem when writing a scalar or scalar variable/equation in format CSV.

3.31.4.4 Object Oriented APIs

- Fixed a potential problem with `GAMSModelinstance` and certain OS culture settings.

3.31.4.5 Model Libraries

GAMS EMP Library

- **transecs**: Fixed formulation as embedded complementarity system and provide equivalent alternative as single-agent equilibrium system.

3.31.5 24.4.5 Maintenance release (May 26, 2015)

3.31.5.1 Solvers

COIN-OR solvers, SCIP, SoPlex

- On Windows 32bit, OsiGurobi was split off into a separate library, so that other COIN-OR solvers (e.g., Bonmin, Cbc, Couenne, Ipopt) and SCIP and SoPlex do not require the Gurobi 6.0.4 library anymore. Thus, for this release, only Gurobi and OsiGurobi do not run on Windows XP anymore (see 24.4.4 notes on Gurobi).

3.31.6 24.4.6 Minor release (June 26, 2015)

3.31.6.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Erwin Kalvelagen, Rich Roberts, and Jens Schulz.

3.31.6.2 Solvers

CONOPT

- New libraries 3.16F.
- New option `LMUSDF` for handling definitional equations.
- New option `RVFILL` for memory allocation when memory is tight.
- Changed a sorting procedure in the preprocessor. Models with some very dense rows and many pre-triangular variables can experience a significant speedup for the preprocessor.

CPLEX

- New libraries 12.6.2.0.
- Option changes:
  - The popular option value -1 for `BarCrossAlg` to turn off the crossover after a run with barrier has been deprecated. The new way to turn off crossover is to set the new option `SolutionType` to 2.
  - The option `SolutionTarget` has been renamed to `OptimalityTarget`.
  - The option `CutsFactor` has a new default (-1).
- New options:
  - `BQPCuts`: Boolean Quadric Polytope cuts for solving nonconvex QP or MIQP to global optimality.
  - `CPUMask`: Switch and mask to bind threads to processors. Binding threads to processors/cores helps to reduce variability in running time when using multiple threads.
  - `SolutionType`: Type of solution (basic or non basic) for an LP or QP. Set this option to 2 to prevent crossover after barrier.

MOSEK

- New libraries 7.1.0.31.

3.31.6.3 Tools

GAMSIDE

- Added `.ref` for reference file to the files open dialog.
- Left arrow in the first character position now moves to the end of the previous line.

GDXXRW

- Restore the old behavior when reading a set with `Values=Strings`; all elements will be included, not only the ones with a string.
  - Option `Values=String` and `All` are now deprecated and results in a warning; replaced with `Dense`.
  - New options `Values=Dense` or `Sparse`.

XLSTalk

- Allow for up to 9 parameters for macro call.

3.32 24.3 Distribution

3.32.1 24.3.1 Major release (July 31, 2014)

3.32.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Michael Bond, Sebastian Bothor, Jared Erickson, Michael Ferris, Josef Kallrath, Bastian Niebel, Matt Preston, and Tom Rutherford.

3.32.1.2 Platforms

- The Linux 32bit and Solaris 32bit distributions have been dropped (as announced).
- Support for Windows XP may be dropped with some future GAMS version. As several of our solver vendors have already dropped support for Windows XP, some components of a GAMS system may cease to work under Windows XP in the future. If you notice that a 64bit GAMS system is not working on your Windows XP 64bit machine, please let us know.

3.32.1.3 GAMS System

GAMS

- Introduced new keyword `Singleton Set`: A `Singleton Set` in GAMS is a special `Set` that has at most one element (zero elements are allowed as well). `Singleton Sets` can be especially useful in assignment statements since they don't need to be controlled by a controlling index nor an indexed operator, e.g.:

```
Set           s         / s1*s3 /;
Singleton Set single(s) / s2    /;
Parameter p(s);
Scalar    x;

p(s) = ord(s);
x = p(single);
```

  NOTE: Assigning membership to `Singleton Sets` is different than to usual sets. Since `Singleton Sets` can never have more than one element, any assignment to a `Singleton Set` first clears or empties the set, so no explicit clear is necessary. This is illustrated in the following example:

```
Set            i     Static Set            / a, b, c /
               ii(i) Dynamic Set           /    b    /;
Singleton Set  si(i) Dynamic Singleton Set /    b    /;

ii('c') = yes;
si('c') = yes;

Display ii, si;
```

  Here is the output from the `display` statement in the listing file:

```
----      8 SET ii  Dynamic Set
b,    c

----      8 SET si  Dynamic Singleton Set
c
```

  More information can be found in the GAMS Users Guide.

  - The behavior of assignments to `Singleton Sets` can be influenced by the new option `strictSingleton` [0/1]: This option affects the behavior of a membership assignment to a `Singleton Set`. With `strictSingleton = 0` GAMS does not complain about an assignment with more than one element on the right hand side but takes the first one. With `strictSingleton = 1` (default), such an assignment raises an error. This option be set via an option statement and via a command line option.
  - Similarly, data statements for `Singleton Sets` can be influenced by the new dollar control options `$onStrictSingleton`/`$offStrictSingleton`: With `$offStrictSingleton` GAMS does not complain about a data statement for a `Singleton Set` that has more than one element but takes the first one. With `$onStrictSingleton` (default), such an assignment raises an error.

- Introduced so-called obfuscated save files, which are save files where symbol and UEL names have been obfuscated. The new options `saveobfuscate` (`so`) and `xsaveobfuscate` (`xso`) can be used to generate uncompressed and compressed, respectively, obfuscated save files. Additionally, a new option `restartNamed` (`rn`) has been added which can be used to bring back the original names when restarting from an obfuscated save file. The intended use is the following.

  - Compile (only) a GAMS model into a named and an obfuscated save file:

```
$call gams trnsport a=c s=0named saveobfuscate=0anon
```

– Move the obfuscated save file to a non-secure machine and execute it there:

```
$echo * Empty > empty.gms
$call gams empty r=0anon s=1anon
```

– Bring the new (still obfuscated) save file with the results back to the safe machine and do a continued compilation with reporting and export. The continued compilation restarts from the obfuscated save file with all the results, etc., but gets a second save file with proper names through option `restartNamed`:

```
$echo execute_unload 'supply', supply.m; > unload.gms
$call gams unload r=1anon restartNamed=0named
```

In this execution, everything is taken from the obfuscated restart file, except for the symbol and UEL names and the listing file title and subtitle, which are taken from the file specified via `restartNamed`.

- Allow to load the universe of labels from a GDX file into a set at run-time:

```
execute_load 'someFile', someSet=*;
```

Note, that only labels known to the GAMS program will be loaded.

- Fixed a bug that caused wrong results when evaluating `prod` (or `smin`/`smax`) over an empty set inside `sum` like in the following example:

```
sets r          / 1    /
     s          / 1, 2 /
     rs(r,s)    / 1.1  /
     subrs(r,s) / 1.2  /;
scalar z1;
z1 = sum(rs, prod(subrs(rs), 2));
display z1;
```

- Fixed a bug in calling error logging routine for extrinsic functions from within solvers.

Function libraries

- The extrinsic function library **cppcclib** has been expanded to include functions for the PDF and CDF of the trivariate normal distribution. Additionally, documentation for **cppcclib** has been added to Appendix J of the GAMS User's Guide and the relevant testlib models `cpplib00` thru `cpplib05` have been added or updated.

3.32.1.4 Solvers

ANTIGONE, Bonmin, Cbc, Couenne, SCIP, Sulum

- Solvetrace files now include the GAMS input name (usually GAMS model name with `.gms` extension stripped) in the header line.

BARON

- New libraries 14.0.2
  - Significant advances in the handling of integer programs. In addition to several classes of integer cutting planes, calls to MIP solvers and hybrid LP/MIP/NLP relaxations for MINLPs have been added.
  - Complete rewrite of the interfaces to LP solvers.
  - Improvements in probing routines.
  - More robust checks for the validity of the solution of LP/NLP subsolvers.
  - Introduced interfaces to COIN-OR/OSI and CBC 2.8.9.
  - Increased numerical robustness for numerically challenging problems.
  - New range reduction techniques.
  - Systematic treatment of infeasible problems. With the `CompIIS` option, which by default is 0, the computation of an Irreducible Inconsistent Set (IIS) can be requested. Five different algorithms are available, with corresponding values of `CompIIS` equal to 1, ..., 5. Algorithm 1 is a fast heuristic, while algorithms 2 through 5 are more time consuming exact algorithms. If an IIS is found, it is reported in the GAMS listing file. BARON does not consider bounds on binary variables to be part of the IIS. For general integers, the option `IISint` can be be used to signal that general integers should be considered as potential members of the IIS, i.e, integrality constraints are questioned.
- Fixed handling of BARON termination status when activating `DeltaTerm` option.

BONMIN

- Dropped MIP capability (use CBC instead).

CONOPT

- New libraries 3.16B.
  - The new option `Lsusdf` was added. If turned on (the default) CONOPT's preprocessor will look for definitional constraints which are constraints of the form `x = f(y)` where the bounds on `x` are wider than the range of the function `f`, given the bounds on `y`. CONOPT will search for and select a maximal set of independent definitional constraints. The dependent variable in a definitional constraints will be selected for the initial basis and CONOPT will try to use the definition to initialize `x`. There are two other options associated with this procedure: If `Lsuqdf` is enabled (default), then only unique definitional constraints are selected. If it is disabled, then option `Lfusdf` specifies a limit (default 2) on the number of candidates a definitional constraint can have.
  - The option `Lsflsh` – flush the buffer, has now been implemented and is automatically turned on if there is a systems error.
  - The lower bound on option `Rtipvr` changed from 1.e-3 to 1.e-10.

Convert

- The AMPL writer supports special ordered sets, semicontinuous variables, and semiinteger variables now.
- Row names in `.lp` and `.mps` files now start with 'e' instead of 'c', so they match the names written by the `dict` option.

CPLEX

- New libraries 12.6.0.1.

CPLEX, SCIP, XPRESS

- Fixed handling of indicator constraints when specified with explicit labels.

DE

- New option `VaRBigM` to control the Big M for a Value at Risk reformulation.

EMPSP

- New keywords `VaRUp` (=`VaR`) and `VaRLo`: These keywords can be used to optimize the Value at Risk for a certain confidence level. The syntax is similar to the one from `cVaRUp` (=`cVaR`) and `cVaRLo`:

  ```
  VaR [rv var] scalar
  ```

  More information can be found here.

- Changed the order of the parameters for the triangular distribution from

  ```
  randvar <name> triangular <low> <high> <mid>
  ```

  to

  ```
  randvar <name> triangular <low> <mid> <high>
  ```

Examiner2

- Fixed a bug in processing special ordered sets.

GUSS/Scenario Solver

- GUSS/Scenario solver can now be combined with the GAMS Grid Facility (see example `GUSSGRID` in the GAMS Model Library).
- New option `RestartType` to determines restart point for the scenarios:
  - 0: Restart from last solution (default)
  - 1: Restart from solution of base case
  - 2: Restart from input point
- New option `SolveEmpty` (default 0) to limit the number of empty scenarios (no scenario data) that are being solved. When the limit is reached, further empty scenarios will be skipped. Skipped scenarios will be reported to the log and listing file.

Ipopt

- New libraries.

Knitro

- As Ziena Optimization has dropped support for Knitro on Solaris (x86) a while ago, we plan to drop GAMS/Knitro on Solaris with the next release.

Lindo/LindoGlobal

- New libraries 8.0.550.
- LindoGlobal is no longer available for Sparc Solaris (as announced).

Mosek

- New libraries 7.0.0.121 (Linux, Mac OS X) and 7.0.0.123 (Windows).

OQNLP

- OQNLP is no longer available for Linux (as announced).

OS

- GAMS to OSiL conversion now creates more compact instances, especially for quadratic equations and long sums or products in general nonlinear expressions.

SCIP

- New libraries `3.1 #695c979`.
  - Added new primal heuristics "random rounding", "proximity", and "dual value", new branching rule "cloud branching", and new node selectors "breadthfirst" and "uct".
  - Added support for strong branching with domain propagation in full strong and reliability pseudo cost branching.
  - Improved numerical stability (now taking the rank of cuts into account; more checks on LP solution; disabled scaling in feasibility check of nonlinear constraints).
  - Many improvements in presolving.
  - Strong branching LP solutions are now checked for feasibility.
  - Changed or removed parameters:
    - branching/relpscost/maxlookahead: default changed from 9 to 8.
    - branching/relpscost/maxreliable: default changed from 8 to 5.
    - constraints/bivariate/scaling: default changed from TRUE to 'o' (type changed to character).
    - constraints/quadratic/scaling: default changed from TRUE to 'o' (type changed to character).

* constraints/soc/scaling: default changed from TRUE to 'o' (type changed to character).
* constraints/varbound/maxlpcoef: default changed from 1E6 to 1E9.
* heuristics/crossover/minnodes: default changed from 500 to 50.
* heuristics/dins/minnodes: default changed from 500 to 50.
* heuristics/feaspump/objfactor: default changed from 1 to 0.1.
* heuristics/rens/minnodes: default changed from 500 to 50.
* heuristics/rins/freq: default changed from -1 to 25.
* heuristics/rins/freqofs: default changed from 5 to 0.
* heuristics/rins/minfixingrate: default changed from 0 to 0.3.
* heuristics/rins/minnodes: default changed from 500 to 50.
* heuristics/shiftandpropagate/sortkey: default changed from 'u' to 'v'.
* lp/checkfeas: replaced by new parameters lp/checkdualfeas and lp/checkprimfeas.
* numerics/dualfeastol: default changed from 1E-6 to 1E-7.
* presolving/dualfix/∗: replaced by propagating/dualfix/∗
* propagating/pseudoobj/presoldelay: default changed from TRUE to FALSE.
* propagating/pseudoobj/timingmask: default changed from 5 to 7.
* propagating/redcost/timingmask: default changed from 2 to 6.
* separating/cgmip/objweighsize: renamed to separating/cgmip/objweightsize and default changed from FALSE to TRUE.
* separating/minefficacyroot: default changed from 0.01 to 0.001.
* separating/closecuts/relintnormtype: removed

SoPlex

- New libraries `2.0.0`.

XPRESS

- New libraries 26.01.08.

3.32.1.5 Tools

GDX2SQLITE

- GDX2SQLITE is a new tool to dump the complete contents of a GAMS GDX file into a SQLite database file.

GDXDUMP

- New option `SymbolsAsSet` to write the symbol table for a set as data.

GDXMRW

- New utilities `irgdx` and `iwgdx` for exchanging indexed GDX data with Matlab.

GDXRENAME

- New utility to rename the same unique elements in a GDX file using a mapping given by a second GDX file.

GDXXRW

- New option to allow the use of R1C1 notation to specify a cell or a range.
- Reading a set using the option `Values=string` now skips empty cells.

3.32.1.6 Expert Level APIs

GMO

- Removed previously deprecated function `gmoDirtyExtractDefVar`.
- Added a number of functions to access information from an EMP info file, to compute the optimality gap, to access extrinsic function libraries, and to get the name of the GAMS input file.

3.32.1.7 Object Oriented APIs

- New example `transport14`.
- New method `GAMSSymbolRecord.Key(int index)` (Java: `GAMSSymbolRecord.getKey`, Python: `_GamsSymbolRecord.key(int index)`) to retrieve the key of `GAMSSymbolRecord` (Python: `_GamsSymbolRecord`) for a given positional index.
- `GAMSSymbol.CopySymbol` now works for the universe of a `GAMSDatabase` (`GAMSDatabase.GetSet("*")`). When copying into the Universe, a merge will be performed.
- Real domains are now registered when exporting a `GAMSDatabase` to GDX (so far, only relaxed domains were registered).
- `GAMSJob.Run` (Python: `GamsJob.run`) now creates `OutDB` (Python: `out_db`) also if it raises a `GAMSExceptionExecution`.
- `GamsModelInstance.Solve` (Java/Python: `GamsModelInstance.solve`) now also works for solvers which require a subsolver, e.g., DICOPT.

.NET

- Fixed default value for `systemDirectory` argument in `GAMSWorkspace` constructor when using MONO: If no value is given, first the `PATH` and then the (`DY`)`LD_LIBRARY_PATH` is checked for a valid GAMS system directory.

Java

- It is no longer necessary to specify `-Djava.library.path` when running a program. If `java.library.path` is specified, the shared libraries will be loaded from `java.library.path`. Otherwise, the shared libraries will be loaded from the class path that contains `GAMSJavaAPI.jar`.
- It is no longer necessary to set up environment variables (`PATH`, (`DY`)`LD_LIBRARY_PATH`) before running a program to find the GAMS system directory, as it can now be specified during run time.
- In the `GAMSWorkspace` default constructor, the default setting for finding the GAMS system directory from environment variables in the following order (depends on the target platform) are applied:
  - Windows: first from `PATH` environment variable. If not found, then from the Windows registry `gams.location`,
  - Mac OS X: first from `PATH` environment variable. If not found, then from `DYLD_LIBRARY_PATH`,
  - other Unix: from `PATH` environment variable. If not found, then from `LD_LIBRARY_PATH`.
- In the non-default `GAMSWorkspace` constructor, the following rules are applied:
  - In case a user specifies a system directory, the API will verify the directory and will not search for GAMS system directory from an environment variable, even when the directory is invalid.
  - In case the specified system directory is `null` or a user does not specify a system directory, the API will apply the default setting from above.
- A memory leak in `GAMSDatabaseIterator` has been closed.
- Issues when using a non-standard locale (`LANG` environment variable) have been fixed.
- All deprecated classes and methods since 24.1 have been removed.

Python

- New behavior on determining a system directory automatically when a workspace is created.
  - Linux: If no system directory is specified in the `GamsWorkspace` constructor, check `PATH` first. If no system directory was found, check `LD_LIBRARY_PATH`.
  - Mac OS X: If no system directory is specified in the `GamsWorkspace` constructor, check `PATH` first. If no system directory was found, check `DYLD_LIBRARY_PATH`.
  - Windows: If no system directory is specified in the `GamsWorkspace` constructor, check the Windows registry.

3.32.1.8 Model Libraries

GAMS Data Library

- `MakeQL`: Moved querylibrary generator into `trisource.zip`, which comes with Test Library models **trilib01**, **trilib02**, **trilib03**, and **trilib04**. Removed `MakeQL` model.
- **invert1.gms : Matrix inversion via R** (89)
- **invert2.gms : Matrix inversion via Matlab** (90)

GAMS Model Library

- **clad.gms : Computation of Fairs extramarital affairs model estimates** (397)
- **gussex1.gms : Simple GUSS example** (398)
- **gussrisk.gms : Simple investment example with varying weight for risk using GUSS** (399)
- **gussgrid.gms : Simple GUSS Grid example** (400)
- **circpack.gms : Pack circles in the smallest possible rectangle** (401)
- **tablelayout.gms : Configuring text layout in table cells to minimize table height** (402)
- **asyncjobs.gms : Execute asynchronously several GAMS jobs and collect the fastest** (403)

GAMS Test Library

- **trilib01**: Reworked tricclib.c source code. Rewrote querylibrary generator.
- **single01.gms : Check handling of singleton sets** (639)
- **execerr1.gms : Test for execerr option** (640)
- **single02.gms : Check assignment to singleton sets** (641)
- **scensol4.gms : Basic GUSS Test** (642)
- **gdxmrw06.gms : run a battery of GDXMRW tests** (643)
- **single03.gms : Check singleton sets in put statement** (644)
- **unload11.gms : Check that GAMS does not crash when writing to non-existing folder** (645)
- **refact00.gms : Check that GAMS produces expected workfile with option sys14=1** (646)
- **refact01.gms : Refactor suite test 1** (647)
- **scenempty.gms : Empty scenario GUSS Test** (648)
- **obfusc01.gms : Test use if obfuscated workfile** (649)
- **load11.gms : Load UEL Table** (650)
- **gussskip.gms : Simple GUSS example with skipped scenario** (651)
- **refact02.gms : Refactor suite test 2** (653)
- **cpplib03.gms : Test extrinsic functions in cppcclib** (654)
- **cpplib04.gms : Test extrinsic functions in cppcclib** (655)
- **cpplib05.gms : Test extrinsic functions in cppcclib** (656)
- **cpplib00.gms : Test build of CPP library** (657)
- **syschk2.gms : Test impact of sys10 setting** (658)
- **scensol5.gms : Test handling of scenario dictionary sets with more than 50 entries** (659)
- **call5.gms : Check that gams works with COMSPEC unset** (660)
- **idxoper1.gms : Test indexed operations** (661)
- **gdxsqlite1.gms : Test basic functionality of GDX2SQLITE tool** (662)

3.32.2 24.3.2 Minor release (August 29, 2014)

3.32.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Alvaro Lorca Galvez, Scott McDonald, Tom Rutherford, Nick Sahinidis, and Uwe Schneider.

3.32.2.2 GAMS System

GAMS

- Allow empty scalar data statements:

```
$onempty
scalar xyz / /;
```

Extrinsic Function libraries

- Pass on parameter file name via `LibInit` call.

3.32.2.3 Solvers

BARON

- New libraries 14.0.3.

CBC

- New libraries.

CONOPT

- New libraries 3.16C.
  - Added checks for internal intermediate results being NaN (Not A Number). If this happens, CONOPT will stop and return a message about it. The solver status will return 4 "Terminated by Solver" and model status 6 or 7, "Intermediate Infeasible" or "Intermediate Feasible."

CONVERT

- Fixed bug in interval evaluations for functions and gradients.
- The interval evaluations are now triggered by their own option (e.g. `intervalEval=yes`) instead of being part of every Jacobian or Hessian dump.

Cplex

- Some Cplex tuning parameters had no or the wrong effect. This has been fixed.
- The BCH usercutcall was not called in case of new constrains only for true cuts (see e.g. GAMS Model library model bchtsp). This has been fixed.

Examiner2

- Fixed error in processing options file.
- Fixed error in trace file generation.

Ipopt

- New libraries.

Mosek

- New libraries 7.0.0.126.
  - Fixed an issue with using the Mosek extended license for MIP solving on Mac OS X.

Osi links

- Fixed writing MPS files with row and column names.

SCIP

- New libraries 3.1 #322574a
  - Changed default of option `heuristics/proximity/minimprove` from 0.25 to 0.02.
  - Changed default of option `heuristics/proximity/usefinallp` from TRUE to FALSE.

XPRESS

- New libraries 26.01.14 for Linux and Windows.

3.32.2.4 Tools

GDXDUMP

- When generating `$GDXIN filename`, the filename now includes the full path of the GDX input file.

GDXXRW

- In the previous release, we changed the behavior that when reading with `Values=String`, the empty cells no longer created an entry. Because reading with this option was the default for the cases with `RDim=0` or `CDim=0`, we changed this default to `Values=NoData` in order not to break existing code.

Shellexecute

- Fixed error causing problems during parameter processing.

3.32.2.5 Expert Level APIs

- Delphi: Distinguish between 32 bit and 64 bit compiler.

3.32.2.6 Object Oriented APIs

- Make more GAMS options available through the GAMSOptions class:
  - GAMSOptions.AppendExpand: Expand file append option
  - GAMSOptions.AppendOut: Output file append option
  - GAMSOptions.DumpOpt: Writes preprocessed input to the file input.dmp
  - GAMSOptions.DumpParms: GAMS parameter logging
  - GAMSOptions.ErrMsg: Placing of compilation error messages
  - GAMSOptions.Expand: Expanded (include) input file name
  - GAMSOptions.FErr: Alternative error message file
  - GAMSOptions.JobTrace: Job trace string to be written to the trace file at the end of a Gams job
  - GAMSOptions.LimCol: Maximum number of columns listed in one variable block
  - GAMSOptions.LimRow: Maximum number of rows listed in one equation block
  - GAMSOptions.LogLine: Amount of line tracing to the log file
  - GAMSOptions.On115: Generate errors for unknown unique element in an equation
  - GAMSOptions.Output: Output file
  - GAMSOptions.PageContr: Output file page control option
  - GAMSOptions.PageSize: Output file page size (=0 no paging)
  - GAMSOptions.PageWidth: Output file page width
  - GAMSOptions.Reference: Symbol reference file
  - GAMSOptions.ScriptExit: Program or script to be executed at the end of a GAMS run
  - GAMSOptions.Suppress: Compiler listing option
  - GAMSOptions.Symbol: Symbol table file
  - GAMSOptions.TraceLevel: Solvestat threshold used in conjunction with a=GT

3.32.2.7 Model Libraries

GAMS Data Library

- Fixed some errors in some Matlab examples.

3.32.3 24.3.3 Minor release (September 19, 2014)

3.32.3.1 Acknowledgments

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

3.32.3.2 Solvers

ANTIGONE, Bonmin, CBC, Couenne, Ipopt, SCIP

- Fixed issue loading MKL libraries on Linux machines with AVX2 instruction set.

3.32.3.3 Tools

ASK, GDXVIEWER, SQL2GMS, XLS2GMS

- Fixed problem with character set used for error messages.

GDXXRW

- Fixed error when reading special values from Excel.

MDB2GMS, SQL2GMS

- Fixed problem when using comma as decimal separator.

3.32.3.4 Object Oriented APIs

Java

- Fixed a location of listing file when creating a job from (full-path) file without giving a job name.

3.33 24.2 Distribution

3.33.1 24.2.1 Major release (December 09, 2013)

3.33.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Fernando Consigli, Eligius Hendrix, Erwin Kalvelagen, Alexander Lieder, Phani Murukutla, Yaser Tohidi, and Alexander Weber.

3.33.1.2 Platforms

- Sparc Solaris builds are now built on a SunOS 5.10 (Solaris 10) machine.
- We plan to drop support for Linux 32bit and Sparc Solaris 32bit with the next major release (24.3). As a consequence, we plan to also drop OQNLP on Linux 64bit.

3.33.1.3 GAMS System

Installer

- The dialog for choosing a license file to be copied starts from the user's desktop now
- Fixed a bug, when trying to copy license file to itself

3.33.1.4 Function libraries

- A new extrinsic function library **cppcclib** was added to the system. This library is coded in C++ and implements PDFs and CDFs for the univariate and bivariate normal distributions. The use of C++ is crucial here, as an expression class together with operator overloading is used to do all the relevant computation. In this way, we can automatically compute exact derivatives (first and second) for the function values being computed. Further information on extrinsic function libraries can be found in the GAMS User's Guide, Appendix J. The testlib models `cpplib01` and `cpplib02` are also relevant.

3.33.1.5 GAMS

- Fix bug that caused a negative value for model attribute `etSolver`
- `EXECUTE_LOADDC`: Enhanced `execute_load` with domain checking. Any domain violation will be reported and flagged as execution error. In contrast, `execute_load` ignores all domain violations and loads only data that meets the domain restrictions.
- `EXECUTE_UNLOADDI`: This new keyword does not only write the defined symbols to GDX but also the domain sets.
- Loading one dimensional sets from multi-dimensional parameters/sets via `load i&lt;gdxsym`. Details can be found in the *gdxutils* document in the tools section of the documentation tree.
- New (command line) option `gdxUELs` [squeezed/full]: If `gdxUELs` is set to `squeezed` (default) only the UELs that are required by the exported symbols are registered to GDX (compared to all UELs with `gdxUELs=full`, which behaves like the former default).
- New (command line) option `DumpParmsLogPrefix` (synonym `DPLP`) prefixes log lines that are triggered by `DumpParms=2` with a specific string for easy filtering of these lines.
- New (command line) option `intVarUp` as a replacement for the command line option `pf4` (which is still usable as a synonym for `intVarUp`)
- The option `sys10`, which changes rpower to ipower when the exponent is constant and within e-12 of an integer, is also available as command line option now
- New option `DmpOpt`: This writes all options, which can be set using the option statement, and their values to the listing file

- GAMS checks now more frequently if a GAMS program exceeds the preset elapsed time limit `EtLim` (default 1e20 seconds). In the past this was done only before solve and execute statements. Now, GAMS checks this limit at the same time GAMS checks for a pending Ctrl-C event.

- Renamed model status `7 Intermediate Nonoptimal` to `7 Feasible Solution`. The string constant `%modelStat.intermediateNonOptimal%` is still recognized by the GAMS compiler, but it is advised to switch to `%modelStat.feasibleSolution%`. The model status test is e.g. printed in the solve summary in the listing file and can be written to put files with the model attribute `mymodel.tModStat`.

- New model attributes:

  - `myModel.marginals`: Indicates the presence (1) or absence (0) of a dual solution from the solver after the solve statement. In some circumstances, a solver does not provide a dual solution (.m), in such a case GAMS will not print the marginal column in the solution listing and set the marginal field in variables and equations to NA.
  - `myModel.defPoint`: If this attribute determines what point is passed to the solver as an input point. By default, the user supplied point (set via `x.l(j)=...` and `e.m(i)=...`) is passed to the solver. In some circumstances (mostly during debugging) it can be useful to pass a standard default input point, i.e. level set to 0 or lower bound in case 0 is not within the bounds and marginal set to 0. `myModel.defPoint` controls this behavior: Values other than 1 or 2 result in default behavior, value 1 results in passing default levels and marginals, value 2 results in passing the levels set by the user and defaults (0) for the marginals.

- Added `%system.dirsep%` and `system.dirsep` (for use in put files) to get access to the OS specific file/directory separator character. On Windows platforms this is \ and on Unix and Mac this is `/`. This helps writing platform independent GAMS models.

- Added option 6 to `[file].TF` (text fill) fills `.te()` with quoted element names separated by spaces

- Allow the `Alias` statement on multidimensional sets as well:

```
Set i       / i0*i9 /
    j       / j0*j9 /
    ij(i,j) / #i.#j /;
Alias (ij, ji);
```

3.33.1.6 Solvers

ANTIGONE, BARON, Couenne, SCIP

- Now return model status 7 (feasible but not necessarily optimal) instead of 2 (locally optimal) when finding feasible solution for a model without discrete variables and not having proven global optimality (local optimality cannot be guaranteed).

ANTIGONE

- Fixed bug where optimality gaps were reported for convex problems.

BARON

- New libraries 12.7.3
- Performance improvements in local search, branching, integer presolve, and relaxations for problems with bilinear and/or integer variables
- Added facilities for complementarity problems
- Absolute and relative constraint feasibility and integrality feasibility tolerances
- Added flexible termination criteria based on progress
- Improved treatment of unbounded/infeasible problems
- Selection of specific NLP solvers to be included in BARON's dynamic NLP solver selection strategy
- Automatic setting of many options based on problem characteristics and learning algorithms. As a result, many options are now deprecated
- New options:
  - `AbsConFeasTol`: Absolute constraint feasibility tolerance
  - `AbsIntFeasTol`: Absolute integer feasibility tolerance
  - `AllowExternal`: Indicator for use of External NLP solver with automatic NLP solver selection
  - `AllowIpopt`: Indicator for use of IPOPT with automatic NLP solver selection
  - `AllowMinos`: Indicator for use of MINOS with automatic NLP solver selection
  - `AllowSnopt`: Indicator for use of SNOPT with automatic NLP solver selection
  - `DeltaA`: Absolute improvement for insufficient progress termination
  - `DeltaR`: Relative improvement for insufficient progress termination
  - `DeltaT`: Time interval for insufficient progress termination
  - `DeltaTerm`: Indicates whether insufficient progress termination is on or off
  - `RelConFeasTol`: Relative constraint feasibility tolerance
  - `RelIntFeasTol`: Relative integer feasibility tolerance
- Options deprecated: `modbrpt`, `convexratio`, `maxredpass`, `maxnodepass`, `redreltol`, `redabstol`, `postreltol`, `postabstol`, `hreltol`,`habstol`, `maxheur`, `pbin`, `twoways`, `pstart`, `pend`, `pfreq`, `profra`, `pxdo`, `maxpretime`, `basfra`, `baskp`, `prelpdo`, `numbranch` and `numstore`.

Bonmin(H), CBC, Couenne, Ipopt(H), OS, OsiCplex, OsiGurobi, OsiMosek, OsiXpress, SoPlex

- The solver manual coin.pdf has been split into one document for each solver(group) (bonmin.pdf, cbc.pdf, couenne.pdf, ipopt.pdf, os.pdf, osi.pdf).

CPLEX

- New libraries 12.6.0.
- Cplex 12.6 solves non-convex (MI)QP problems to global optimality when parameter `SOLUTIONTARGET` is set to 3.
- Cplex 12.6 also offers the capability to solve a single MIP instance distributed over a number of machines. The feature, knows as *Distributed MIP*, is available with GAMS/CplexD only and requires a special license and additional software.

CONOPT

- New libraries 3.15M.
  - Fixed a problem with the definition and execution of 'implied free' post-triangular variables. The tolerance was too loose compared with other tolerances and a projection was missing. That could result in functions or derivatives being called with variables slightly outside their bounds (by up to 4.e-10).

Convert

- Fixed writing of mod operator in AMPL syntax.

ConvertD

- The converter to OSiL format can now handle SOS1 and SOS2 variables.

EMPSP

For the keyword `cvarup` (which is the same as `cvar`), the meaning of the defined `scalar` was changed:

```
cvarup [rv var] scalar
```

`Scalar` represents now the confidence level for the Conditional Value at Risk. It holds `NewScalar = 1 - OldScalar`.

Gurobi

- New libraries 5.6
- New options:
  - `PreSOS1BigM` and `PreSOS2BigM`: These new parameters provide user control of presolve SOS linearization
  - `Disconnected`: MIP models are sometimes made up of multiple, independent sub-models. This parameter controls the strategy used to exploit this structure
- Distributed MIP algorithms (in connection with Gurobi's Compute Server):
  - distributed tuning (option `TuneJobs`)
  - distributed concurrent MIP (option `ConcurrentMIPJobs`)
  - See also option `ServerPool` and `ServerPassword`

Ipopt/IpoptH, Bonmin/BonminH, Couenne

- On Linux and Windows, MKL PARDISO has been enabled as additional linear solver for Ipopt (experimental). To try it out, set the option `linear_solver pardiso`.
- For users with an IpoptH license, OpenMP support has been enabled for the HSL codes MA86 and MA97.
  The number of threads to use for HSL MA86, HSL MA97, and PARDISO is controlled by the GAMS threads option.

KNITRO

- New libraries 9.0.0.
- MPEC models are now supported.

Lindo/LindoGlobal

- New libraries 8.0.483.

Mosek

- New libraries 7.0.0.96.
- Mosek provides two algorithms for solving problems with discrete variables: 1) mixed-integer optimizer (`MSK_OPTIMIZER_MIXED_INT`) and 2) mixed-integer optimizer for conic and linear problems (`MSK_OPTIMIZER_MIXED_INT_CONIC`). Algorithm 1 has been available since the introduction of MOSEK to GAMS, but requires an additional license code. Algorithm 2 has been introduced with version 7 of MOSEK (GAMS 24.1). Benchmarks have shown that both algorithms perform very similar. While we will support algorithm 1 for existing GAMS/MOSEK customers for some time, it will not be possible to purchase this option for new GAMS/MOSEK users. They will have to use algorithm 2.
- Fixed issues with solution obtained from fixed solve when using Mosek's conic MIP solver.
- An experimental interface to Mosek's semidefinite programming (SDP) solver has been added, see the GAMS/Mosek solver documentation for details. Note, that the interface is likely to change in the future.

OsiCplex

- Fixed report of CPLEX Error 1217 at end of MIP solve.

OsiCplex, OsiGurobi

- Fixed setting of LP algorithm via solver specific options file.

OsiMosek

- The MIP solver choice has been changed to Mosek's conic mixed-integer programming solver.

OS

- New libraries 2.8

SoPlex

- New libraries 1.7.2

SCIP

- New libraries 3.0 #6078424
- SCIP setting files can be setup with the options editor of the GAMS IDE now.

SULUM

- New libraries 2.0.432.
- Branching priorities are now supported.

XPRESS

- New libraries 25.01.05.
- Fixed best bound field of start record in MIP trace file.

3.33.1.7 Tools & APIs

Tools

- New GDX tool MCFilter. This tool removes duplicate and dominated points in a multi-criteria solution set.

Expert Level APIs

VBA

- Use camel case instead of all lower case for public functions

VB.NET

- Use camel case instead of all lower case for public functions
- Use dynamic loading of DLLs

Object Oriented APIs

- Do not throw an Exception if an Alias is read by GetSet
- Fix wrong equation subtype definition when reading equation from GAMSDatabase
- Throw an GAMSException if UEL exceeds the maximum size of 63 characters
- Throw an GAMSException if explanatory text exceeds the maximum size of 255 characters

.NET

- Add MONO implementation of the object oriented GAMS .Net API for Mac OS X and Linux.
- New version of function GAMSWorkspace.AddOptions creating an instance of GAMSOptions initialized by an existing option file
  - If we pass a nullptr to this Function from C++, we need to cast this to GAMSObject^ now because of the overloading
- New properties: GAMSWorkspace.APIVersion, GAMSWorkspace.APIMajorRelNumber, GAMSWorkspace.APIMinorRelNumber and GAMSWorkspace.APIGOLDRelNumber
- New property GAMSWorkspace.MyEPS: Reset value to be stored in and read from GAMSDatabase for Epsilon, default is double.Epsilon
- New function GAMSOptions.Export: Write GAMSOptions into a parameter file
- Switching type of property GAMSOptions.NoNewVarEqu from Integer to Enum (ENoNewVarEqu)
- Renaming of GAMSEnum.ModelStat.NonOptimalIntermed to GAMSEnum.ModelStat.Feasible
- Renaming of GAMSOptions.PoolFree4 to GAMSOptions.IntVarUp
- New C# Example GAMSRemoteObject

Java

- New
  - tutorial: (`GAMS_java_Tutorial.pdf`) under `<Path/To/GAMS>/docs/API`. From the GAMSIDE this document can be accessed at Help -> Docs -> API.
  - method in `GAMSOptions`: `export`, to write GAMSOptions into a parameter file
  - methods in `GAMSWorkspace`: `addOptions` to create a GAMSOptions object from either another object or an option file
  - methods in `GAMSWorkspace`: `getAPIVersion`, `getAPIMajorReleaseNumber`, `getAPIMinorReleaseNumber`, and `getAPIGoldReleaseNumber`, to retrieve API version number
  - method in `GAMSWorkspace`: `setMyEPS` to reset `GAMSGlobals.SpecialValues.EPS`, the value to be stored in and read from GAMSDatabase for Epsilon.
- Changed
  - default value of `GAMSGlobals.SpecialValues`: `NAN`, `PLUS_INF`, `MINUS_INF`, and `EPS`
  - deprecated `GAMSGlobals.ModelStat.NONOPTIMAL_INTERMED`, replaced by `GAMSGlobals.ModelStat.FEASIBLE`
  - deprecated `GAMSOptions.PoolFree4`, replaced by `GAMSOptions.IntVarUp`
  - deprecated the type of `GAMSOptions.NoNewVarEqu`, replaced by enum `GAMSOptions.ENoNewVarEq`
- Fixed
  - a bug when running a job with an input directory IDir added into GAMSOptions object.

Python

- Add tutorial (`GAMS_python_Tutorial.pdf`) to `<Path/To/GAMS>/docs/API`. From the GAMSIDE this document can be accessed at Help -> Docs -> API.
- New parameter opt_file in function GamsWorkspace.add_options allows to create an instance of GamsOptions that is initialized by an existing option file.
- New function GamsOptions.export: Write GamsOptions into a parameter file
- New properties: GamsWorkspace.api_version, GamsWorkspace.api_major_rel_number, GamsWorkspace.api_minor_rel_
  and GamsWorkspace.api_gold_rel_number
- New property GamsWorkspace.my_eps: Reset value to be stored in and read from GamsDatabase for Epsilon.
- New class NoNewVarEqu providing static fields to set option nonewvarequ
- Renaming of GamsOptions.PoolFree4 to GamsOptions.IntVarUp and property poolfree4 to intvarup
- Renaming of ModelStat.NonOptimalIntermed to ModelStat.Feasible
- The following functions throw an exception, when a wrong data type is passed for the keys/slice parameter: delete_record, find_record, add_record, merge_record, first_record, last_record. Valid data types are str, list, tuple and their subclasses
- Fixed a bug in _GamsSymbol.add_record() that occurred when using empty strings as keys
- Fixed a bug in GamsModelInstance.solve() that affected the logging behavior

3.33.1.8 Model Libraries

GAMS Data Library

- **GDXMRWShowJac1 (86)**: Visualize model Jacobian in Matlab
- **SpawnGAMSAccess (87)**: SpawnGAMS: Spawn GAMS from Access
- **triobal (88)**: Tommasino-Rao Input Output Balance Software

GAMS EMP Library

- **vidualvar (96)**: VI with dualvar specified for VI constraint
- **vi_equil (97)**: Identical VI models with and without containing equilibrium
- **vi_mcp (98)**: Identical models specified using MCP and VI syntax
- **equil_bilevel (99)**: Equilibrium model with and without containing bilevel

GAMS Model Library

- **gastrans (217)**: Added a simple straightforward NLP formulation of the problem, which is now solved by default.
- **knp (321)**: Removed redundant model equations, now regarding best bounds reported by solvers, improved readability.
- **tricp (395)**: Triangular Graph Circle Packing
- **allbases (396)**: Enumerate All Basic Solutions of an LP

GAMS Test Library

- **load7 (613)**: Tests execute_loaddc
- **unload10 (614)**: Test unload options new with GAMS 24.2
- **mpec04 (615)**: MPEC model testing free rows
- **mpec05 (616)**: Simple MPEC unique solution lower bounded matches
- **mpec06 (617)**: Simple MPEC unique solution upper bounded matches
- **mpec07 (618)**: Simple MPEC unique solution doubly-bounded matches
- **mpec08 (619)**: Simple MPEC unique solution LB matches integer var
- **mpsge13 (620)**: MPSGE test - model returns invalid income level
- **card02 (621)**: Tests miniparser card() function
- **uldidx01 (622)**: UnloadIdx - basic operation
- **uldidx02 (623)**: UnloadIdx - checking execution restrictions
- **uldidx03 (624)**: UnloadIdx - checking compilation restrictions
- **ldidx01 (625)**: $loadIdx - checking basic operation
- **ldidx02 (626)**: $loadIdx - checking restrictions
- **ldidx03 (627)**: $loadIdx - checking restrictions
- **ldidx04 (628)**: $loadIdx - checking restrictions
- **ldidx05 (629)**: $loadIdx - checking restrictions
- **sdp01 (630)**: Test of correct solving a simple conic program
- **load8 (631)**: Domain projection load tests
- **load9 (632)**: Domain projection load tests
- **lindgl03 (633)**: Check that Lindo(Global) Option NLP_QUADCHK works
- **dbg01 (634)**: Test debugging option on Windows
- **cpplib01 (635)**: Test extrinsic functions in cppcclib
- **cpplib02 (636)**: Test extrinsic functions in cppcclib
- **alias01 (637)**: Check handling of multidimensional aliases
- **load10 (638)**: Domain projection load tests

3.33.1.9 Solver/Platform Availability Matrix

3.33.2 24.2.2 Maintenance release (March 04, 2014)

3.33.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Michael Bond, Michael Ferris, Erwin Kalvelagen, Timo Lohmann, Kourosh Marjani Rasmussen, Javier Salmeron, Bassel Timani, Shigeru Tsubakitani, Tom Walker, and Jürgen Wolpert.

3.33.2.2 Platforms

- Including now the correct Fortran and C++ standard libraries in a Solaris 64bit Intel system.

3.33.2.3 GAMS System

- Fix a bug that could cause a crash when unloading an alias to the universe to GDX.
- Fix a bug when unloading to a GDX file failed.
- Accept a Mac license when using the GAMS Windows Distribution via Wine.

3.33.2.4 Solvers

BARON

- New libraries 12.7.7.
- Missing option `results` added.
- Option `PrLevel` accepts only 0 or 1.

Bonmin

- New libraries.

CBC

- New libraries.
- Add solution report for LPs that have not been solved to optimality (with corresponding infeasibility, non-optimality, unboundedness markers).

CONOPT

- New library 3.15N.

Couenne

- New libraries.

Cplex

- Fix a bug that resulted in Cplex solving an LP as a MIP in case of solvelink=5.

Gurobi

- New libraries 5.6.2.
- This Gurobi version introduces an experimental no relaxation heuristic that attempts to find good quality feasible solutions to MIP models without first solving the root relaxation. This can be useful in situations where the root relaxation is extremely large or difficult.

GUSS/Scenario Solver

- Fix a bug with keyword 'opt' when retrieving model attributes from the scenario solves.

KNITRO

- New libraries 9.0.1.

Lindo/LindoGlobal

- We will drop LindoGlobal libraries for Sparc Solaris with the next major release (24.3).
- Fix potential problem using multiple threads on LP problem.
- New libraries 8.0.498.

Mosek

- New libraries 7.0.0.106.

SCIP

- New libraries 3.0 #70041f0.

XPRESS

- New libraries 25.01.07.

3.33.2.5 Tools & APIs

GDX Tools

- Fixed a problem with `gdxrank` that surfaced with the new default `Squeezed` of option `gdxUELs`.
- CSV formatted output of `gdxdump` no longer includes a `Val` column for sets.
- Tools like `gdxdump`, `gdxdiff` and `gdxrank` will process an input file without a file extension if that file exists. If the file does not exist, the file with the `.gdx` file extension will be used.

Object Oriented APIs

- Fix error when exporting GAMSDatabase with Aliases.
- Fixed a bug regarding the subtype of equations and their default records, when the equation was added to a GamsDatabase from the APIs.
- Fix GAMSExitCodes.
- Make sure that GAMSModelInstance.Instantiate() does not solve the model, but only prepares everything required for the following Solve().

Java

- Changed
  - null string is treated as an invalid key for all record operations of `GAMSSymbol`.
- Fixed
  - a bug when creating `GAMSDatabase` from source database.
  - a bug when initializing a variable type in `GAMSVariable`.
  - a bug in `GAMSSymbols`: methods `getVarType()` and `getEquType()`.
  - a bug in `GAMSWorkspaceInfo`: method `getSystemDirectory()`.

Python

- Fixed a bug in GamsSymbol.delete_record().

3.33.3 24.2.3 Maintenance release (May 22, 2014)

3.33.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Michael Bond, Wolfgang Britz, Carla Caldeira, Markus Drouven, Guillaume Erbs, Michael Ferris, Inki Kim, Bruce McCarl, Nazmi Sener.

3.33.3.2 Platforms

- On AIX, GAMS executables could allocate at most 2 GB of memory. This limit has been increased to 200 GB.

3.33.3.3 GAMS System

- Fixed option `sys10`

3.33.3.4 Solvers

CONOPT

- New libraries 3.15P.

CBC

- New libraries.

CPLEX, Gurobi and Xpress

- Fixed issues regarding missing libraries and wrong library dependencies on AIX.

DICOPT

- Fixed option setting `relaxed = 0`.

EMPSP

- Fixed a bug with chance constraints.
- Fixed a bug where a certain order of random variables could cause a crash.

Gurobi

- New libraries 5.6.3.
- The `writeprob` option now writes an additional mipstart file (`.mst.gz`), if a MIP is written and the `mipstart` option has been set.

GUSS/Scenario Solver

- Previously, GUSS only solved scenarios that had some update data. Now, GUSS solves also an empty scenario, i.e., a scenario without any update data. It now depends on the update type which scenario is solved.
- Fixed a bug which caused a crash when the scenario dictionary set has more than 50 entries.

Ipopt

- New library.
  - Fixed a bug that lead to convergence issues on some instances.

Lindo

- Fixed option `NSAMPLE_PER_STAGE`.

Mosek, OsiMosek

- New libraries 7.0.0.114.
- Fixed deadlock on Windows when using Mosek or OsiMosek with a user (i.e., non-GAMS) license.

3.33.3.5 Tools & APIs

- har2gdx implements -H option
- gdxxrw understands RC notation for cells when using option `UseRC`.

Object Oriented APIs

- Fixed memory leak in GAMSDatabase.

Java

- Fixed a bug in GAMSDatabase.getDatabaseDomainViolation.

3.34 24.1 Distribution

3.34.1 24.1.1 Major release (May 30, 2013)

3.34.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Étienne Ayotte-Sauvé, Stephen Frank, Youngdae Kim, Andres Ramos, Steffen Rebennack, and Francisco Trespalacios.

3.34.1.2 GAMS System

Installer

- The GAMS installer for Windows is now digitally signed which allows to verify the authenticity

Documentation

- McCarl guide has been updated

GAMS

- new log output option lo=4 has been added: writes simultaneously to log file and stdout.
- Memory leak for GUSS/ScenarioSolver fixed. This also was a problem for the GAMSModelInstance in the OO-API.
- Command line option `OptFile` does not overwrite `model.optfile` anymore. `OptFile` now behaves as other GAMS options e.g. like `ResLim`. A new command line option `ForceOptFile` overwrites all other methods of setting a solver option.
- New option `NoNewVarEqu` will trigger a compilation error when new equation or variable symbols are introduced. This is useful for testing GAMS run-time environments.
- New option `SymPrefix` that prefixes all user symbols compiled in this run with the string value of this option before saving to a save/restart file. This is useful when merging multiple models together to avoid name clashes.

GAMS IDE

- The text comparison utility menu has been changed and now has a structure similar to the GDXDIFF menu; previous file selections are available in the drop-down fields
- The help menu has a link to the search page on the GAMS website

3.34.1.3 Solvers

"D" Solvers

- The "D" solvers GAMS/ConvertD and GAMS/CplexD started as research and development versions of the production solvers GAMS/Convert and GAMS/Cplex and offer some interesting new features that eventually will migrate into the production version.

ALPHAECP

- Alpha-ECP v2.10.02. New option *ECPmaster* and minor performance improvements.

ANTIGONE

- The new solver ANTIGONE (Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations) is a computational framework for deterministic global optimization of nonconvex MINLP. ANTIGONE performs equivalently to GloMIQO when all nonlinearities in MINLP are quadratic.
- ANTIGONE has been developed by Computer-Aided Systems Laboratory at Princeton University; it was completed in collaboration with the Centre for Process Systems Engineering at Imperial College.
- GAMS/ANTIGONE is available for the 32-bit and 64-bit versions of Windows and Linux.
- GAMS/ANTIGONE requires the presence of a GAMS/CPLEX license and either a GAMS/CONOPT or a GAMS/SNOPT license.

Bonmin, Couenne, Osi∗, SCIP

- Marginals are now reported as 'NA', if not available, instead of 0.

Baron

- New libraries 12.3
- New relaxations for certain types of quadratic problems
- Clp 1.14.8
- Farkas infeasibility test to validate LP subsolver claims of infeasibility (Cplex and Clp only)
- Dynamic memory allocation, options workspace and workfactor have been removed
- Improved integer presolve
- Incorporation of convex envelopes for certain low-dimensional functions

CBC

- New libraries Clp 1.15 and Cbc 2.8
- An implementation of zero-half cuts is now available. So far, they may help only on a small subset of problems and may need some tuning, so they are off by default. The new option zerohalfcuts can be used to enable them.
- Alternative implementations of Gomory mixed-integer and reduce and split cuts are now available. By default, these cuts are off. The new options gomorycuts2 and reduceandsplitcuts2 can be used to enable them.
- Added new parameter cut_passes_slow to encourage the use of some of the more exotic/expensive cut generators.
- The random seeds for CLP and CBC can now be set by user via the new options randomseedclp and randomseedcbc.
- Cbc can now solve the root node multiple times, each with its own different seed. This can be enabled via the new option multiplerootpasses.
- New option extravariables that switches on a trivial re-formulation that introduces extra integer variables to group together variables with same cost.
- New option cutoffconstraint to add the objective function as a constraint which right hand side is set to the current cutoff value to the problem.
- New option parallelmode to switch between deterministic and opportunistic parallelization. NOTE: The default is deterministic, while it was opportunistic in the past.

Cplex

- New libraries 12.5.0.1 (fix pack 1)
- New link option `solnpoolmerge` to write all solutions into a single GDX file
- Solutions from the solution pool are written in sorted order from best to worst
- GAMS/CplexD supports the Cplex remote object. See option `ComputeServer`.

CONOPT

- New libraries 3.15K
- Fixed a system error (303) related to inversion of very large matrices.
- Adjusted some test output to handle more than 1 mill rows and columns.
- If the directional 2nd-derivative routines returned an error and should be turned off they were not turned off after all and the result was poor performance and many strange messages. Problem Fixed.
- The final model could be infeasible and still declared feasible if the initial and final scaling factors were very different. Problem Fixed.

DICOPT

- added parameter 'infbnd' to control which value to use for missing bounds in integer cuts (default is 10000)

EMPSP

New experimental keywords making use of the extrinsic function library `lsadclib` to sample random variables with parametric distributions, any feedback will be appreciated:

- `sample <rv1> [rv2 ... rvn] <sampleSize> [varRedGroup]`
  - `sample`: keyword
  - `rv?`: name of random variable
  - `sampleSize`: size of sample
  - `varRed`: variance reduction group, mapped to variance reduction method using solver option file
  - Example:
    - %emp.info%:
      - `sample d 5 group1`
  - lindo.opt
    - `SVR_LS_ANTITHETIC=group1`
- `setSeed <seed>`
  - Sets the seed for `sample`

GloMIQO

- New libraries 2.2
- Fixed bug in detecting when aBB cuts should be generated.

Gurobi

- New libraries 5.5
- New options
  - GAMS/Gurobi support Gurobi's Compute Server. See options `ComputeServer` and options starting with `CS`
  - Parameter Tuning Tool: See option `Tuning` and options starting with `Tune`
  - `ConcurrentMIP`: This new feature launches multiple, independent solves on the same MIP model, using different settings for each. The solve returns when the first one finishes. This approach allows you to exploit multiple cores to explore a diverse set of search strategies
  - `NumericFocus`: This new parameter allows you to indicate that a model is likely to experience numerical trouble, which then causes our internal algorithms to favor numerical robustness over speed

Ipopt, IpoptH

- New libraries 3.11
- default for option ma57_automatic_scaling changed to no
- new options print_frequency_iter and print_frequency_time to adjust amount of iteration summary output
- fixed assignment of INFES markers in listing file (infeasibilities between constraint violation tolerance and acceptable constraint tolerance were not marked)

IpoptH

- Added a new linear system solver HSL_MA97
- Updated linear system solver HSL_MA86
- Both MA57 and HSL_MA97 should be considered as alternative to the default MA27 (see Ipopt option linear_solver)

JAMS

- fixed a bug in substituting variables in a nonlinear equation during a convex-hull reformulation

KNITRO

- new libraries 8.1.1

Lindo, LindoGlobal

- New libraries Lindo 8.0
- Multithreading support
  - The new GAMS/Lindo/LindoGlobal option `num_threads` defines the number of threads to be used, it is initialized by the GAMS option `threads`
  - The new GAMS/Lindo/LindoGlobal option `multithread_mode` defines the threading mode (auto, concurrent or parallel)
- Stochastic Solver Improvements
  - Nested Benders Decomposition implementation has been improved significantly, achieving speed factors up to 6X compared to the previous version
  - The Chance-programming solver adds a Genetic Algorithm to find high-quality feasible solutions to large-scale instances. Models in this class can now also be solved using the Simple Benders Decomposition method
  - Multithreading with Nested Benders Decomposition (NBD) solver leads to speed improvements from 2.5 to 3.5 when using 4 threads
- MIP Solver Improvements
  - The heuristics are improved significantly, simple rounding and feasibility pump now use bound propagation to improve the current path to a new feasible MIP solution
  - Multithreading can lead to speed improvements from 1.5 to 3.0 times on difficult problems using 4 threads rather than 1, for easy MIP problems, e.g., < 600 seconds, multi-threading may give not much speedup
- Multistart Solver Improvements
  - Multistart solver has been improved significantly, achieving speed factors up to 2X compared to the previous version
  - The likelihood of getting the global optimum has improved by 10-15% over a wide range of nonconvex models
  - Multithreading often leads to speed improvements from 2.0 to 3.0 times when using 4 threads, speed improvements tend to improve as the model size and the number of multistarts increase
- Changed meaning of `objwgt` in chance constraints (CC): now it gets multiplied by the violation ratio of the CC before it was added to the objective, in previous versons it was multiplied by (1 - the violation ratio)
- Further new options
  - `find_block`: graph partitioning method to find block structures
  - `gop_linearz`: flag indicating if GOP exploits linearizable model

Mosek

- Mosek 7.0.0.65
- An additional parallelized mixed integer optimizer has been added. This optimizer is chosen for MIP solves if no Mosek/MIP license is available. See also option MSK_IPAR_OPTIMIZER.
- Recognizes some quadratic constraints as cone constraints, so `=c=` equations can be avoided.
- AVX instructions in the latest INTEL CPUs are now exploited.
- Dropped support for Solaris Intel 64bit platform.

- The option writembt is now deprecated. Please consider using the option MSK_SPAR_DATA_FILE_NAME instead.
- The option mipstart is now a synonym for MSK_IPAR_MIO_CONSTRUCT_SOL and thus takes values MSK_ON and MSK_OFF.
- Numeric values for options with symbolic constants (e.g. MSK_IPAR_INTPNT_BASIS) are not supported anymore.

MSNLP, OQNLP

- fixed function evaluation issues in lsgrg when using external equations

OsiMosek

- dropped OsiMosek on Solaris Intel 64bit platform

OSL/OSLSE

- dropped OSL and OSLSE from this and future GAMS distributions

SCIP

- New libraries 3.0 #a4a627b

SULUM

- New libraries 2.0.263 (Beta)
- The new SULUM libraries comes with support for mixed-integer linear programs (MIP). The new MIP optimizer is an advanced implementation of a branch and cut method, with many performance enhancements added. The key features of the MIP optimizer can be highlighted as :
  - Advanced MIP presolve to reduce the problem size and provide a better formulation for the optimizer.
  - Tight integration with the Sulum LP optimizer to efficiently solve LP's in node and during heuristics.
  - Various branching and node selection methods from computational inexpensive to more expensive schemes.
  - Cutting plane generation and filtering if deemed necessary.
  - Heuristics to either find an initial solution or improve the current incumbent, which includes rounding, diving and sub-mipping heuristics types.
  - Improvements to ensure numerical stability.

XPRESS

- New libraries 24.01.04
  - Improved concurrent LP solve
  - Improved performance when using the parallel barrier solver
  - Fix faulty dispatch of SSE3 instructions in QP
- New link option `solnpoolmerge` to write all solutions into a single GDX file

3.34.1.4 Tools & APIs

Object Oriented APIs

- New features were added to the object oriented GAMS APIs including e.g. the capability to specify the domains of symbols, check for domain violations, copying ModelInstances, or setting the debug level using a environment variable. More details about new and modified functions can be found in the following sections about the different languages.

.NET

- Documentation
  - Add tutorial (`GAMS.net4.Tutorial.chm`) to `<Path/To/GAMS>/docs/API`. From the GAMSIDE this document can be accessed at Help → Docs → API.
- GAMSDatabase
  - New function CheckDomains: Check for all symbols if all records are within the specified domain of the symbol
  - New function GetDatabaseDVs: Return all GAMSDatabaseDomainViolations
  - New property SuppressAutoDomainChecking: Controls whether domain checking is called in GAMSDatabase export
  - New variants of the functions AddEquation, AddParameter, AddSet, AddVariable: Allow to specify domain information
  - Function Compact becomes obsolete and will be dropped in future
- GAMSModelinstance
  - New function CopyModelInstance: Copies a ModelInstance to a new ModelInstance which gets constructed at this call
  - New function Interrupt: Sends interrupt signal to running GAMSModelInstance
- GAMSModifier
  - Allow to define SymbolUpdateType for each GAMSModifier separately
- GAMSSymbol (GAMSEquation, GAMSParameter, GAMSSet, GAMSVariable):
  - New function CheckDomains: Check if all records are within the specified domain of the symbol
  - New function CopyToArray: Copies values of a dense symbol into a dense array
  - New function CopyToSqzdArray: Copies values of a sparse symbol into a squeezed array
  - New function CopySparseToDenseArray: Copies values of a sparse symbol into a dense array
  - New function CopyFromDenseArray: Copies values from dense array into a symbol
  - New function CopySliceFromDenseArray: Copies values from slice of dense array into a symbol
  - New function GetSymbolDVs: Return all GAMSSymbolDomainViolations
  - New function MergeRecord: Finds record in GAMSSymbol if it exists, adds it if not
  - New property Domains: Domains of Symbol, each element is either a GAMSSet (real domain) or a string (relaxed domain)
- GAMSWorkspace
  - Change the Debug parameter from a Boolean flag to an enum type called DebugLevel
  - The Debug parameter can be overwritten by the system environment variable "GAMSOOAPIDEBUG" if set to one of the following: Off, KeepFiles, ShowLog, Verbose

  - Improve performance significantly for function AddDatabase(GAMSDatabase)
  - New functions: AddJobFromGamsLib, AddJobFromTestLib, AddJobFromEmpLib, AddJobFromDataLib and AddJobFromFinLib
  - New properties: Version, MajorRelNumber, MinorRelNumber and GOLDRelNumber
  - New optional parameter inModelName for functions AddDatabaseFromGDX/AddDatabase: GAMS string constant that is used to access this database
- Add new sub class of GAMSException: GAMSExceptionExecution. This provides additional info about the reason of the failed execution.
- New examples:
  - C#: DomainChecking, Markowitz, SimpleCutstock, Transport13, TSP, MessageReceiverWindow
  - C++: Transport Sequence
  - VB.NET: Transport Sequence

Java

- Changes in GAMSDatabase:
  - deprecates the compact method, as it has no effect anymore.
- Changes in GAMSGlobals:
  - the default value of working directory has been changed from System.getProperty("user.dir") to System.getProperty("java.io.tmpdir").
- Changes in GAMSModelInstance:
  - deprecates the instantiate(GAMSOptions options), instantiate(GAMSModifier[]), and instantiate(GAMSOptions, GAMSModifier[]) methods and replaced by instantiate(String, GAMSModifier ...) and instantiate(String, GAMSOptions, GAMSModifier ...) methods.
- Changes in GAMSWorkspace and GAMSWorkspaceInfo:
  - deprecates boolean debug flag and replaced by a debug level flag (type of a new class GAMSGlobals.DebugLevel).
  - allows an override of debug level flag from an environment variable "GAMSOOAPIDEBUG"
- Fixed a bug when iterating through the records of a GAMSSymbol.
- New enumeration class GAMSGlobals.DebugLevel:
  - defines values of different GAMS Debug Levels.
- New enumerated value of GAMSModelInstance.SymbolUpdateType:
  - GAMSModelInstance.SymbolUpdateType.INHERIT: to specify SymbolUpdateType separately for each GAMSModifier.
- New methods in GAMSDatabase:
  - addEquation, addParameter, addSet, and addVariable: to add symbols with domain information.
  - checkDomains: to check whether or not all records of all symbols are within the specified domain of the symbols.
  - getDatabaseDomainViolations: to retrieve a domain violation information as a list of GAMSDatabaseDomainViolation objects.

  - isAutoDomainCheckingSuppressed and suppressAutoDomainChecking: to control whether domain checking will be called when exporting a database.

- Changes in GAMSDatabase:
  - the compact method is obsolete and has no effect anymore. It will be removed in the future.
- New class GAMSDatabaseDomainViolation:
  - contains domain violation information of all symbols (if any) in the database.
  - returns call from a new method GAMSDatabase.getDatabaseDomainViolations.
- New methods in GAMSModelInstance:
  - copyModelInstance: to copy a GAMSModelInstance object.
  - interrupt: to send an interrupt signal to a running GAMSModelInstance.
- new methods in GAMSModifier:
  - constructor: to specify SymbolUpdateType for each GAMSModifier object.
  - getUpdateType: to retrieve SymbolUpdateType property of the object.
- New methods in GAMSSymbol:
  - checkDomains: to check whether or not all records of the symbol are within the specified domain.
  - getDomains: to retrieve a list of domains of the symbol, each element is either a GAMSSet (real domain) or a String (relaxed domain).
  - getSymbolDomainViolations: to retrieve a domain violation information as a list of GAMSSymbolDomainViolation objects.
  - mergeRecord: to add a new symbol record in case the record does not exist.
- Fixed a bug when iterating through the records of a GAMSSymbol.
- New class GAMSSymbolDomainViolation:
  - contains domain violation information of the symbol (if any).
  - returns call from a new method GAMSSymbol.getSymbolDomainViolations.
- New methods in GAMSWorkspace:
  - getGAMSVersion: to retrieve information about GAMS Version.
  - getGoldReleaseNumber: to retrieve GAMS GOLD Release Number.
  - getMajorReleaseNumber: to retrieve GAMS Major Release Number.
  - getMinorReleaseNumber: to retrieve GAMS Minor Release Number.
- Changes of location of examples:
  - from <Path/To/GAMS>/apifiles/Java/Benders∗.java to <Path/To/GAMS>/apifiles/Java/benders/Benders∗.ja
  - from <Path/To/GAMS>/apifiles/Java/Custock.java to <Path/To/GAMS>/apifiles/Java/cutstock/Cutstock.ja
  - from <Path/To/GAMS>/apifiles/Java/ConsoleInterrupt.java to <Path/To/GAMS>/apifiles/Java/interrupt/Co
  - from <Path/To/GAMS>/apifiles/Java/Transport∗.java to <Path/To/GAMS>/apifiles/Java/transport/Transpo
  - from <Path/To/GAMS>/apifiles/Java/Warehouse.java to <Path/To/GAMS>/apifiles/Java/warehouse/Wareh
- New examples:
  - <Path/To/GAMS>/apifiles/Java/cutstock/SimpleCutstock.java
  - <Path/To/GAMS>/apifiles/Java/domain/DomainCheck.java
  - <Path/To/GAMS>/apifiles/Java/transport/Transport13.java
  - <Path/To/GAMS>/apifiles/Java/tsp/Tsp.java

Python

- New examples: markowitz.py, tsp.py, transport13.py, simple_cutstock.py, domain_checking.py
- New facility for GamsSymbols with domain information:
  - GamsDatabase:
    - New functions add_parameter_dc, add_variable_dc, add_equation_dc and add_set_dc: Create GamsSymbols with domain information
    - New function check_domains: Check for all symbols if all records are within the specified domain of the symbol
    - New function get_database_dvs: Return all GamsDatabaseDomainViolations
    - New property suppress_auto_domain_checking: Controls whether domain checking is called in GamsDatabase export
  - _GamsSymbol and its derived classes:
    - New function check_domains: Check if all records are within the specified domain of the symbol
    - New function get_symbol_dvs: Return all GamsDatabaseDomainViolations
    - New property domains: Domains of Symbol, each element is either a GamsSet (real domain) or a string (relaxed domain)
  - New classes GamsDatabaseDomainViolation and GamsSymbolDomainViolation that are returned by GamsDatabase.get_database_dvs and _GamsSymbol.get_symbol_dvs
- New functions in GamsModelInstance:
  - copy_modelinstance: Copies a GamsModelInstance to a new GamsModelInstance which gets constructed at this call
  - interrupt: Sends interrupt signal to running GamsModelInstance
- New function in _GamsSymbol and its derived classes: merge_record finds a record if it exists and adds it if not
- New functions in GamsWorkspace: add_job_from_gamslib, add_job_from_testlib, add_job_from_emplib, add_job_from_datalib and add_job_from_finlib
- New properties in GamsWorkspace: version, major_rel_number, minor_rel_number, gold_rel_number
- New optional parameter in_model_name for functions add_database_from_gdx/add_database in GamsWorkspace: GAMS string constant that is used to access this database
- Changed the debug argument passed to the GamsWorkspace constructor. Use members of class DebugLevel: Off, KeepFiles, ShowLog and Verbose instead of True and False
- New sub class of GamsException: GamsExceptionExecution, which provides additional info about the reason of the failed execution
- Allow to define SymbolUpdateType for each GamsModifier seperately
- Significant performance improvement for function GamsWorkspace.add_database when creating from an already existing database
- Changed the unicode settings on Linux from UCS2 to UCS4
- The Debug parameter of GamsWorkspace can be overwritten by the system environment variable "GAMSOOAPIDEBUG" if set to one of the following: Off, KeepFiles, ShowLog, Verbose
- The compact method of GamsDatabase is obsolete and has no effect anymore. It will be removed in the future
- Fixed a bug in GamsModelInstance.solve()
- Fixed a bug in GamsDatabase, where the special value for undefined was set to 0 instead of 1.0E300 (SV_UNDEF)
- Fixed a bug when iterating through the records of a GamsSymbol

GDXXRW

- Some format of range names are now recognized
- More details in error message for a bad range

GDXDUMP

- Option to replace header
- Option EpsOut
- Performance improvement when writing a large CSV file

CSV2GDX

- New utility to convert a CSV file to a GDX file
  - A simple example is in the GAMS data library

MessageReceiverWindow

- New Windows tool that receives messages from GAMS. The GAMS Test Library model **mrw01** demonstrates its usage.

3.34.1.5 Model Libraries

GAMS Data Library

- **SpawnGAMSExcel (84)**: Spawn an arbitrary model from Excel
- **csv2gdx1 (85)**: testing CSV file conversions

GAMS Model Library

- **cpack (387)**: Packing identical size circles in the unit circle
- **trigx (388)**: Another Trigonometric Example
- **stablem (389)**: Stable Marriage Problem
- **srtree (390)**: Simple Scenario Tree Construction Example
- **tgridmix (391)**: Grid Transportation Problem with Single Submit and Collect Loop
- **prisoner (392)**: Prisoners dilemma as EMP and MCP
- **cesam2 (393)**: Cross Entropy SAM Estimation
- **solmpool (394)**: Cplex Solution Pool for a Simple Facility Location Problem with Merged GDX Solution File

GAMS Test Library

- **nlcode7 (586)**: Test for NL code bug from Dist 24.0.1
- **n3707 (587)**: MIP model used by CTRLC test
- **mod011 (588)**: MIP model used by CTRLC test
- **fuzzy (589)**: MINLP model used by CTRLC test
- **nuclear49b (590)**: MINLP model used by CTRLC test
- **enpro56 (591)**: MINLP model used by CTRLC test
- **popdynmMCP25 (592)**: MCP model used by CTRLC test
- **popdynmMCP250 (593)**: MCP model used by CTRLC test
- **popdynmMCP1000 (594)**: MCP model used by CTRLC test
- **gft (595)**: MPSGE model used by CTRLC test
- **pf4mip (596)**: Test unbounded integer variables (MIP)
- **pf4minlp (597)**: Test unbounded integer variables (MINLP)
- **emp12 (598)**: Test of EMP equilibrium models and fixed vars
- **emp13 (599)**: Test of EMP equilibrium models and fixed vars
- **emp14 (600)**: Test of EMP equilibrium model with vi func
- **emp15 (601)**: Test of EMP equilibrium model with vi func
- **emp16 (602)**: Test of NLP -> MCP via JAMS
- **miqcp03 (603)**: Test modsolstat & solution correctness - multiple QCons & binaries
- **mcp10 (604)**: MCP model with negative equ.var
- **convert8 (605)**: Test that eps in nonlinear code is kept by convert
- **gurobi01 (606)**: GUROBI test suite - tuning test
- **lp15 (607)**: LP with many zeros at solution
- **convert9 (608)**: CONVERT test suite - handling of fixed vars for nlp2mcp
- **lsalib01 (609)**: Test extrinsic functions in lsadclib
- **mrw01 (610)**: Test MessageReceiverWindow.exe
- **rs01 (611)**: Solving a Transportation Problem using Cplex and Gurobi remote server
- **xpress06 (612)**: Solution enumerator example with solnPoolMerge

3.34.1.6 Solver/Platform Availability Matrix

3.34.2 24.1.2 Maintenance release (June 16, 2013)

3.34.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Sergio Corvalan, George Mavrotas, Renger van Nieuwkoop, and Andres Ramos.

3.34.2.2 GAMS System

GAMS

- Fixed a problem when restarting from restart files from 24.0 and previous if extrinsic functions were present

3.34.2.3 Solvers

BARON

- New libs 12.3.3
  - SNOPT 7.2-12.1 with fixes
  - New defaults for options `PreLPDo` (0) and `PDo` (-2) which lets BARON decide about probing
- Fixed an issue with reporting the correct solve time back to GAMS

Cplex/CplexD

- new libs 12.5.1
- new parameters `LiftProjCuts` and `CalcQCPDuals`
- Fixed a problem in CplexD where models with all discrete variables relaxed (prior=inf) were still solved as discrete problems

MOSEK

- New libraries 7.0.0.70

SULUM

- New libraries 2.0.284 (Beta)

3.34.2.4 Tools & APIs

GDXXRW

- Fixed a bug where a range like B5:B5 became a single anchor point

3.34.3 24.1.3 Maintenance release (July 26, 2013)

3.34.3.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Sebastian Dilly, Sascha Herrmann, Aida Khajavirad, and Johan Villaume.

3.34.3.2 GAMS System

Installer

- The GAMS installer for Windows now asks if an existing GAMSDIR environment variable from a previous installation should be updated.

GAMS

- Fix crash with huge dictionary files

GAMS IDE

- Fixed formatting of Ipopt, Bonmin, and Couenne options files when written by options editor.
- Using Help | About now uses the project directory for temporary files avoiding issues with a write protected system directory.

3.34.3.3 Solvers

ANTIGONE and GloMIQO

- New libraries ANTIGONE 1.1 and GloMIQO 2.3
  - eliminate constraint on the number of variables (ANTIGONE/GloMIQO was previously limited to 45,000 variables)
  - improve nonlinear term bound inferences
  - improve automatic scaling of problems with many disparate scales
  - track numerically sensitive variables for possible instability associated with optimality-based bounds tightening

COIN-OR / SCIP / SoPlex

- fixed an issue loading coincclib64.dll on some Windows 64bit machines

CONOPT

- New Libraries 3.15L

IPOPTH

- New HSL libraries MA57 3.8, HSL_MC68 3.3, HSL_MA86 1.5, HSL_MA97 2.2
- Changed default of option ma27_meminc_factor from 10 to 2

MOSEK

- New library 7.0.0.75

SCIP

- New library 3.0 #0b46aef

SULUM

- New library 2.0.314

3.34.3.4 Tools & APIs

Object Oriented APIs

Java

Improve data iterator:

- New classes:
  - `GAMSDatabaseIterator` implements `java.util.Iterator`
  - `GAMSSymbolIterator` implements `java.util.Iterator`
- Deprecated class:
  - `GAMSSymbolIterable`
- Changes in `GAMSDatabase`:
  - `GAMSDatabase` implements `GAMSDatabaseIterator` instead of `GAMSSymbolIterable`
  - deprecated the implemented methods of `GAMSSymbolIterable`: `next()`, `hasnext()`, and `remove()`
- Changes in `GAMSSymbol`:
  - `GAMSSymbol` implements `GAMSSymbolIterator` instead of `GAMSSymbolIterable`
  - deprecated the implemented methods of `GAMSSymbolIterable`: `next()`, `hasnext()`, and `remove()`
- new methods in `GAMSSymbolRecord`:
  - `moveNext()`: to iterate to the next record using the current data iterator criterion
  - `movePrevious()`: to iterate to the previous record using the current data iterator criterion

Updated example:

- `<Path/To/GAMS>/apifiles/Java/transport/Transport12.java`

GDXXRW

- Better error reporting for certain range errors
- Issue a warning when writing to an XLSX file by default when there is a file with the same name that has the extension .XLS

3.35 24.0 Distribution

3.35.1 24.0.1 Major release (December 24, 2012)

3.35.1.1 Acknowledgments

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

3.35.1.2 Platforms

- Dropped Mac OS X 32bit build
- Mac OS X 64bit build now on Lion (10.7)

3.35.1.3 GAMS System

APIs

Documentation

- Moved existing documentation about the APIs to <Path/To/GAMS>/docs/API. Also added new documentation there. In the GAMSIDE this documentation can be accessed at Help → Docs → API.

.NET

- Add function Interrupt to GAMSJob: Sends interrupt signal to running GAMSJob

Java

This release contains a beta version of the object-oriented Java API that can be used to control GAMS from a Java program. It allows the seamless integration of GAMS into Java by providing appropriate classes for the interaction with GAMS. GAMS Java API objects allow a convenient way to exchange input data and model results with in-memory representation of data (GAMSDatabase), and to create and run GAMS models (GAMSJob) that can be customized by GAMS options (GAMSOptions). Furthermore, they introduce a way to solve a sequence of closely related model instances in the more efficient way (GAMSModelInstance).

- A Java program that uses object-oriented Java API requires at least Java SE 5 to compile and run.
- All classes are distributed within one single jar file GAMSJavaAPI.jar with a namespace com.gams.api, located under the <Path/To/GAMS>/apifiles/Java/api/ directory.
- Java program examples are distributed with a namespace com.gams.examples, located under the <Path/To/GAMS>/apifiles/Java/ directory.
- Installation and detailed documents can be found in <Path/To/GAMS>/apifiles/readme.txt and <Path/To/GAMS>/docs/API/GAMS_java.pdf.
- Javadoc for GAMSJavaAPI.jar can be found under the <Path/To/GAMS>/apifiles/java/api/javadoc directory.

Python

- In the API to the option object the signature of the function `optGetStrStr` has changed from: `value = optGetStrStr(optHandle, "GDX", dummyString)` to `value = optGetStrStr(optHandle, "GDX")`
- The compiled Python libraries are built with Python 2.7 now.
- This release contains a beta version of the object-oriented Python API that can be used to control GAMS from within Python 2.7. It allows the seamless integration of GAMS into Python by providing appropriate classes for the interaction with GAMS. The GamsDatabase class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the GamsJob class and by using the GamsModelInstance class a sequence of closely related model instances can be solved in an efficient way. To use this API one has to perform one of the following steps:

  Installing the API and the required low level APIs:

  ```
  cd <Path/To/GAMS>/apifiles/Python/api && python setup.py install && cd ..
  ```

  Using the API without installing:

  ```
  export PYTHONPATH=<Path/To/GAMS>/apifiles/Python/api  (Unix)
  set PYTHONPATH=<Path\To\GAMS>\apifiles\Python\api  (Windows)
  ```

  Running transport1.py example:

  ```
  export LD_LIBRARY_PATH=<Path/To/GAMS>:$LD_LIBRARY_PATH (DYLD_LIBRARY_PATH on OS X, not require
  python transport1.py
  ```

  Documentation about this API can be found in <Path/To/GAMS>/docs/API/GAMS_python.pdf.

Documentation

- added separate GAMS installation notes for Mac OS X

External Equations

- Added a list of examples to the GAMS Test Library.

GAMS

- If Solvelink is set to a non-default value. The GAMS log prints out the solvelink selection: `--- Executing SOLVER (Solvelink=i): elapsed 0:00:00.000`
- New date in audit line. Each GAMS component writes an audit line to the GAMS log, e.g. GAMS/Cplex write `IBM ILOG CPLEX Jul 14, 2011 23.7.3 WIN 27723.27726 VS8 x86/MS Windows`. The date in this line used to be the license check date, which does not change with maintenance releases (23.9.X). Starting with 24.0.1 the date displayed in the audit line is the build date of the system and changes with every release including maintenance releases.

- When GAMS is run from the command line on Windows systems and a solver is interrupted, the OS issued a message `Terminate batch job (Y/N)?` and the user had to interact. Now the message is still printed, but GAMS does not wait for a user action anymore. This has a rare side effect that solver keyboard interactions (e.g. in GAMS/Cplex with option `Interactive` and GAMS/SCIP with option `GAMS/Interactive`) will not work unless the new GAMS option `InteractiveSolver` is set to 1.
- For compile time commands like `$include`, the `$` had to be in column 1. Leading blanks are now allowed for such commands, but require an additonal `$`: `$$include`. The `$$` can only be used for the first dollar command in a line. For example, `$set a 'aa' $$set b 'bb'` does not work.
- GAMS has added UserName and ComputerName to the recorded fields of the GAMS Trace facility. Besides many uses in quality assurance testing, the trace facility e.g. can be used to audit the GAMS activities in a multi user environment with a shared GAMS installation.
- Report equations with general non-linearity in lst file for QCP models
- New `put_utility` feature `WinMsg` allows to send a Windows message to a window:

```
put_utility fx 'WinMsg' / 'WindowTitle' / 'Message';
```

- Asynchronous calls on Windows can be started with a new console rather than sharing the console of the parent process. `$CALL.ASYNCNC`, `Execute.AsyncNC` and `Put_Utility` keywords `Exec.AsyncNC` and `Shell.AsyncNC` have been added. `NC` and non `NC` calls behave identical on non-Windows platforms.

GAMS IDE

- The IDE has now an integrated Text differ to compare two text files; see File | Diff Textfiles

GDXRRW

- GDXRRW is a suite of utilities to import/export data between GAMS and R and to call GAMS conveniently from R. The software gives R users the ability to use all the optimization capabilities of GAMS, and allows visualization and other operations on GAMS data directly within R.
- GDXRRW is unique among the GDX interface utilities in that it is an R extension made available as an R package. As such, it is run as part of an R session or script, not as part of a GAMS run, and it follows the usual R package conventions.
- Source and binary packages for GDXRRW are part of the GAMS distribution.

3.35.1.4 Solvers

BARON

- New libs 11.8.0
- BARON uses an improved link to Clp
- BARON uses latest SNOPT 7.2-12
- BARON uses latest IPOPT 3.10.3
- Memory requirements have been significantly reduced

Bonmin

- Faster reaction to user interrupt during solve of initial NLP relaxation.
- Changed default setting for parameters variable_selection and milp_strategy to strong-branching and solve_to_optimality, respectively.
- Renamed miptrace options to solvetrace.

CBC

- New library 2.8
  - A new primal heuristic "Proximity Search" (proposed by Fischetti and Monaci, 2012) has been added. The idea is to define a sub-MIP without additional constraints but with a modified objective function intended to attract the search in the proximity of the incumbent. The heuristic is off by default and can be enabled by setting `proximitysearch 1` in a GAMS/Cbc options file.
- Parallel branch-and-bound search (multithreading) is now also possible under Windows by using [POSIX threads for Win32] (http://sourceware.org/pthreads-win32).
- New option dumpsolutions to output alternative solutions to gdx files.
- The miptrace options have been renamed to solvetrace and activating them does not affect the solving process anymore.

ConvertD

- Added option `localsolver` to convert GAMS models into [LocalSolver language] (http://www.localsolver.com). This is still an experimental feature, i.e., it may not always be possible to process the created .lsp files with LocalSolver. A major limitation is the restriction to binary and bounded integer variables in LocalSolver. To allow for continuous variables that can be defined via other variables, ConvertD can read information about *defined variables* from a ConvertD options file. In such an option file, a line `defvar x1 e1` denotes that the continuous variable x1 is defined via equation e1. Equation e1 then needs to be an equality constraint which contains x1 in a linear expression. Analogously, a statement `defvar y(n,i,j) e(n,i,j)` indicates that variable y(n,i,j) is defined via equation e(n,i,j) for all (n,i,j).

Couenne

- Couenne is now linked against the nauty library, which enables symmetry recognition for MINLPs and their utilization for orbital branching.
- The miptrace options have been renamed to solvetrace.

Cplex/CplexD

- New library 12.5
- The number of threads for CPLEX to use can now be set to any positive integer number, even if this number exceeds the number of cores on the machine. The default behavior of GAMS/CPLEX with respect to the number of threads and cores remains unchanged. That is, by default, GAMS/CPLEX continues to run with a single thread. Setting `THREADS` to 0, result in setting the threads to the number of cores.
- New parameters:
  - `ProbeDetTime`: Limits the amount of time (expressed in deterministic ticks) spent probing
  - `PolishAfterDetTime`: Sets the amount of time expressed in deterministic ticks to spend during a normal mixed integer optimization after which CPLEX starts to polish a feasible solution
  - `TuningDetTiLim`: Sets a time limit in deterministic ticks per model and per test set (that is, suite of models) applicable in tuning
  - `RandomSeed`: Sets the random seed differently for diversity of solutions.
- GAMS/Cplex prints the time spend in Cplex optimization calls. The time is printed in elapsed seconds and elapsed deterministic time in ticks.
- GAMS/CplexD returns proper duals for QCP models

CONOPT

- new library 3.15H

GloMIQO

- new library GloMIQO 2.1:
  - improved reformulation strategies
  - better recognition of special mathematical structure allowing for dominant polyhedral cuts
  - additional strategies for deterministically inferring missing variable bounds
- new option cplex_optfile specifies a GAMS/CPLEX options file that will be applied to every LP and MILP subsolve
- new option dumpsolutions to output alternative solutions to gdx files

GUROBI

- New library 5.0.2

IPOPT and IPOPTH, BONMIN and BONMINH

- IPOPT is now available in two variations. The Open Source IPOPT is part of the Base Module. IPOPTH uses higher performance (but not Open Source) linear algebra routines (HSL) and is separately priced.
- Analogously, BONMIN is part of the Base Module and uses IPOPT. BONMINH requires a license for IPOPTH.
- For Couenne and SCIP, IPOPTH is used to solve NLP subproblems if a corresponding license is available, otherwise IPOPT is used.

LogMIP

- Fixed a bug when reformulating disjunctions with terms that are indicated by a negated variable (disjunction not y1 e1 else ...).

MINOS

- Maintenance updates, primarily for memory allocation issues.

MOSEK

- New libs Mosek 6 Rev 148.

Optimization Services

- New library 2.6

OsiXpress

- OsiXpress is now also available on Mac OS X.

SCIP

- New library 3.0.1
  - New presolvers have been added, among them one that recognizes and utilizes block structures in an instance.
  - New primal heuristics NLP diving (for MINLPs) and Zero-Objective ("Hail Mary") have been added.
  - New propagators for optimization-based bound tightening for MINLPs have been added.
  - The variable bounds propagator has been extended to cliques and implications.
  - Memory limits are now better handled, especially for larger problems.
  - The probing algorithm has been revised and should give better performance.
  - The performance for pseudo-boolean optimization problems has been improved.
  - The complete release notes can be found [here] (https://www.scipopt.org/doc/html/RELEASENOTES.shtml) and a list of all changes (including changes to parameters and their default values) can be found [here] (https://www.scipopt.org/doc/html/CHANGELOG.shtml)
- The GAMS workspace option can now be used to set the SCIP memory limit ('limits/memory' in optionfile).
- The `gams/miptrace` options have been renamed to `gams/solvetrace`.
- The `gams/printstatistics` option has been renamed to `display/statistics`.
- The `gams/interactive` option is now a string option that takes commands for the SCIP shell as arguments. If the commands do not end with a 'quit' command, the SCIP shell still open for user interaction. Note, that in order to be able to input commands interactively on Windows, you have to set the GAMS option interactivesolver to 1 (see above).
- The LP solver in SCIP can now be changed by the `lp/solver` option. Supported values are "cplex", "soplex", and "clp". The default LP solver is now CPLEX, if a CPLEX license is available, and SoPlex otherwise.

SNOPT

- New library 7.2-12 (maintenance release)
- Completely revamped linking code
  - The GAMS/SNOPT link is now thread-safe, allowing it to be used in multi-threaded applications built with the high-level GAMS APIs mentioned above.
  - Updated memory allocation scheme to increase the amount of memory that can be allocated and used for problem solution, and to improve reporting when limits are reached.
  - Miscellaneous bug fixes and usability improvements.

SoPlex

- New library 1.7.1
  - Performance improvements in pricing algorithms
  - Many bugfixes and improvements in numerical stability and infeasibility detection
  - Complete release notes [here] (http://soplex.zib.de/notes-170.txt).

Sulum

- GAMS 24.0 introduces the new LP/MIP Solver Sulum from Sulum Optimization ApS.
- While Sulum currently does not compete with the high-end LP/MIP solvers, Sulum offers a good cost-benefit ratio for LP and MIP solution technology.
- GAMS 24.0.1 features Sulum LP only, but as soon as Sulum Optimization releases their library including the MIP optimizer, a maintenance release will feature the full LP/MIP capabilities of Sulum.

Xpress

- New library Xpress Optimizer 23.01.06 (maintenance release).

3.35.1.5 Model Libraries

GAMS Data Library

GAMS EMP Library

- **simplevi4 (93)**: Equilbrium model consisting of two VIs, one of which has a non-trivial constraint set
- **transsp (94)**: A Stochastic Transportation Problem
- **oa (95)**: Outer Approximation for Convex Minimization Problem with Binary Variables

GAMS Model Library

- **powerset (381)**: PowerSetLeft and PowerSetRight examples
- **linearne (382)**: Linearization techniques for extremal-Nash equilibria
- **saras (383)**: South African Regionalized Farm-level Resource Use & Output Supply Response (SARAS) model
- **epscmmip (384)**: Improved version of eps-Constraint Method for Multiobjective Optimization
- **bidpwl (385)**: Bid Evaluation with Piecewise Linear Functions
- **trnspwlx (386)**: A Transportation Problem with Piecewise Linear Functions

GAMS Test Library

- **testexeq (562)**: Test external equations
- **ex1 (563)**: External Function - Example 1
- **ex1x (564)**: External Function - Example 1x
- **ex2 (565)**: External Function - Example 2
- **ex3 (566)**: External Function - Example 3
- **ex4 (567)**: External Function - Example 4
- **ex4x (568)**: External Function - Example 4x
- **ex5 (569)**: External Function - Example 5
- **er1 (570)**: External Function - Error Example 1
- **er2 (571)**: External Function - Error Example 2
- **er3 (572)**: External Function - Error Example 3
- **exmcp1 (573)**: External Function - Example MCP 1
- **exmcp2 (574)**: External Function - Example MCP 2
- **exmcp3 (575)**: External Function - Example MCP 3
- **exmcp4 (576)**: Hansen/Koopmans: External Function - Example MCP 4
- **exmcp5 (577)**: Intermixed External Rows: External Function - Example MCP 5
- **complink (578)**: Compile and link external equation libraries
- **lindgl02 (579)**: Check that Lindo(Global) works with SOS variables
- **qcp10 (580)**: Test for QCP correctness
- **empdisj6 (581)**: Test disjunctions with negated equality equations
- **gdx9 (582)**: Test unloading and loading a GDX file with variable attributes
- **empdisj7 (583)**: Test disjunctions with negated variables
- **traceuc (584)**: Test if we can get the user and computer name in a trace file
- **asynntrp (585)**: Start GAMS job asynchronously and send interrupt signal to it

3.35.1.6 Tools

SCENRED2

- fixed visualization output to work with recent versions of gnuplot

3.35.1.7 Solver/Platform Availability Matrix

3.35.2 24.0.2 Maintenance release (February 14, 2013)

3.35.2.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Arne Drud, Michael Ferris, Erwin Kalvelagen, Aida Khajavirad, Ignacio Perez, Nick Sahinidis, and Dimitri Tomanos.

3.35.2.2 Platforms

- removed dependency on GLIBC 2.7 in Linux systems

3.35.2.3 GAMS System

APIs

Object Oriented APIs

- introduced new function CopySymbol (see individual languages for precise name)
- label and symbol lookup is now case insensitve
- fixed handling of infinite bounds for GAMSModifiers

.NET

- added new function GAMSSymbol.CopySymbol
- added new examples in Visual C++ and VB.NET

Java

- added new function GAMSSymbol.copySymbol

Python

- added new function _GamsSymbol.copy_symbol
- on Windows and Linux, compiled libraries for both Python 2.6 and Python 2.7 are distributed while the Python 2.7 version can be found in <GAMSDir>\apifiles\Python\api as before, the Python 2.6 version is in <GAMSDir>\apifiles\Python\api_26

GDX

- fixed a casing problems with the function `gdxSymbolGet/SetDomainX` for client languages C++, C#, Fortran, Java, Python, VBA, VB.NET

GMO

- increased API version number to 11 (was forgotten in 24.0.1)

GAMS

- the reason for ignoring an option in the GAMS options file reader is now printed
- fix a bug when reading scalars from a compressed GDX file

GAMS IDE

- changed the interface for the Text Differ to look like the GDXDiff interface
- moved menu for Text Diff under Utilities menu

Utilities

GDXXRW

- gdxxrw works now with Excel Binary Workbook files (file extension `.xlsb`)

3.35.2.4 Solvers

BARON

- new library 11.9.1
- fixed handling of workspace and workfactor options

BONMIN, CBC, Couenne, GloMIQO, SCIP

- fixed too small value in 'seconds' column in solvetrace file
- improved resolution of solvetrace file for Cbc and SCIP

CONOPT

- new library 3.15I
- major update of the CONOPT solver manual (pdf file only)
- revised definition of workspace and workfactor options: workfactor is now ignored if workspace is defined
- minimum memory allocation adjusted for some smaller machines
- removed no longer used options GCFORM, GCPTB1, GCPTB2

GloMIQO

- fixed bug in reporting of dual bound when optca tolerance is larger in magnitude than optcr

GUROBI

- new library 5.1
- new options
  - `ImproveStartNodes`: A new option for transitioning from tree exploration to solution improvement in MIP
  - `Seed`: Modifies the random number seed. This acts as a minor perturbation to the solver, which typically leads to a different solution path. This can be useful for experimentation (e.g., for testing the robustness of a particular set of parameter changes)
- Gurobi dropped support for Linux 32bit, so GAMS/Gurobi uses Gurobi library 5.0.2 on Linux 32bit

MOSEK

- write clearer text error message to log and listing file

MPSGE

- fixed memory issues when both workspace and workfactor are set and when estimates are too high

MSNLP / OQNLP

- fixed problems with LSGRG on instances with more than 10000 variables or more than 10000 equations

SCIP

- new SCIP library 3.0 #0134f8c
- fixed bug where the reported objective value did not equal the level value of the objective variable
- improved cleanup of SCIP solutions that are not feasible in the original problem; the model status is now adjusted accordingly

XPRESS

- fixed reporting of resused and iterused attributes for infeasible and unsolved models

3.35.2.5 Tools

ConvertD

- in LocalSolver output, long linear parts of equations are now printed by using the sum() operator
- added solution output option and functionality for LocalSolver (lspsol)
- fixed bug when using objvar option in conversion to GAMS format
- fixed bug with variable indices when writing nonlinear expressions

EXAMINER

- fixed check of complementarity slackness for discrete variables in MIQCPs
- added a consistency check on model attributes (currently only objval)

JAMS

- fixed many problems with objective functions in equilibrium followers, especially those arising when variables were skipped or when the objective var was reformulated out
- fixed problems with VI models and vars getting squeezed out that should not have been

3.36 23.9 Distribution

3.36.1 23.9.1 Major release (July 04, 2012)

3.36.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Steve Gabriel, Oyvind Hoveid and Renger van Nieuwkoop.

3.36.1.2 Platforms

- We added some instructions on how to install the GAMS Windows version (including the IDE) on the Mac using Wine.
- The Unix installer has already been changed with 23.8. When running the installer, a subdirectory is created (e.g. `gams23.9_linux_x86_32_sfx` for the Linux 32bit installer). Previous versions unpacked the GAMS distribution into the current directory.

3.36.1.3 GAMS System

APIs

.NET

This release contains a beta version of the object-oriented GAMS .NET that can be used to control GAMS from within C# and other programming languages of the .NET framework 4. It allows the seamless integration of GAMS into the .NET environment by providing appropriate classes for the interaction with GAMS. The `GAMSDatabase` class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the `GAMSJob` class and by using the `GAMSModelInstance` class a sequence of closely related model instances can be solved in the most efficient way.

To use this one has to reference `GAMS.net.dll` which can be found in the GAMS system directory. Documentation about this can be found in `<GAMSDir>\apifiles\GAMS.net4.chm`.

C#

- Gamsx, gdx and opt implement IDisposable interface
- New example example2.cs using APIs to gamsx, gdx and opt
- Several new examples using the new .NET

GDX

- Added functions to get and set relaxed domain information
- Added function to get memory usage of a gdx object

GAMS

On/OffOrder

- Lag and lead operations require the reference set to be ordered and constant. In some special cases one would want to use lags and leads on dynamic and/or unordered sets. A dollar control option `on/offOrder` has been added to locally relax the default requirements. The use of this option comes at a price: the system will not be able to diagnose odd and incorrect formulations and data sets. A small example in Test Library **lagd1** illustrates the use of `on/offOrder`.

gamsbatch

- The script to run GAMS in the background (`gamsbatch`) has been removed from Unix distributions. This script was a source of confusion when multiple GAMS systems were installed. A GAMS job can be run in the background easily without this script.

Minor enhancements

- GAMS solvers that provide a bound on the optimal solution (e.g. Branch-and-Cut based solvers) can return this bound to the calling GAMS model through the model attribute `objest`. In previous versions of GAMS this model attribute was not set if no solution was found. This has been fixed with this release.
- When GAMS/Base writes a GDX file containing a subset of all symbols present, it will now also write relaxed domain information to the GDX.

Function libraries

- The new extrinsic function library `lsadclib` was added to the system. It allows to use sampling routines from Lindo inside GAMS. A license for GAMS/Lindo is required to use this library. Without a license only a demo version is available which is restricted to the Normal and the Uniform distribution with not more than 10 sample points. Further information can be found in the GAMS User's Guide, Appendix J.

Utilities

GAMSIDE

- The data viewer in the GAMSIDE shows the domain information
- New history combo box for alternate system directory

GDX2Access

- Use relaxed domain information to generate column names

GDXDUMP

- Added option to write CSV formatted file (Format=[normal, gamsbas, csv])
- Added option to use last dimension as a column header for CSV output (CDim=[Y, N])
- Added option to not filter default values (FilterDef=[Y, N])
- Option to show domain information (DomainInfo)

XLSDump

- Truncate long UELs

XLSTalk

- Allow for a relative path in the file specification

Other

3.36.1.4 Solvers

Baron

- BARON 11: This version comes with a wealth of new branching, relaxation, convexity exploitation, local search, and range reduction techniques. You will see improvements in many problems, including convex ones. In addition, BARON 11 offers systematic treatment of missing bounds through safe LP relaxations. Users will see far fewer messages about missing bounds from BARON from now on, while many more problems will be guaranteed global.

Coin-OR

- new libraries Bonmin 1.6 and OS 2.5
- bugfixes in Cbc and Couenne

Conopt

- New libraries 3.15F

Cplex

- new libraries 12.4 fix pack 1

GloMIQO

- New library GloMIQO 2.0
- GloMIQO 2.0 alternatively uses Conopt or Snopt for finding feasible solutions (default Conopt).
- GloMIQO 2.0 fully integrates integer variables, discrete/discrete products, and discrete/continuous products
- GloMIQO 2.0 implements a branch-and-cut framework with cutting planes derived from edge-concave aggregations, alphaBB cuts, and convex terms. Cuts are based on both individual equations and the collection of quadratic and bilinear terms in MIQCQP.
- GloMIQO 2.0 automatically adds bilinear terms to the model formulation to create Reformulation-Linearization Technique (RLT-1) equations.
- The Boost Interval Arithmetic Library improves the reliability of the GloMIQO 2.0 bound tightening schemes.

Gurobi

- New library Gurobi 5.0.1
- Quadratic constraint support: You can now add Second-Order Cone constraints, rotated Second-Order Cone constraints, and general Quadratic Constraints to an optimization model. Continuous models containing these constraints are solved using a new barrier SOCP solver. Mixed-integer models are solved using a branch-and-cut algorithm, employing either QCP node relaxations or an outer-approximation algorithm.
  - The new options are:
    - `BarQCPConvTol`: Convergence tolerance for the barrier algorithm when solving a QCP.
    - `PreQLinearize`: Controls linearization of Q matrices in the quadratic constraints or a quadratic objective.
    - `QCPDual`: Determines whether dual variable values are computed for QCP models. Gurobi has this off by default. In GAMS/Gurobi this is on by default.
    - `MIQCPMethod`: Determines whether outer approximation is used to solve an MIQCP model.
- Feasibility relaxation: Gurobi now provides a feasibility relaxation model for an infeasible model. This related model will find a solution that minimizes the violation of the original constraints. Check option `FeasOpt` for details.
- Simplex warm-starting from a solution: You can now warm-start the simplex method using a primal and dual solution vector. In previous versions, warm-starting was only possible if you had a simplex basis. Check option `UseBasis` for details.
- Barrier homogeneous algorithm: Gurobi now provides a homogeneous algorithm in the barrier solver. This version should be used when you are likely to ask the barrier solver to solve infeasible or unbounded models. Use the new `BarHomogeneous` option to select this.
- Gurobi now provides both exact and approximate condition number estimates for the optimal simplex basis. Use `Kappa` if you want a quick estimate, or `KappaExact` if you want to spend the sometimes substantial time required to compute the exact condition number.

GUSS

- Introduce option `NoHotStart` to suppress hot starts with hot start capable solver.
- Temporary scenario files are located now in the scratch directory, so multiple GUSS runs can take place in the same directory without interference.

MOSEK

- New libraries Mosek 6 rev 137

PATH

- Bug fix in preprocessor

SCIP

- New library SCIP 2.1.2

Xpress

- New libraries: Xpress Optimizer 23.0.05
  - Improved Barrier performance, especially on modern Intel CPUs. The Barrier solver will now take advantage of Intel's Advanced Vector Extensions (AVX) when available.
  - New symmetry detection and orbital branching for MIPs.
  - Improved in-tree cutting strategy.
  - Added some quick heuristics that will be run before the initial root LP is solved.
- New platform supported: Mac OSX 64-bit
- Minor updates to link
  - bugs fixed: handling of QCP models with equality constraints, interrupted solves
  - pass the objective constant on to Xpress Optimizer
  - add new options to control XPRESS symmetry detection

3.36.1.5 Documentation

- Updated Expanded GAMS Guide (McCarl) to reflect release 23.9
- The solver Manual has been split into two parts: *I The Commercial Solvers* and *II The Free Solvers*. Printed versions of the manual will only contain part I. The on-line/ version has both parts available.

3.36.1.6 Model Libraries

GAMS Data Library

- **TrnsxcllStarter (83)**: Excel Spreadsheet in Charge of GAMS

GAMS EMP Library

GAMS Model Library

- Updated:
  - **qp7 (271)** and **emfl (273)**: Both models use second order cone constraints. The constraints are written with the `=c=` syntax if MOSEK is the selected solver and as general quadratic constraints if CplexD or Gurobi are the selected solvers
  - **feasopt1 (314)**: This model has been modified so it can be used by Cplex and Gurobi. Both solvers support the `FeasOpt` option to find minimal relaxations for infeasible models
- New:
  - **ps5_s_mn (377)**: Parts Supply Problem w/ 5 Types w/ Random p(i)
  - **iobalance (378)**: Updating and Projecting Coefficients: The RAS Approach
  - **fdesign (379)**: Linear Phase Lowpass Filter Design
  - **pmeanvar (380)**: Mean-Variance models with variable upper and lower bounds

GAMS Test Library

- Updated:
  - **circlen (551)**: Added an SOCP formulation to the smallest circle problem
- New:
  - **mpsge12 (558)**: MPSGE test - inter-mixed MPSGE and MCP rows in model
  - **gdxcomp2 (559)**: GDX - compressed and MT operation
  - **lagd1 (560)**: Lag and Lead operations on Dynamic Sets
  - **miqcp02 (561)**: Test of correctness of solvestat

3.36.1.7 Solver/Platform Availability Matrix

3.36.2 23.9.2 Maintenance release (August 29, 2012)

3.36.2.1 GAMS System

APIs

.NET

- `GAMSModelInstance.Instantiate` is now thread-safe
- Additional flag `createOutDB` for method `GAMSJob.Solve` which allows to switch off automatic `outDB` creation
- New class `GAMSModelInstanceOpt` to customize method `GAMSModelInstance.Solve`
- Delete some temporary files, add `GAMSWorkspace.ScratchFilePrefix` (default: `_gams_net`) for temporary files
- `GAMSJob` no longer implements `IDisposable`; examples have changed because the `using` statement no longer works with `GAMSJob`

Java

- Now methods that returned `long` in Java but `void` in C also return `void` in Java

GAMS

MaxProcDir

Changed default for GAMS parameter `MaxProcDir` from 26 to 700 which allow to have up to 700 scratch directories (225a, 225b, ...) in the project/current directory

Utilities

GAMSIDE

Double-click to open a file in the IDE works again (also fixes a single click in the McCarl guide)

3.36.2.2 Solvers

AlphaECP

- Fixed objective estimate `modelname.ObjEst`

Baron

- New BARON 11.3.0 library (bugfixes)

Coin-OR

- New Couenne library (bugfixes)
- New Bonmin library (bugfix for B-OA as algorithm choice)

ConvertD

- New option `DictMap` in ConvertD

Cplex

- CplexD: fixed bug when registering an advanced basis

GloMIQO

- New GloMIQO library (bugfixes)

Gurobi

- New Gurobi 5.0.1 library for AIX
- OsiGurobi: fixed mipstart option (now only values for discrete variables are passed to Gurobi)

Jams

- New option `DisjBinRelax` in JAMS to allow continuous (but implicit binary) variables

Knitro

- Work around problems with knapsack cuts for MINLPs

Lindo

- New Lindo 7.0.1.487 library (bugfixes)

SCIP

- Fixed wrong sign of objective when maximizing a nonlinear objective

3.36.3 23.9.3 Maintenance release (September 26, 2012)

3.36.3.1 GAMS System

APIs

.NET

- Constructor of `GAMSWorkspace` now appends conditionally the GAMS system directory to the PATH environment variable. This fixes a problem with applications that create and dispose many `GAMSWorkspace` instances.

GAMS

- Spaces in Project/Current Directory: With 23.9.2 some solvers (e.g. BARON) do not work if the project/current directory name contains a space. This has been fixed.

Utilities

- GDXMRW: Some bugfixes for the GAMS/MATLAB connector.

3.36.3.2 Solvers

Coin-OR

- New Couenne library (bugfixes)

3.36.4 23.9.4 Maintenance release (October 20, 2012)

3.36.4.1 GAMS System

GAMS

- GAMS did not pass models with quadratic constraints containing the objective variable properly to some solvers (CplexD, Gurobi, Mosek and Xpress). Thanks goes to [Erwin Kalvelagen] (http://yetanothermathprogrammingconsultant.blogspot.com/2012/10/gamsgurobi-qcp-problems.html) bringing this to our attention. This has been fixed.
- The GAMSIDE remembers the scope setting for text searches even after the IDE closes.

3.36.4.2 Solvers

Baron

- New libs 11.5.0
- Fixed a bug when using the external NLP solver on models without constraints
- New speed ups in Baron's parser/reformulator for large scale problems
- Improvements in local search that result in higher quality solutions identified earlier in the search
  - New dive-and-round heuristic for MINLPs. The number of rounds is selected dynamically and the search is activated and applied based on problem characteristics
  - For continuous problems, Baron introduces the ability to automatically select and switch back and forth between NLP solvers. Again, this is done in a dynamic strategy that exploits problem characteristics and solver performance for the problem at hand
  - The `nlpsol` option can now take the value of `-1` in addition to the previously allowed values. `-1`, which is the default value, triggers the automatic NLP solver selection and switching strategy. If the user sets `nlpsol` to a solver that is not licensed, Baron sets it automatically to `-1`
  - The `dolocal` option can now take only two values (`0: no local search; 1: BARON's automatic strategy`). Default is `1` and will determine when/how to apply local search
  - Baron uses the most recent Snopt library (7.2-4)
  - Improvement for links to NLP solvers Minos, Snopt, and IpOpt.

Conopt

- New 3.15G library with bug fixes

GloMIQO

- A problem with parsing integer options has been fixed

Minos

- Improved memory estimation calculations and logging

3.36.5 23.9.5 Maintenance release (November 09, 2012)

3.36.5.1 GAMS System

**GDX
**

- When importing a scalar variable or equation from a GDX file, the `.scale` attribute was set incorrectly. This has been fixed. Thanks go to Étienne Ayotte-Sauvé unearthing this problem.
- Under some circumstances GAMS ended up with an inconsistent database when importing unsorted data. This has been fixed. Thanks go to Wolfgang Britz unearthing this problem.

3.36.5.2 Solvers

Baron

- New libs 11.5.2
- Fixed a reporting problem when no duals are available.

CplexD

- GAMS/CplexD produced in some cases incorrect quadratically constraint and SOCP problems for the Cplex engine. This has been fixed.

LindoGlobal

- Lindo's global solver cannot deal with SOS variables. If a model has SOS variables, the link automatically switches to the local solver.

LogMIP

- Fixed bugs in `bigm` and `indic` reformulation of negated equality equations. The convex hull reformulation still has problems and has therefore been disabled for this case for now. Thanks go to Silvia Tomasi for reporting this problem.

3.37 23.8 Distribution

3.37.1 23.8.1 Major release (March 17, 2012)

3.37.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Alessandro Brozzi, Jan Philipp Dietrich, Michael Ferris, Christophe Gouel, Josef Kallrath, Cheng Khor, Brage Knudsen, Jeff Larson, Yanchao Liu, Alexander Mitos, Renger van Nieuwkoop, Stefan Vigerske.

3.37.1.2 Platforms

- Beta GAMS/Wine system (32bit). Comments are highly appreciated (support@gams.com).
- Improved Windows Installer
  - the default location has changed to `C:\GAMS\(win32|win64)\23.8` to avoid problems produced by the permission settings of the *Program Files* location
  - Two different setup modes. A simple default mode and an advanced mode with additional options like automatic *PATH* manipulation
  - Support for detecting and installing a license from the clipboard

3.37.1.3 GAMS System

APIs

- C#
  - Major improvements, lost compatibility to previous versions!
  - Mixed casing of function names
    - old: gdxdatawritestr
    - new: gdxDataWriteStr
  - Object handle (pointer) no longer needed
    - old: int gdxdatawritestr(IntPtr pgdx,string[] keystr,double[] values)
    - new: int gdxDataWriteStr(string[] KeyStr,double[] Values)
  - Create calls replaced by different constructors
    - old: gdx = new gdxcs(); gdx.gdxCreateD(ref PGX, Sysdir, ref Msg);
    - new: gdx = new gdxcs(Sysdir, ref Msg);
  - See examples in <`GAMS Dir`>\`apifiles`\`CSharp` for reference
- Python
  - Add source code to build C extensions (no need to run SWIG anymore)
  - Add compiled C extension on Linux and Windows (both 32 and 64 Bit)
- VBA
  - Support for 64 Bit version (VBA 7)

GAMS

New model definition features

- **Flexible model list**
  Model definitions containing complementary equ.var pairs can now be used with all model types. The matching information is only used with the appropriate model types (e.g. MCP, MPEC, RMPEC).
- **Model list edits**
  In addition to the ',' which enforces uniqueness of equation entries, two additional model list element separators have been introduced. The '+' separator allows replacement of existing entries. The '-' separator deletes model list entries. All three of these operations can be used with single equation names, equ.var pairs and previously defined model names.
- **Example**

```
$eolcom !
equation e1,e2,e3,e4,e5; variable x1,x2;
e5.. x1 =l= 0;
e4.. x1 =l= 0;
model m1 / e1,e2 /
      m2 / e3,e4 /
      m3 / m2,m1,e5 /                ! e3,e4,e1,e2,e5
      m4 / all-m1+e1+e1 /            ! e3,e4,e5,e1
      m5 / all + e5.x1 + e4.x2 -e1/  ! e2,e3,e5.x1,e4.x2
      m6 / m5 -e5, e5.x2 /           ! e2,e3,e4.x2,e5.x2
      m7 / m5 + e5.x2 /              ! same as m6
```

Note that the equation list is ordered left to right. If we have a '+' separator adding an equation already in the list, the equation (possibly modified) will move to the end of the list.

Permutations via the option statement

- Using the '>' sign one can create complete permutations of one- and multi-dimensional sets and parameters:

```
Option PermAllI > I;
```

  Examples of this new feature can be found in the models **perm1**, **ptsp**, **flowshop**, and **pmelody**.

Minor enhancements

- Functions can have up to 10 arguments. This effects the intrinsic functions `edist` and `poly` and extrinsic function libraries.

Stochastic Programming

In this release we made a first cut at supporting stochastic programming in GAMS. With a few changes uncertainty can be added to an existing deterministic model. For this, the EMP framework is used to replace parameters in the model by random variables. This way (stagewise-independent) multi-stage recourse problems and chance constraint models can be formulated. Three solvers can be selected to solve those problems: DE, DECIS and LINDO.

Further information can be found in the document Stochastic Programming (SP) with EMP.

In addition, you can check examples of type `SP` from the **GAMS EMP Library**.

Utilities

- **mps2gms** MPS input file allows LI (lower integer) bound types and treats tabs as spaces.

Other

- By default GAMS does not call an exit script anymore. If this is required, the GAMS parameter `ScriptExit` has to be set explicitly to the script that should be called after GAMS terminates. An empty template of an exit script can be found in the GAMS system directory (`gmsxitnt.cmd` (Windows) or `gmsxitus.run` (Unix)).

3.37.1.4 Solvers

AlphaECP

- New libraries 2.09.01
- Support of =N= rows

Baron

- New libraries Baron 10.1
- Main developments
  - Minos and Snopt are no longer available as LP subsolvers
  - Clp can be used as the LP subsolver (`lpsol = 8`)
  - Ipopt with Mumps and Metis can be used as the NLP subsolver (`nlpsol = 9`)
  - BARON automatically selects Clp and Ipopt if the selected/default subsolver is not licensed
  - Improved interface to Xpress-LP
  - Introduced cutting planes for convexity exploitation for a variety of functions, including products and ratios, perspectives, norms (including absolute functions) and seminorms, and quadratic functions that are automatically recognized and exploited
  - Polyhedral convexification routines have been rewritten and are now more efficient and numerically robust, as well as performing a number of simplifying transformations that reduce relaxation gaps
  - Introduced relaxations based on inferred bilinear relationships (certain types of first-order RLT)

Bench

- calls solvers to benchmark in parallel using GAMS' grid functionality

Coin-OR

- OSIXpress supported on Solaris/Intel
- Dropped support of GLPK (alias OsiGlpk, alias CoinGlpk)
- New libraries:
  - Bonmin 1.5
  - Cbc 2.7
    - improvements in heuristics and cut generation
  - Clp 1.14
  - Couenne 0.4 (for details see the [Couenne release notes] (http://list.coin-or.org/pipermail/couenne/201
    - feasibility pump heuristic for MINLP (can use SCIP as MIP solver)
    - fixed point bound tightening and two-implied bound tightening
    - introduction of semi-auxiliaries for smaller LP relaxations
  - Ipopt 3.10
  - OS 2.4
    - now supports semicontinuous and semiinteger variables
  - Osi 0.105
- Cbc, Bonmin, and Couenne support `miptrace`

Convert

- Convert supports writing [Optimization Services instance Language (OSiL)] (http://www.coin-or.org/OS/OSiL.ht

Conopt

- New libraries 3.15C

Cplex

- new libraries 12.4
- `dettilim`: option to limit 'ticks'

DE

- `GAMS/DE` accepts EMP models that have been annotated with information about uncertainty. The DE 'solver' reformulates the stochastic model into the extensive form equivalent with implicit non-anticipativity constraints. The reformulated model is solved with any of the regular GAMS solvers. All optimization model types (LP, MIP, QCP, MIQCP, NLP, DNLP, and MINLP) are accepted.

DECIS

- `GAMS/DECIS` accepts 2 stage EMP stochastic models. The stochastic information is processed and is passed to the well known `GAMS/DECISC` or `GAMS/DECISM` solver. The need to provide the stochastic information based on the matrix elements is now unnecessary. `DECIS`, as `GAMS/DECISC` and `GAMS/DECISM`, accepts linear recourse models only (although the stochastic effects might be more complex).

DICOPT

- If the model given to DICOPT is linear, DICOPT just passes this model to the MIP solver. Previously, the MIP solver used parameters OptCR, OptCA, and ResLim as specified by GAMS options even if DICOPT used an option file. Now OptCR, OptCA, and ResLim from a DICOPT option file will overwrite the GAMS parameters.

GloMIQO

The new Global Mixed-Integer Quadratic Optimizer, GloMIQO (GLO-me-ko), solves Quadratically Constrained Programs (QCP) and Mixed-Integer Quadratically Constrained Programs (MIQCP) to epsilon-global optimality.

- GloMIQO has been developed by Prof. C. Floudas and R. Misener from Princeton University
- The GloMIQO solver is available for the 32-bit and 64-bit versions of Windows and Linux
- GloMIQO requires the presence of a GAMS/CPLEX and GAMS/SNOPT license

Gurobi

- New library Gurobi 4.6.1
- New sifting option for linear programs: LP models can now be solved with a sifting algorithm. Gurobi will choose sifting automatically when using the dual simplex algorithm for models that are likely to benefit (those with many more columns than rows). If you want to invoke sifting manually, choose dual simplex and set the new `Sifting` parameter to a value greater than 0.
- Branch priorities: You can now specify priorities for branching variables, allowing greater user control of the MIP search process. Priorities can be specified through the .prior variable attribute and the .prioropt model attribute.
- New Zero Objective heuristic: Gurobi 4.6 contains a new Zero Objective heuristic that can be useful for finding solutions to MIP models where other strategies fail to find feasible solutions in a reasonable amount of time. Use the new `ZeroObjNodes` parameter to control this new heuristic.
- New presolve sparsify option: A new presolve sparsify algorithm is available for MIP models. This method attempts to reduce the number of non-zero coefficients in the constraint matrix. Reductions of 3X or more have been observed on some models, often leading to significant performance improvements. Use the new `PreSparsify` parameter to enable this algorithm.

Guss

- Added option `NoMatchLimit` (default 0) that controls the maximum number of accepted unmatched scenario records before terminating the solve.

Ipopt

- The Serial Graph Partitioning and Fill-reducing Matrix Ordering software METIS can now be used by some of Ipopt's linear solvers.
- Starting with 23.8 GAMS offers a commercially supported version of GAMS/IPOPT. This version of GAMS/IPOPT includes the commercial strength linear solvers MA27, MA57, and MA86 from the Harwell Subroutine Library.

Jams

- Support Xpress and SCIP when reformulating disjunctions with indicators

Knitro

- New libraries Knitro 8.0
- Introduction of a presolver to simplify and reduce the size of the model
- Introduction of parallel (multi-threaded) execution
  - Multi-start
  - Multi-algorithm
- Multi-algorithm method
  - New mode to run multiple algorithms, serially or in parallel
  - Uses the first or (optionally) the best solution found
- New feasibility-only mode: improves robustness and efficiency of detecting infeasible models
- Knitro 8.0 discontinues support for 32-bit Darwin systems - only 64-bit systems are supported on the Mac.

Lindo

- New solver Lindo, similar to LindoGlobal, but
  - Without size limitations
  - No requirement for additional Conopt license
  - `GAMS/LINDO` accepts stochastic EMP models and solves either the extensive form equivalent or uses Benders' decomposition to solve the stochastic model. In addition to discrete random variables, Lindo features also continuous distribution functions and various sampling techniques. All optimization model types (LP, MIP, QCP, MIQCP, NLP, DNLP, and MINLP) are accepted.

LindoGlobal

- New libraries Lindo 7.0
- MIP
  - Significant improvements in root node heuristics for quickly finding good integer feasible solutions
  - Improved identification of special structures, as in multi-period models, and the ability to exploit these structures to achieve significant reductions in solve times
- Global Solver
  - Improved heuristics for finding a good feasible solution quickly
  - Improved ability to identify constraints that can reformulated as second order cone (SOC) constraints and thus be solved by the fast SOC solver
  - Improved ability for efficiently handling polynomial terms
  - Improved bounds for non-convex quadratic terms, using SDP and eigenvalue reformulations
  - Improved control over multi-start algorithm

MOSEK

- New libraries Mosek 6 rev 135

SCIP

- New libraries SCIP 2.1.1 (for details see the [SCIP release notes] (https://www.scipopt.org/doc-2.1.1/html/RELE
- Beta-version of supporting (mixed-integer) nonlinear programs (also nonconvex)
- Supports `miptrace`
- Supports indicators for linear constraints through option `gams/indicatorfile`
- Added option `gams/dumpsolutions` to write all solutions found into gdx solution pool

Snopt

- bug fix for log of LP models

SoPlex

- New libraries SoPlex 1.6.0 (for details see the [SoPlex release notes] (http://soplex.zib.de/notes-160.txt))

XA

- New libraries XA 17 on Windows
- Introduction of native 64-bit Windows support

Xpress

- New libraries 22.01.15 (maintenance release)
- MIP solution pool capability - allows a number of feasible solutions to be stored for later processing. By default the global search is not altered, but there is an option to enumerate all or selected integer feasible solutions. The enumeration can be tailored via the option file to execute a number of different strategies, e.g.:
  - all integer feasible solutions
  - N-best solutions
  - as-good-as-X solutions (all solutions with objective at least as good as X)
  - N-first solutions (first N solutions found)
- Solution pool examples are available in testlib: **xpress03** (default global search), **xpress04** (enumeration of all solutions), **xpress05** (various enumeration strategies).
- Support for continuous models with quadratic constraints (previously only quadratic objectives were supported) and for all mixed-integer quadratic (MIQCP) models
- Support for indicator constraints
- New option `isGoodEnough=X` implemented in the link. If set, the global search will stop once an integer feasible point is found with an objective at least this good. Surprisingly, none of the MIP solver vendors implement such an option directly in their libraries.

3.37.1.5 Documentation

Amongst others, the following topics were revised in both, the **GAMS Users Guide** and the **Bruce McCarl's Expanded GAMS Guide**

- Command Line Parameters
- Dollar Control Options
- Model Attributes
- GAMS Intrinsic and Extrinsic Functions

3.37.1.6 Model Libraries

GAMS Data Library

- **GDXMRWPlotting01 (81): Placeholder for the GDXMRW examples that deal with plotting in Matlab**
- `makeql (82)`: Create Querry Library routines for extrinsic functions

GAMS EMP Library

- **two3emp (68)**: EMP Formulation of Simple 2 x 2 x 2 General Equilibrium Model
- **airsp3 (69)**: Aircraft allocation - stochastic optimization
- **apl1psp (70)**: Stochastic Electric Power Expansion Planning Problem
- **apl1pcasp (71)**: Stochastic Electric Power Expansion Planning Problem
- **clearlaksp (72)**: Scenario Reduction: ClearLake exercise
- **farmsp (73)**: The Farmer's Problem - Stochastic
- **kilosafarm (74)**: Kilosa farm problem
- **landssp (75)**: Optimal Investment
- **nbContIndep (76)**: Newsboy problem, continuous and independent distribution
- **nbContJoint (77)**: Newsboy problem, continuous and joint distribution
- **nbDiscIndep (78)**: Newsboy problem, discrete and independent distribution
- **nbDiscJoint (79)**: Newsboy problem, discrete and joint distribution
- **portfolio (80)**: Stochastic portfolio model
- **prodsp3 (81)**: Stochastic Programming Example
- **simpleChance (82)**: Simple chance constraint model
- **sp3x2 (83)**: Simple stochastic model
- **tr20 (84)**: Extended transport model with stochastic demand and costs
- **nbsimple (85)**: Simple newsboy problem, discrete
- **airlift (86)**: Airlift operations schedule
- **stocfor3 (87)**: Long Range Forest Planning
- **circlesp (88)**: Circle Enclosing Points - Stochastic Example
- **batchsp (89)**: Design of batch chemical plants with stochastic demand and price
- **cargonet (90)**: Cargo network scheduling with stochastic transportation demand
- **gen2s (91)**: Two stage stochastic program in the generic form
- **sku1sp (92)**: Multi-product assemble model with discrete and Poisson demand distribution

GAMS Model Library

- **cubesoln (371)**: Three-dimensional Noughts and Crosses Multiple Solutions
- **sipres (372)**: Global optimization of semi-infinite programs via restriction of the right-hand side
- **cclinpts (373)**: Finding Optimal Breakpoints when linearizing a power utility function
- **ptsp (374)**: Traveling Salesman Problem Instance solved with explicit Permutation Enumeration
- **pmelody (375)**: Choose notes for melodic lines and chords with permutations
- **flowshop (376)**: Flow shop scheduling

GAMS Test Library

- **nlcode6 (534)**: Test for NL code bug from Dist 23.6
- **scensol3 (535)**: NoMatchLimit GUSS Test
- **gzip01 (536)**: Test gzipped input files
- **sl4qcp01 (537)**: Test of correctness for levels & marginals of QCP, with solvelink=1,2,4,5
- **sl4qcp02 (538)**: Test modsolstat & solution correctness - multiple QCons, with solvelink=1,2,4,5
- **sl4qcp03 (539)**: Test case for cancellation in quadratic terms, with solvelink=1,2,4,5
- **miqcp01 (540)**: Test of correctness for levels & marginals of MIQCP
- **indic01 (541)**: Test of =g= indicator constraints
- **indic02 (542)**: Test of =e= indicator constraints
- **kestrel1 (543)**: Kestrel test for lp solvers
- **kestrel2 (544)**: Kestrel test for mcp solvers
- **kestrel3 (545)**: Kestrel test for nlp solvers
- **kestrel4 (546)**: Kestrel test for minlp solvers
- **indic03 (547)**: Test of =e= indicator constraints
- **xpress03 (548)**: XPRESS test suite - solution pool example
- **mip05 (549)**: Maximum queens chess problem
- **xpress04 (550)**: XPRESS test suite - solution enumerator example
- **circlen (551)**: Circle Enclosing Points n dimensional
- **trilib04 (552)**: Demonstrates problems with a stateful function library
- **mip06 (553)**: Cuts and solution enumeration
- **xpress05 (554)**: XPRESS test suite - solution enumerator example
- **qcp09 (555)**: Simplest test for QCP correctness
- **perm1 (556)**: Test for various permutations
- `ctrlcmip (557)`: Test interrupt handling *[not available anymore]*

3.37.1.7 Solver/Platform Availability Matrix

3.37.2 23.8.2 Maintenance release (April 05, 2012)

- Bugfix for models with `=X=` rows and MCP (i.e. MPSGE)
- GloMIQO resets `OptCR=0` properly to 1e-9 and accepts now Ctrl-C on Windows
- Knitro handles MINLP failures better
- XPRESS handles `=E=` quadratic constraints better now
- Changed position of Library Form in GAMSIDE (avoid hang in corner to corner screen setup)
- Model `Trnsxcll` added to GAMS Data Library
- Model `ps5_s_mn` added to GAMS Model Library

3.38 23.7 Distribution

3.38.1 23.7.1 Major release (July 14, 2011)

3.38.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Michael Ferris, Josef Kallrath, Andreas Lundell, Renger van Nieuwkoop, Sabine Ritter, Stefan Vigerske, and Kent Zhao.

3.38.1.2 Platforms

- The 32bit and 64bit Linux systems are now built using the [GNU Compiler Collection (GCC)] (http://gcc.gnu.org/) toolset.
- Dropped support of Windows 95, 98, ME, and 2000.

3.38.1.3 GAMS System

GAMS

Asynchronous Execution

For a long time GAMS supports calling executables via the `$call` and `Execute` in a synchronous way. The new release also supports asynchronous job handling. This means you can start a job without waiting for the result. You can continue in your model and collect the return code of the job later. There are three ways to start a job asynchronously:

- `$Call.ASync ...` (compile time)
- `Execute.ASync '...';` (execution time)
- `put_utility fx 'Exec.ASync' / '...'; / put_utility fx 'Shell.ASync' / '...';` (execution time)

After each of those the function `JobHandle` can be used to get the Process ID (pid) of the last job started. With `JobStatus(pid)` one could check for the status of a job. Possible return values are:

- 0: error (input is not a valid PID or access is denied)
- 1: process is still running
- 2: process is finished with return code which could be accessed by errorlevel
- 3: process not running anymore or was never running, no return code available
- With `JobTerminate(pid)` a interrupt signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.
- With `JobKill(pid)` a kill signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.
- The model **asynexec** from the GAMS Test Library demonstrates the use of this new feature.

Function Libraries

The new GAMS Function Library Facility allows users to import functions from an external library into a GAMS model. Apart from the import syntax, the imported functions can be used in the same way as intrinsic functions. In particular, they can be used in equation definitions. Some function libraries are included with the standard GAMS software distribution but GAMS users can also create their own libraries using an open programming interface. Simple examples in the programming languages C, Delphi and Fortran come with every GAMS system.

Function libraries are made available to a model using the compiler directive:

```
$FuncLibIn <InternalLibName> <ExternalLibName>
```

Similar to sets, parameters, variables, and equations, functions must be declared before they can be used:

```
Function <InternalFuncName> /<InternalLibName>.<FuncName>/;
```

There are a few libraries which come ready to use with the GAMS system:

- fitfclib: packages FITPACK from P. Dierckx in a way that it works with this facility
- pwpcclib: evaluates piecewise polynomial functions
- stodclib: provides random deviates, probability density functions, cumulative density functions and inverse cumulative density functions for certain distributions
- tricclib, tridclib, trifclib: these simple examples can be found compiled and as source code written in C, Delphi and Fortran respectively

Detailed information can be found in the GAMS User's Guide in chapter 6.3.

In addition, the following examples from the GAMS Test Library can be referred to:

- **stolib01**: Uses the stochastic library which comes with the GAMS system
- **trilib01**: Uses trigonometric function from a library written in C, comes with C source code
- **trilib02**: Uses trigonometric function from a library written in Delphi, comes with Delphi source code
- **trilib03**: Uses trigonometric function from a library written in Fortran, comes with Fortran source code
- **pwplib01**: Uses a library for piecewise polynomial functions which comes with the GAMS system
- **fitlib01**: Uses FITPACK from P. Dierckx which was packaged in a way that it works with this facility and comes with the GAMS system

If you need detailed instructions for connecting your library to GAMS please contact support@gams.com.

Other

- Dropped support for EBCDIC. Functions `ordebcdic` and `ordascii` are no longer available. `oldascii` should be replaced by `ord`.

Bug Fix

- We found and corrected a bug related to the internal organization of non-linear code. This bug was introduced with version 23.6 but fortunately only occurs in rare cases.

Utilities

GAMSIDE

- Version 23.6 made it more difficult to find the directory where GAMS is installed. In this release we reintroduce the EXECUTABLE entry in the GAMSINI file to make this easier.
- The PageWidth option was limited to 255
- Increased the maximum number of jobs that can be run at he same time from 5 to 20.

GAMSINST

- The gamsinst program allows to set the default solvers and to install contributed software packages (e.g. Tom Rutherford's [GAMS Programming Tools] (http://www.mpsge.org/inclib/tools.htm)). Previous versions of gamsinst supported the installation of contributed **solver** packages. The new version of gamsinst does not support these types of solver packages anymore. If you need to install a contributed solver, please contact support@gams.com.

GDXXRW

- Fixed a memory leak that could make it difficult to read large spreadsheets.
- Worked around a limitation to read large spreadsheets by reading the data in multiple slices.
- The option NAin allows a string to be specified to recognize the value for NA when reading a spreadsheet; this is in addition to recognizing the string 'NA'.
- Undf is now recognized as a value without generating an error when reading from a spreadsheet so the entry can be processed in the GAMS model using the $ONUNDF dollar option in GAMS.
- Strings that are not recognized as a special value are no longer read as NA but as Undf instead; use the $ONUNDF in GAMS to read these values.

3.38.1.4 Solvers

AlphaECP

- Update to version 2.04.01
- New options
  - `CUTdelcrit` Improved heuristic to reselect cuts.
  - `CUTnrcuts` Option that the algorithm decides was added.
  - `ECPdumpsol` Write encountered solutions in GDX files.
  - `MIPsolstrat` Strategy for multiple MIP solutions
  - `NLPcall` Option that the algorithm decides was added.
  - `solvelink` Determines the way the NLP and MIP solvers are called. By default AlphaEXP now calls the NLP and MIP solver in memory.
- Deprecated options
  - `CUTautogrowth`, `CUTcheckgrowth`, `CUTgrowth`, `CUTgrowtheq`, `CUTgrowthnr`, `CUTmincuts`
  - `ECPcheckviol`
  - `NLPepsilon`, `NLPlimepsilon`

Baron

- New libraries 9.3
- Compared to previous versions, BARON 9.3 comes with improved multilinear relaxations and improved branching
- Support for the LP solvers XA and OSL has been dropped

Conopt

- New libraries 3.14Y

Coin-OR

- Updated Bonmin and Couenne libraries (bugfixes).
- On Linux and Windows, Ipopt now uses a multithreaded linear algebra library. The number of threads can be set via the GAMS option THREADS.
- Added Bonmin option "print_funceval_statistics" to enable statistics on number of function evaluations during a Bonmin run.
- Added Ipopt, Bonmin, and Couenne option "print_eval_error" to enable printing of information about function evaluation errors into the listing file.
- OsiCplex, OsiGurobi, and OsiXpress now support using the variable level values as initial solution for a MIP solve. This is enabled by setting the GAMS option INTEGER4 to 1.

Cplex

- New libraries 12.3
- Cplex 12.3 solves a greater variety of nonconvex quadratic programming models; that is, QP models with a quadratic term in the objective function. These models are also known as indefinite QPs (in contrast to positive or negative semi-definite QPs). In fact, Cplex 12.3 can find solutions that satisfy Karush-Kuhn-Tucker (KKT) conditions for certain indefinite QPs. A new parameter, solution target type (`SolutionTarget`), lets you specify to Cplex whether you will accept a solution that satisfies first-order optimality conditions (in contrast to accepting only globally optimal solutions), and Cplex computes and searches accordingly.
- Log files improved. You may observe slight differences from previous versions in the format of logs produced by Cplex 12.3. For example, the name of the column formerly titled "best node" in logs of MIP optimizations is now titled "best bound" to reflect more accurately the data recorded there.
- Supports in-core communication (`solvelink=5`)

DEA

- The solver DEA was dropped from our distribution.
- Data Envelopment Analysis (DEA) models can now be solved in GAMS via the Gather-Update-Solve-Scatter (GUSS), for more information see GAMS/DEA modeling

Dicopt

- New option
  - `solvelink` Determines the way the NLP and MIP solvers are called. By default Dicopt now calls the NLP and MIP solver in memory.

Gurobi

- New libraries 4.5.1
- Now available on AIX
- New default Method for continuous models: The new version uses a new Automatic setting as the default for solving continuous models. In previous releases, continuous models were solved with the dual simplex method by default. While the exact strategy used by the new Automatic setting may change in future releases, in this release the new approach uses the concurrent optimizer for continuous models with a linear objective (LPs), the barrier optimizer for continuous models with a quadratic objective (QPs), and the dual simplex optimizer for the root node of a MIP model. You should change the Method parameter if you would like to choose a different method.
- New Minimum Relaxation heuristic: The new version contains a new Minimum Relaxation heuristic that can be useful for finding solutions to MIP models where other strategies fail to find feasible solutions in a reasonable amount of time. Use the new MinRelNodes parameter to control this new heuristic.
- New branch direction control: The new version allows more control over how the branch-and-cut tree is explored. Specifically, when a node in the MIP search is completed and two child nodes, corresponding to the down branch and the up branch are created, the new BranchDir parameter allows you to determine whether the MIP solver will explore the down branch first, the up branch first, or whether it will choose the next node based on a heuristic determination of which sub-tree appears more promising.
- Cut pass limit: The new version allows you to limit the of cut passes performed during root cut generation in MIP. Use the new CutPasses parameter.

Gather-Update-Solve-Scatter (GUSS)

- The purpose of this new Gather-Update-Solve-Scatter manager or short GUSS is to provide syntax at the GAMS modeling level that makes an instance of a problem and allows the modeler limited access to treat that instance as an object, and to update portions of it iteratively. Detailed documentation of this facility is part of the Solver Manual (chapter GUSS).
- Solving MCP and CNS models with GUSS is an experimental feature. GAMS checks the consistency of MCP and CNS model, these checks are currently disabled when models of these types are solved in the GUSS framework.

Jams

- Supports `logic equations`
  - Logic equations can contain
    - binary variables
    - expressions that evaluate to constants
    - Boolean operators (`AND`,`OR`,`XOR`,`NOT`,`IMP(&rarr;)`,`EQV(&hArr;)`)

  - Logic equations can be used in conjunction with disjunctions
  - A version of the food model demonstrates the use of this new feature **foodemp (59)**
- Supports coupling and indicators through EMP's disjunction syntax
  - Indicator `disjunction b e` says that constraint e has to hold if b is true
  - Coupling `disjunction b e else not e` says that constraint e can only hold if and only if b is true
  - Indicators and Couplings can be reformulated using BigM, Convex Hull or CPLEX indicator constraints

LindoGlobal

- New libraries Lindo 6.1.1.588
- New Option `checkrange`: Calculates the feasible range for every variable in each equation while all other variables are fixed to their level. If set, the value of this option defines the name of the GDX file where the results are written to.

LogMIP

- The former LogMIP solvers `lmbigm` and `lmchull` are combined in the new solver solver `logmip`.
- LogMIP now uses the EMP syntax and modeltype.
- The LogMIP examples in the GAMS Model Library were revised.

Minos

- Supports in-core communication (`solvelink=5`)

Mosek

- New libraries 6.0 Rev 114

Msnlp/Oqnlp

- New option
  - `solvelink` determines the way the NLP and MIP solvers are called

Oslse

- The solver Oslse was dropped. Oslse is now an alias to osl.

Path

- Supports in-core communication (`solvelink=5`)

SBB

- New option
  - `solvelink` determines the way the NLP solver is called. By default SBB now calls the NLP solver in memory.

Scip/Soplex

- Dropped support for LPs/RMIPs in Gams/SCIP interface.
- Added new solver OsiSoplex for solving LPs with SoPlex (via COIN-OR/OSI interface).

Snopt

- Supports in-core communication (`solvelink=5`)

Xpress

- New libraries 22.01
- Improved performance through dynamic synchronization in the deterministic concurrent solver
- The stability and performance of the barrier code has been improved
- The default performance of the simplex algorithm has been improved on some models
- Improved zero-half cuts and aggregated cuts for MIPs.
- Tuned some of the automatic strategies for cutting and branching in MIP solves.
- Threads during a MIP solve will now busy-wait by default instead of going to sleep when waiting for work. This is to overcome a performance issue with modern speed-stepping CPUs, which might step down to a lower clock frequency when the load is less than 100%. This feature can be toggled using the SLEEPONTHREADWAIT control.

3.38.1.5 Model Libraries

GAMS Data Library

- **datatest (80)**: Driver for datalib tests of all sorts

GAMS EMP Library

- **foodemp (59)**: Food Manufacturing Problem - Blending of oils
- **simenlp (60)**: Simple example of ENLP
- **nlp2mcp (61)**: nlp2mcp: Form the KKT conditions of an NLP as an MCP
- **simplevi3 (62)**: Another simple VI example
- **simpequil2 (63)**: Simple Equilibrium with external constraint
- **hark-monop (64)**: SPE model from Harker - monopolist and competetive versions
- **hark-oligop (65)**: SPE model from Harker - oligopoly version
- **hark-oligop-ind (66)**: SPE model from Harker - oligopoly version
- **hark-stack (67)**: SPE model from Harker - Stackelberg version

GAMS Model Library

- **kqkpsdp (355)**: SDP Convexifications of the Cardinality Constraint Quadratic Knapsack Problem
- **prodplan (356)**: A Production Planning Example
- **sddp (357)**: Multi-stage Stochastic Water Reservoir Model solved with SDDP
- **ps2_f_s (358)**: Parts Supply Problem w/ 2 Types w/o & w/ Asymmetric Information
- **ps2_f (359)**: Parts Supply Problem w/ 2 Types w/o Asymmetric Information
- **ps2_f_eff (360)**: Parts Supply Problem w/ Efficient Type w/o Asymmetric Information
- **ps2_f_inf (361)**: Parts Supply Problem w/ Inefficient Type w/o Asymmetric Information
- **ps2_s (362)**: Parts Supply Problem w/ 2 Types w/ Asymmetric Information
- **ps3_f (363)**: Parts Supply Problem w/ 3 Types w/o Asymmetric Information
- **ps3_s (364)**: Parts Supply Problem w/ 3 Types w/ Asymmetric Information
- **ps3_s_gic (365)**: Parts Supply Problem w/ 3 Types w/ Global Incentive Comp. Const.
- **ps3_s_mn (366)**: Parts Supply Problem w/ 3 Types w/ Monotonicity Constraint
- **ps3_s_scp (367)**: Parts Supply Problem w/ 3 Types w/o & w/ SCP
- **ps10_s (368)**: Parts Supply Problem w/ 10 Types & w/ Asymmetric Information
- **ps10_s_mn (369)**: Parts Supply Problem w/ 10 Types w/ Random p(i)
- **ccoil (370)**: Oil Pipeline Design Problem using concurrent MIP solves

GAMS Test Library

- **interval (506)**: Test for interval evaluation
- **empdisj3 (507)**: Test Disjunctions using not (equivalence)
- **emplog1 (508)**: Test disjunctions and logical conditions
- **maxima (509)**: Test several maxima of Cmex
- **emplog2 (510)**: Test simple logical conditions
- **qcp07 (511)**: Test correctness for QCP with poly function
- **qcp08 (512)**: Test correctness for QCP with power($*$,2) function
- **empdisj4 (513)**: Test disjunctions with negative bounds on variables
- **funcback (514)**: Test function backward compatibility
- **asynexec (515)**: Test asynchronous execution at compile and execution time
- **scen01 (516)**: Compile time test for the scenario facility
- **empdisj5 (517)**: Test disjunctions using not on binary variable and constraint
- **scen02 (518)**: Execution time test for the scenario facility
- **scen03 (519)**: Multi-dimensional scenario solver test
- **stolib01 (520)**: Test extrinsic functions in stodclib
- **trilib01 (521)**: Test extrinsic functions in tricclib

- **trilib02 (522)**: Test extrinsic functions in tridclib
- **trilib03 (523)**: Test extrinsic functions in trifclib
- **lindorg2 (524)**: Test for LindoGlobals CheckRange Option
- **mip04 (525)**: Exercise new XPRESS return code for unbounded MIP
- **load5 (526)**: Tests UNDF with $load
- **pwplib01 (527)**: Test piecewise polynomials in pwpcclib
- **fitlib01 (528)**: Test the use of FITPACK inside GAMS
- **load6 (529)**: Tests dynamic error messages during $loaddc
- **fnspowx (530)**: Test correctness of signpower intrinsic
- **fnspown (531)**: Test correctness of signpower intrinsic
- **nlcode5 (532)**: Test for NL code bug from Dist 23.6
- **scensol2 (533)**: Test handling of unsorted scenario UELs in GUSS

3.38.1.6 Solver/Platform Availability Matrix

3.38.2 23.7.2 Maintenance release (July 22, 2011)

- Alphaecp: Alphaecp v2.04.02
- Conopt3: Fixed a system error 2003
- Dicopt: Fixed a bug for accidentally determined crossovers
- Inclib: `put_reorderit.gms` update from Bruce McCarl
- Cbc: Disabled preprocessing in case of semicont/semiint variables, it seems buggy
- Cplex: Does no longer require finite bounds on semicont variables
- Gamsinst: Fix problem with blanks in sysdir and IDE error when not having a license

3.38.3 23.7.3 Maintenance release (August 23, 2011)

- alphaecp: fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
- apifiles:
  - get rid of warnings in cc interface,
  - fix problem with compilation of fat binaries on Darwin in cc interface
  - fix error in `gdxSymbolGetDomain` in Python interface
  - fix errors with constant definitions in C# interface
  - add constant definitions and additional constructor (which gets a handle) in Java interfaces
- conopt3: Version 3.15A maintenance release
- cplex:
  - fix for interrupt (Ctrl-C) when running with `solvelink=5`
  - fix memory leak

- dicopt:
  - fix reporting of `objest` in case of a crossover
  - fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
- gamscmex: fix bug related to the internal organization of non-linear code, affects two-argument functions in rare cases
- gamside:
  - close chart files too when changing system directory
  - GAMS project file name independent of casing
- gdxdump: suppress on/off empty when writing single symbol
- gdxmrw: was missing on 64bit Linux
- grid: fix potential problem with scaling
- kestrel: added Xpress support
- minos fix: for interrupt (Ctrl-C)
- sbb: fix problem with accumulating solving times of subsolvers which might lead to an earlier stop
- snopt fix: for interrupt (Ctrl-C)
- xpress: update global search to use callbacks called during root node processing and improve scheme to stop XPRESS when gap is achieved or on user interrupt

3.39 23.6 Distribution

3.39.1 23.6.2 Major release (December 13, 2010)

3.39.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Nico Keyaerts, Leon Lasdon, Xiang Li, Timo Lohmann, Sebastian Ritter, Yannick Rombauts, Tom Rutherford, Uwe Schneider, Stefan Vigerske, Achim Wechsung, and Gyeongbeom Yi.

3.39.1.2 GAMS System

GAMS

- Added the function `RandBinomial(n,p)` which generates random numbers from the Binomial distribution where n is the number of trials and p the probability of success for each trial.
- Added the gams date and release functions to $eval and [..] constant evaluations: `jdate`, `jtime`, `gyear`, `gmonth`, `gday`, `gdow`, `gleap`, `ghour`, `gminute`, `gsecond`, `gmillisec`, `jstart`, `jnow`, `gamsversion`, `gamsrelease`.
- Modified operator precedence binding of `IMP` (->), `EQV` (<->), `OR` and `XOR`. These operator used to have the same precedence binding, so with the usual left to right evaluation an expression like `a -> b OR c` used to be equivalent to `(a -> b) OR c`. `IMP` and `EQV` have now a lower precedence binding, so the expression `a -> b OR c` is now equivalent to `a -> (b OR c)`.

- There is no longer a practical limit for the number of non-linear instructions for a single equation.
- New interpretation of value `default` for command line parameters `gdx` and `rf`: `default` is interpreted as the input file name, this means for example, that `gams trnsport gdx=default` will write the file `trnsport.gdx`. Along the same lines, `gams trnsport rf=default` will write the reference file `trnsport.ref`.
- New command line parameter `Threads`: Controls the number of threads to be used by a solver. It can also be set as GAMS option or model attribute. The model attribute has the highest priority, the command line parameter the lowest. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.

APIs

- Determine platform dependent library name automatically, no requirement for compiler flags anymore
- Delphi files also support Delphi 2010
- Python files comprehensively improved
- More Fortran examples

Utilities

Chk4Upd

- The new tool chk4upd checks if there is a newer version of GAMS available than the one you are running. It checks for the most recent version available as well as for the newest version you can run with your license in case your maintenance and support is expired. By default it reads gamslice.txt located next to it. Alternatively, you can specify another license on the command line. In addition to the command line tool you will find an integration in the GAMS IDE at Help -> Check for GAMS Update.

GamsIDE

- Integration of 'Check for GAMS Update' in the Help Menu
- The 'Find in Files' option now allows for one or more sub-directories to be excluded from the search
- The selection to use a different GAMS system has been changed. We no longer specify the gams.exe executable but define the alternate GAMS system directory instead.

GDXDiff

- The function to test if two values are different when Eps/RelEps has been specified was changed. The documentation describes the new implementation.

GDXMRW

- GDXMRW is a collection of utilities that make it easier and more reliable to share data between GAMS and Matlab and that allow the Matlab user to call GAMS using something similar to a subroutine interface. Formerly contributed software, these utilities were integrated into GAMS Distribution 23.4 and have been updated and improved since then.

XLSDump

- This new program will write all worksheets of an Excel workbook to a gdx file. Unlike GDXXRW, the program does not require that Excel is installed. Windows platforms only.

3.39.1.3 Solvers

Baron

- Option ExtNLPsolver allows to specify an option file for the GAMS NLP solver chosen (e.g. `ExtNLPSolver conopt.1`)

Coin-OR

- New libraries
  - BONMIN 1.4
  - CSDP 6.1.1
  - CSDP now also available on Solaris x64 on Intel
- The old names COINCPLEX, COINGUROBI, COINMOSEK, and COINXPRESS are not available anymore. Please use the new names OSICPLEX, OSIGUROBI, OSIMOSEK, and OSIXPRESS introduced with distribution 23.4.

Conopt

- Supports in-core communication (`solvelink=5`)
- Dropped support of Conopt 2 (old version of Conopt)

Cplex

- New libraries 12.2.0.1

EMP

- The emp information file now supports block-wise definition
  - A flip operator (`-`) was introduced, see model **flipper (50)**

Gurobi

- New libraries 4.0
- Quadratic programming: The Gurobi Optimizer now supports models with quadratic objective functions. The new version includes primal simplex, dual simplex, and parallel barrier optimizers for continuous QP models, and a parallel branch-and-cut solver for Mixed Integer Quadratic Programming (MIQP) models.
- Concurrent optimizer: The Gurobi Optimizer now allows you to run multiple algorithms simultaneously when solving a linear continuous model on a multi-core machine. The optimizer returns when the first algorithm solves the model. We include both a standard concurrent optimizer and a deterministic concurrent optimizer. The latter returns the exact same solution every time you run it, while the former can sometimes return different optimal solutions from one run to the next. The former can sometimes be significantly faster.
- MIP performance: The MIP solver is faster in release 4.0. These improvements do not require any parameter changes.
- LP performance: The simplex and barrier solvers are slightly faster in release 4.0. We have also improved the numerical stability of the primal simplex solver and the barrier crossover algorithm.
- Delayed MIP strategy change: The Gurobi Optimizer now gives you the option to change a few MIP parameters in the middle of the optimization in order to dynamically shift the search strategy. Specifically, two new parameters, ImproveStartGap and ImproveStartTime, allow you to specify when the algorithm should modify the values of a few parameters that control the intensity of the MIP heuristics. By setting one or both of these parameters to non-default values, you can cause the MIP solver to switch from its standard parameter settings, where it tries to strike a balance between finding better solutions and proving that the current solution is optimal, to a set of parameter values that focus entirely on finding better solutions.
- New approach to choosing the LP algorithm: The functions previously controlled by parameters LPMethod and RootMethod are now controlled by parameters Method and NodeMethod. Use Method to choose the algorithm for solving a continuous model, or for solving the root relaxation in a MIP model. Use NodeMethod to choose the algorithm for solving node relaxations in a MIP model.
- Deprecated options: `LPMethod` and `RootMethod`
- New options
  - `Method`: Controls the algorithm used to solve continuous linear and quadratic models. This parameter also selects the algorithm used to solve the root node of a MIP model.
  - `NodeMethod`: Chooses the algorithm used to solve node relaxations in a MIP model.
  - `ModKCuts`: Controls the generation of mod-k cuts.
  - `ImproveStartGap`: Allows you to specify the optimality gap at which the MIP solver resets a few MIP heuristics parameters in order to shift the attention of the MIP solver to finding the best possible feasible solution.
  - `ImproveStartTime`: Allows you to specify the elapsed time at which the MIP solver resets a few MIP heuristics parameters in order to shift the attention of the MIP solver to finding the best possible feasible solution.
  - `PreMIQPMethod`: Chooses the presolve transformation performed on MIQP models.
  - `PSDTol`: Sets a limit on the amount of diagonal perturbation that the optimizer is allowed to do on the Q matrix for a quadratic model. If a larger perturbation is required, the optimizer will terminate with a message that Q is not positive semi-definite.

Knitro

- New libraries 7.0.0
- Supports the MultiStart feature in Knitro
- Supports in-core communication (`solvelink=5`)
- Supported on 64bit Mac and 64bit Solaris on Intel
- Dropped support on Solaris Sparc and Sparc64

Lgo

- supports in-core communication (`solvelink=5`)
- support of external equations

Mosek

- New libraries 6.0.96
- Supports in-core communication (`solvelink=5`)
- Dropped support on Solaris Sparc and Sparc64

Scip

- New libraries 2.0 ([more details] (https://www.scipopt.org/download/files/SCIP-release-notes-2.0))
- The GAMS/SCIP interface now also supports semicontinuous and semiinteger variables
- GAMS/SCIP now uses SoPlex 1.5.0 as LP solver

Xa

- Dropped support on Solaris Sparc and Sparc64

Xpress

- Supports in-core communication (`solvelink=5`)
- New libraries 21.01

3.39.1.4 Model Libraries

GAMS Data Library

- **GDXXRWExample15 (76)**: Write spreadsheet using a filter
- **GDXXRWExample16 (77)**: Write spreadsheet using text and hyperlinks
- **tompivot (78)**: Little GAMS Program from Tom Rutherford that Illustrates Report Generation with Excel
- **readdata (79)**: Read Data from .inc, .xls, .mdb and .csv file

GAMS EMP Library

- **flipper (50)**: Test of flipping equations
- **scarfemp-dem (51)**: Scarf's Activity Analysis Example
- **scarfemp-altdem (52)**: Scarf's Activity Analysis Example - non-closed form demand function
- **scarfemp-primal (53)**: Scarf's Activity Analysis Example
- **scarfemp-dual (54)**: Scarf's Activity Analysis Example
- **scarfexpend (55)**: Scarf's Activity Analysis Example
- **pies (56)**: PIES Energy Equilibrium
- **exc2x2emp (57)**: pure exchange model (ie no production)
- **exc2x2emp-dem (58)**: pure exchange model (ie no production)

GAMS Model Library

- **relief (353)**: Relief Mission
- **spatequ (354)**: Spatial Equilibrium

GAMS Test Library

- **emp09 (492)**: Test initial levels for equilibrium EMP models
- **xerr1 (493)**: External Function Errors: RHS wrong
- **xerr2 (494)**: External Function Errors: Aij wrong
- **cerr1 (495)**: Cone Equation Errors
- **merr1 (496)**: Matrix Errors
- **mip02 (497)**: Check on MIP solution value
- **mip03 (498)**: Test for zero gap when optcr and optca set to zero
- **emp10 (499)**: Test of EMP equilibrium models and flip operator
- **emp11 (500)**: Test EMP formulations of scarfmcp
- **empdisj2 (501)**: Test disjunctions involving the objective
- **gdxxrw6 (502)**: Test for dset reading problem
- **qcp06 (503)**: Nonlinear model cannot be solved as QCP
- **gdxdump1 (504)**: Use GdxDump NoData on Transportation Problem
- **lindorng (505)**: Test for LindoGlobal's CheckRange Option

3.39.1.5 Solver/Platform Availability Matrix

3.39.2 23.6.3 Maintenance release (February 15, 2011)

- Cmex: Fix for variable level projection
- Bonmin: Several fixes for outer-approximation based algorithms
- Cplex: New version 12.2.0.2
- Gurobi: New version 4.0.1
- Knitro: Documentation update
- Scip: New version 2.0.1
- Xpress: Fix for crash on 64 bit Solaris
- McCarl Guide: Updates for 23.6
- Gdxxrw: Includes system error message when we cannot start Excel
- Gdxmrw: Minor bugfixes

3.39.3 23.6.4 Maintenance release (April 01, 2011)

- CMEX: Maximal nested includes were raised to 40
- Mosek: New Version6 Rev 105
- Lindoglobal: New Version 6.1.1.553
- Conopt: New Version 3.14W
- Xpress: Memory leak fix
- Coin: Solver fixes
- JAMS: Updates for disjunctions
- GDXXRW: Fixes
- GDXMRW: Updates
- McCarl and GDXUtils: Documentation updates
- Model Libraries: Updates

3.39.4 23.6.5 Maintenance release (April 08, 2011)

- CMEX: Bug fix related to situations with very large number of labels in connection with a restart with continued compilation

3.40 23.5 Distribution

3.40.1 23.5.1 Major release (July 05, 2010)

3.40.1.1 Acknowledgments

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

3.40.1.2 GAMS System

APIs

- new VB.net example using GDX, Option and GAMS DLL: apifiles\examples\example2.vb
- combine Java api files in package com.gams.api, existing applications need to be adjusted when using the new Java api files, see e.g. apifiles\examples\example1.java:

```
package com.gams.example1;
import com.gams.api.*;
public class example1 {
...
```

3.40.1.3 Solvers

BARON

- New libraries 9.0.6

CONOPT

- New libraries 3.14U

CPLEX

- New libraries 12.2
- MIP kappa describes a new feature extending the computation of kappa to mixed integer programming (MIP). (In linear programming, kappa is also known as the condition number of a solution.) See GAMS/Cplex parameter `mipkappastats` for details.
- Intel processors prior to Pentium 4 are no longer supported.
- The MIP node log interval (GAMS/Cplex parameter `MIPInterval`) now accepts either a positive or negative integer value, making it easier for you to adjust the amount of information logged for your problem.
- The default value of the GAMS/Cplex `ClockType` parameter has changed from 0=automatic (let Cplex choose) to 2 (wallclock time).
- The concurrent optimizer launches distinct optimizers on multiple threads. Cplex now offers concurrent, deterministic optimization of linear programming models (LPs) when more than one thread is given to Cplex (GAMS/Cplex parameter `Threads`). Running concurrent optimizers simultaneously on multiple threads may consume more memory. If the memory reduction parameter (`MemoryEmphasis`) is turned on, then you have instructed CPLEX to conserve memory. In that case, Cplex does not run concurrent, deterministic optimization unless parameter `LPMethod` is explicitly set to 6.
- New parameter `AuxRootThreads`: Partitions the number of threads for Cplex to use for auxiliary tasks while it solves the root node of a problem. On a system that offers N processors or N global threads, if you set this parameter to n, where $N > n > 0$ then Cplex uses at most n threads for auxiliary tasks and at most N-n threads to solve the root node.

GUROBI

- New libraries 3.0.1
- Now also available on Mac X Intel 64bit

LINDOGLOBAL

- New libraries Lindo 6.1.1 for all platforms except for Solaris Sparc 32 bit and 64 bit
- Now also available on Mac X Intel 64bit

3.40.1.4 Model Libraries

GAMS Test Library

- **gen_r1 (483)**: Generate GDX file for test of GDXMRW
- **gen_r2 (484)**: Generate GDX file for test of GDXMRW
- **gen_rbig (485)**: Generate GDX file for test of GDXMRW
- **gdxmrw03 (486)**: test the Matlab utility rgdx
- **gdxmrw04 (487)**: test the Matlab utility wgdx
- **testinst (488)**: test the Matlab mex-file gams
- **gversion (489)**: test the Matlab mex-file gams
- **gdxmrw05 (490)**: test the Matlab mex-file gams
- **onmulti7 (491)**: Test merge with $onmulti and empty data statement

3.40.1.5 Solver/Platform Availability Matrix

3.40.2 23.5.2 Maintenance release (August 18, 2010)

- AlphaECP: Fix for crash running under IDE
- API: Updated Python and Java APIs and examples
- Baron: Fixed issue with output formatting under Linux
- Baron: Memory leak when using BARON with external NLP solver
- Gdxmerge: Do not use path when checking for `merged.gdx`
- Gurobi: Fixed typo for option displayinterval
- McCarl Guide: Updates for 23.5
- Mosek 6 rev 85 libraries
- MPS2GMS: Removed mps2gms debug message
- Snopt: Fix for wrong return code (local,global)
- Xlsdump utility added

3.41 23.4 Distribution

3.41.1 23.4.1 Major release (May 21, 2010)

3.41.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Phil Bishop, Stefan Boeters, Pavel Borisovski, Wolfgang Britz, Michael Ferris, David Grace, Sascha Herrmann, Alexander Mitsos, Sebastian Ritter, and Tom Rutherford. We also thank Marcel Roelofs for helpful and insightful comments and discussions on Hessian computations.

3.41.1.2 Platforms

AIX

- Now built on AIX 5.3
- 64 bit
- New AIX system no longer supports the following products BARON, CONOPT2, DECIS, OSL3, OSLSE, SCENRED, SCENRED2, and XA

Mac PowerPC

- Dropped support of the Macintosh PowerPC GAMS System. Version 23.3 and older are still available for download

3.41.1.3 GAMS System

GAMS

GridScript

- The GAMS parameter `gridscript` allows to point to a customized grid submission script.

TryLinear

- GAMS checks a model for non-linearities at compile time and requires the user to specify `using nlp` in the `solve` statement. At run time these non-linearities might disappear (cancelation, multiplication of non-linear terms with 0, ...), but the `using nlp` prevented the use of a pure LP solver. The new model attribute `<model>.TryLinear = 1` checks at run time whether a nonlinear model has any non-linearities and then calls the default/chosen solver for the resulting model type.
- Depending on the original model type the resulting model type is:
  - QCP/DNLP/NLP -> LP
  - MIQCP/MINLP -> MIP
  - RMIQCP/RMINLP -> RMIP
- An example was added to the Test Library: **trylin01 (482)**

Compile Time String Comparison

- A compile time equivalent of the predefined symbol `sameas` was added. E.g.

```
$eval a sameas(xyz,xYz)
$eval b sameas ( 'xyz' , "xyz" )
$eval c sameas (12-3,12-3)
$eval d sameas (12-3,13-4)
$log %a% %b% %c% %d%
```

will yield:

```
1 1 1 0
```

Compile Time Constants

For various GAMS options compile time constants were added in order to ease the reading code if used.

- **solPrint**:
  0 %solPrint.summary%
  1 %solPrint.report%
  2 %solPrint.quiet%
- **handleStatus**:
  0 %handleStatus.unknown%
  1 %handleStatus.running%
  2 %handleStatus.ready%
  3 %handleStatus.failure%
- **solveLink**:
  0 %solveLink.chainScript%
  1 %solveLink.callScript%
  2 %solveLink.callModule%
  3 %solveLink.asyncGrid%
  4 %solveLink.asyncSimulate%
  5 %solveLink.loadLibrary%
- **solveStat**:
  1 %solveStat.normalCompletion%
  2 %solveStat.iterationInterrupt%
  3 %solveStat.resourceInterrupt%
  4 %solveStat.terminatedBySolver%
  5 %solveStat.evaluationInterrupt%
  6 %solveStat.capabilityProblems%
  7 %solveStat.licensingProblems%
  8 %solveStat.userInterrupt%
  9 %solveStat.setupFailure%
  10 %solveStat.solverFailure%
  11 %solveStat.internalSolverFailure%
  12 %solveStat.solveProcessingSkipped%
  13 %solveStat.systemFailure%
- **modelStat**:
  1 %modelStat.optimal%
  2 %modelStat.locallyOptimal%
  3 %modelStat.unbounded%
  4 %modelStat.infeasible%

```
5 %modelStat.locallyInfeasible%
6 %modelStat.intermediateInfeasible%
7 %modelStat.intermediateNonoptimal%
8 %modelStat.integerSolution%
9 %modelStat.intermediateNonInteger%
10 %modelStat.integerInfeasible%
11 %modelStat.licensingProblem%
12 %modelStat.errorUnknown%
13 %modelStat.errorNoSolution%
14 %modelStat.noSolutionReturned%
15 %modelStat.solvedUnique%
16 %modelStat.solved%
17 %modelStat.solvedSingular%
18 %modelStat.unboundedNoSolution%
19 %modelStat.infeasibleNoSolution%
```

- E.g. the statments following are the same
  `modelname.solveLink=3; modelname.solPrint=0;`
  `modelname.solveLink=%solveLink.asyncGrid%; modelname.solPrint=%solPrint.summary%;`
- The GAMS model libraries were adjusted and now make use of these constants.

APIs

- new C# example using GDX API: apifiles\examples\example1.cs
- added C++, C# and VB.net project files to make it easier to compile the examples in Visual Studio
- added Delphi Option and Project Configuration Files to make it easier to compile the examples in the Delphi IDE and on the command line
- added Java Native Interface libraries
- `apifiles\common`: dropped file gamsglobals.h, use gclgms.h instead

GDXDCLIB API

- New entry points:
- `gdxOpenAppend` To add symbols to an existing GDX container
- `gdxDataReadRawFast` To read data in raw mode using a callback function

Data Utilities

GDX2ACCESS

- `gdx2access` now supports saving the text associated with set entries in the database.
- Added parameter to indicate which version of the database should be created (.mdb or .accdb)

GDXXRW

- `gdxxrw` now supports writing an Excel file with filters. Filtering can be switched on/off for the symbols that follow the filter option:

```
execute 'gdxxrw.exe test.gdx par=A rdim=3 cdim=0 rng=sheet1!a1 filter=1 par=B rdim=3 cdim=0 rng=
```

- Added options to write texts and hyperlinks to a spreadsheet.

MDB2GMS

- `mdb2gms` now supports writing of the text associated with set entries.

SQL2GMS

- `sql2gms` now supports writing of the text associated with set entries.

EMP

- A new model type `EMP` was added in order to provide the flexibility required for the extended mathematical programming framework.
- The former solver EMP was renamed to JAMS to avoid confusion. JAMS is the default solver for EMP models.
- EMP bilevel programming now supports Variational Inequality (VI) followers in addition to maximization/minimization followers. Example: **multmpec (25)**
- EMP now supports equilibrium models. In contrast to bilevel programs these agent-based systems don't require a leader. Example: **transeql (45)**
- Lots of EMP models were added to the GAMS EMP Library.

IDE

- Fixed bug with writing a symbol with many elements to a spreadsheet
- Opening many .lst files is faster

Libinclude

- McCarl's `put_toexcel` and `put_tohtml` were added to inclib subdirectory of the GAMS system directory and therefore can be used using `$libinclude`. These tools allow greater control when writing multi dimensional symbols to Excel or files. More information can be found here

3.41.1.4 Solvers

BARON

- New Libraries 9.0.5
- Native 64bit Baron libraries in 64bit windows and linux system
- New option `ExtNLPsolver` allows to call any GAMS NLP Solver available
- Dropped support of Baron on platform AIX

Coin-OR

- Renaming
  - of solver links as follows
    - COINBONMIN -> BONMIN
    - COINCBC -> CBC
    - COINCOUENNE -> COUENNE
    - COINGLPK -> GLPK
    - COINIPOPT -> IPOPT
    - COINOS -> OS
    - COINSCIP -> SCIP
  - of bare bone solver links as follows
    - COINCPLEX -> OSICPLEX
    - COINGUROBI -> OSIGUROBI
    - COINMOSEK -> OSIMOSEK
    - COINXPRESS -> OSIXPRESS
  - For convenience of our users we keep the old names as aliases to the new names for the next distribution.
  - Note: The temporary names COINBONMIND, COINCBCD, and COINIPOPTD are not available anymore
- New libraries
  - CBC 2.4
  - Bonmin 1.3
  - Couenne 0.3
  - Ipopt 3.8
  - GLPK 4.43
  - Mumps 4.9 (used by Ipopt and Bonmin)
  - OS 2.1
- The new BONMIN version brings various new MINLP heuristics (Feasibility pump, diving based heuristics, RINS, local branching).
- BONMIN can now use CPLEX as solver for sub-MIPs, see option `milp_solver`
- SCIP now supports convex and nonconvex quadratic constraints (model types QCP, RMIQCP, MIQCP).
- BCH has temporarily been disabled for CBC, BONMIN, and SCIP.
- The OS link now supports only remote solvers via an Optimization Services Server.
- Native CSDP executables were added to the 64bit Windows and Linux system.

CPLEX

- Updated threads option allows specification of cores left free for non-CPLEX work

GUROBI

- New libraries GUROBI 3.0
- New parallel barrier solver
- New MIP features include symmetry handling, improved and additional cutting planes, and additional heuristics.
- Alternate MIP solutions: retrieve all of the feasible solutions found during the branch-and-cut search
- New parameters
  - AggFill: provides finer-grain control of presolve aggregation.
  - BarConvTol: barrier convergence tolerance.
  - BarCorrectors: controls central corrections in barrier.
  - BarIterLimit: limits the number of barrier iterations.
  - BarOrder: controls the fill-reducing ordering in barrier.
  - PreDepRow: controls the presolve dependent row reduction.
  - Crossover: controls barrier crossover.
  - CrossoverBasis: controls the generation of the initial crossover basis.
  - MIPFocus: allows you to modify the MIP solution strategy to better suit the needs of different model types.
  - MIPGapAbs: absolute MIP termination tolerance (GAMS OptCA)
  - NetworkCuts: controls the generation of network cutting planes.
  - PreDual: determines whether presolve should form the dual of the input model.
  - PrePasses: controls the number of passes performed by presolve.
  - PumpPasses: controls the feasibility pump heuristic.
  - RINS: controls the RINS heuristic.
  - Symmetry: controls the new MIP symmetry handling.
  - SubMIPCuts: controls the generation of sub-MIP cutting planes.
  - Threads: allows specification of how many cores to use.

LINDOGLOBAL

- New libraries Lindo 6.0.1.406
- Now also available on Solaris on x64
- Significant improved performance with some models when Mosek is licensed as well

MOSEK

- New libraries MOSEK 6 rev 71

XPRESS

- New libraries XPRESS 20.00

In-core communication solver links

- All Coin-OR solvers are now supported as in-core communication solvers.

3.41.1.5 Model Libraries

GAMS Data Library

- **tsvngdx (75)**: Support GDX Files with `TortoiseSVN` diff

GAMS EMP Library

- **jointc1 (22)**: Educational bilevel model
- **jointc2 (23)**: Educational bilevel model
- **ferris43 (24)**: Educational embedded complementarity system model
- **multmpec (25)**: Educational bilevel model with VI followers
- **transbp (26)**: Transportation model with variable demand function using bilevel programming
- **flds911 (27)**: Princeton Bilevel Optimization Example 9.1.1
- **flds912 (28)**: Princeton Bilevel Optimization Example 9.1.2
- **flds913 (29)**: Princeton Bilevel Optimization Example 9.1.3
- **flds914 (30)**: Princeton Bilevel Optimization Example 9.1.4
- **flds915 (31)**: Princeton Bilevel Optimization Example 9.1.5
- **flds916 (32)**: Princeton Bilevel Optimization Example 9.1.6
- **flds917 (33)**: Princeton Bilevel Optimization Example 9.1.7
- **flds918 (34)**: Princeton Bilevel Optimization Example 9.1.8
- **flds919 (35)**: Princeton Bilevel Optimization Example 9.1.9
- **flds921 (36)**: Princeton Bilevel Optimization Example 9.2.1
- **flds922 (37)**: Princeton Bilevel Optimization Example 9.2.2
- **flds923 (38)**: Princeton Bilevel Optimization Example 9.2.3
- **flds924 (39)**: Princeton Bilevel Optimization Example 9.2.4
- **flds925 (40)**: Princeton Bilevel Optimization Example 9.2.5
- **flds926 (41)**: Princeton Bilevel Optimization Example 9.2.6
- **flds927 (42)**: Princeton Bilevel Optimization Example 9.2.7
- **flds928 (43)**: Princeton Bilevel Optimization Example 9.2.8
- **flds929 (44)**: Princeton Bilevel Optimization Example 9.2.9
- **transeql (45)**: Transportation model as equilibrium problem
- **simplevi (46)**: Simple Variational Inequality
- **simplevi2 (47)**: Simple Nonlinear Variational Inequality
- **affinevi (48)**: Affine Variational Inequality
- **simpequil (49)**: Simple Equilibrium

GAMS Model Library

- **trnspwl (351)**: A Transportation Problem with discretized economies of scale
- **food (352)**: Food Manufacturing Problem - Blending of oils

GAMS Test Library

- **examin04 (464)**: EXAMINER test suite - test returnGamsPoint option and QCP
- **empbp04 (465)**: Bilevel model with and without explicitly defined objective equation
- **empbp05 (466)**: Bilevel model with MIN follower vs. VI follower
- **eval05 (467)**: constant expression test for ceil,floor,trunc,frac
- **eval06 (468)**: matching operators in a column
- **emp06 (469)**: Test of EMP based on trnsport model
- **emp07 (470)**: Test of EMP based on trnsport model
- **emp08 (471)**: Test of EMP based on trnsport model
- **pgams01 (472)**: Test procdir deletion in pgams
- **pgams02 (473)**: Test procdir deletion in pgams
- **tabsubst (474)**: Tab and string substitution for long line
- **eval07 (475)**: Test evaluation of real constants - string2Double conversion
- **dumpsol (476)**: Gurobi Alternate Solutions for a Simple Facility Location Problem
- **ifthen5 (477)**: $ifthen false without sameline
- **ifthen6 (478)**: $ifthen/elseif false without sameline
- **utils02 (479)**: test MPS2GMS - it had range problems
- **utils03 (480)**: test MPS2GMS for reading the second range entry on a line
- **mpsge11 (481)**: MPSGE test - multiple fixed income levels
- **trylin01 (482)**: Test model attribute tryLinear

3.41.1.6 Solver/Platform Availability Matrix

3.41.2 23.4.3 Maintenance release (May 24, 2010)

3.42 23.3 Distribution

3.42.1 23.3.1 Major release (November 01, 2009)

3.42.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Wietse Dol, Daniel T. Fokum, Nuri Gokhan, Iiro Harjunkoski, Josef Kallrath, Lloyd R. Kelly, Kristina Konold, and Uwe Schneider.

3.42.1.2 GAMS System

GAMS

- Assigning members to a set using the asterisk is now possible in decreasing order as well. For example, the following are valid set statements in GAMS.

```
Set years /bc2000*bc1,0*2009/;
Set years /"-20"*"-1"/;
```

- The GAMS parameters `ProcDir` and `MaxProcDir` can be used to control the generation of process directories. `ProcDir=abc` will use abc instead of the 225. The user is responsible to create and remove the directories. `gams.procdir%` will now be defined and give you the actual process directory in use. `MaxProcDir=100` will extend the usual 225a to 225z, 225aa, 225ab, etc. The defaults will be 26 for `MaxProcDir` and 225? for `ProcDir` to make everything work as before.
- The GAMS parameter `RunDir` was removed.
- The Dollar Control Options `$hiddencall` does the same as `$call` but makes sure that the statement is neither shown on the log nor the listing file. This is also true in case `$ondollar` or `dp=2` is used. `$hiddencall` is especially useful in case of an encrypted model that e.g. reads/writes from an password-protected Excel file using `gdxxrw` password option.

GDXXRW

- Support of the Excel 2007 file extension xlsm
- The option `password` allows to pass down a password to a protect Excel file

3.42.1.3 Solvers

Baron

- New libraries: version 9.0
- Conopt can be used to solve nonlinear subproblems, but on 32bit platforms only (NLPSol=6)
- More control over common linear subexpressions

Coin-OR

- CoinScip version 1.2
  - The multi-commodity-flow network cut separator is now used by default
  - Improvements in heuristics, presolving, and branching
- CoinGlpk version 4.39
- CoinCouenne version 0.2
  - A separator for disjunctive cuts has been added
- CoinCplex, CoinGurobi, CoinMosek, CoinXpress
  - These new *bare bone* solver links come free of charge with the GAMS Base system. General GAMS options (reslim, optcr, nodlim, iterlim) are supported. In addition an option file in the format required by the solver can be provided.
- CoinOS 2.0
  - The new experimental link to the Optimization Services project allows you to convert instances of GAMS models into the OS instance language (OSiL) format and to let an Optimization Services Server solve your instances remotely.

GUROBI

- New libraries: version 2.0
- Simplex performance: The simplex optimizers are much faster in the release. The improvements are most pronounced in the dual simplex method.
- MIP performance: The MIP solver is significantly faster as well. Part of this is a consequence of the increased speed of the dual simplex optimizer, and part is due to algorithmic improvements in the MIP itself.
- MIP : You can now compute an Irreducible Inconsistent Subsystem ( ) for an infeasible MIP model. The previous release could only compute IISs for continuous models.
- MIP node files: You can now store search tree nodes on disk. This allows you to solve much larger and more difficult MIP models. Use the new NodefileStart parameter to indicate how much memory you would like to devote to nodes before they are written to disk. The performance impact of putting nodes out to disk is typically quite small.

MINOS

- MINOS5 (old version of MINOS) was dropped. MINOS5 now is an alias to MINOS (Version 5.51) in order to protect users that hard-coded the use of MINOS5.

MOSEK

- New libraries: version 6.0
- Improvement of speed and stability of the interior-point optimizer for linear and conic problems
- More effective presolve for simplex optimizer with hot starts

XPRESS

- New libraries: version 20.00
- Updated licensing: multi-threaded runs and 64-bit versions now included in base license
- Updated `threads` option allows specification of cores left free for non-XPRESS work.

In-core communication solver links

- Lindoglobal supports in-core communication

3.42.1.4 Model Libraries

GAMS Data Library

- **transxls (74)**: Solve classical transportation problem in Excel, using vba API (gamsx, opt and gdx)

GAMS EMP Library

- **negishi (21)**: Pure exchange model solved with EMP, SJM, and CGE

GAMS Model Library

- **bchtsp (348)**: Traveling Salesman Problem Instance with BCH
- **bchstock (349)**: Cutting Stock - A Column Generation Approach with BCH
- **tanksize (350)**: Tank Size Design Problem

GAMS Test Library

- **set10 (449)**: first entry in set/param data has wrong dimension
- **set11 (450)**: test for order when using `set.id` or `set#id` or `#id`
- **gdxcopy5 (451)**: Test GDX environment variables with trailing blanks
- **ifthen2 (452)**: Test nested $ifthen and $endif
- **ifthen3 (453)**: Test $ifthen $elseif/else $endif
- **ifthen4 (454)**: Tests $ifthen from old release notes
- **eval01 (455)**: constant evaluation with non-stop arithmetic
- **eval02 (456)**: $eval/ife/... sameas function
- **forceerr (457)**: ForceError and ForceErrorCnt Example
- **gdxxrw5 (458)**: Test for password handling of gdxxrw and $hiddencall
- **lolp (459)**: LP Test for logoption
- **lonlp (460)**: NLP Test for logoption
- **lomcp (461)**: MCP Test for logoption
- **eval03 (462)**: test constant compile time evaluation functions
- **eval04 (463)**: test constant compile time evaluation operators

3.42.1.5 Solver/Platform Availability Matrix

3.42.2 23.3.2 Maintenance release (November 18, 2009)

- Gurobi 2.0.1 library with bug fixes
- MOSEK 6 rev 53 library with bug fixes
- Updated CoinOS library with bug fixes
- Solver optfile bug fix in GAMSCHK
- Synchronized `tolproj` (1e-8) option for `solvelink=3`

3.42.3 23.3.3 Maintenance release (December 17, 2009)

- MOSEK 6 rev 55 library with bug fixes
- 32bit HAR utilities available in 64bit Windows System
- Slow solution reporting in GAMS/Gurobi and other solvers fixed

3.43 23.2 Distribution

3.43.1 23.2.1 Minor release (August 14, 2009)

3.43.1.1 Acknowledgments

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

3.43.1.2 Solvers

Cplex

- New libraries 12.1
- Now also available on 64-Bit Intel Mac

LindoGlobal

- New libraries 6.0.1.299

3.43.1.3 Solver/Platform Availability Matrix

3.44 23.1 Distribution

3.44.1 23.1.1 Major release (July 13, 2009)

3.44.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Anibal Blanco, Wietse Dol, Arne Drud, Josef Kallrath, Erwin Kalvelagen, Niclas Mattsson, Bruce McCarl, Dominique van der Mensbrugghe, Renger van Nieuwkoop, Stefan Vigerske, and Tongxin Zheng.

3.44.1.2 New Platform

- **(64-Bit) Intel Mac**

3.44.1.3 GAMS System

GAMS

Default Upper Bound on Integer Variables

- The default upper bound on integer variables has changed from 100 to +INF. Since some solve steps in a GAMS applications may inadvertently depend on the previous default value of 100, a the compatibility option, the GAMS parameter `PF4=n`, has been introduced to control what values will be passed to the solver. If the GAMS parameter `PF4` is not used (a value of `PF4=1` is assumed) the solve steps will work as in previous releases.
  - `Pf4=0`: The new default upper bound of +INF will be passed to the solver.
  - `Pf4=1`: The value of 100 instead of +INF will be passed to the solver. The solvers will operate as with older GAMS versions. In addition messages will be written to the log and listing to report on the number of integer or semi-integer variables which had the new default bound of +INF reset to 100.
  - `PF4=2`: The new default values of +INF will be used as with `PF4=0`. When a solution is returned to GAMS and the level value of an integer variable exceeds the old bound value of 100, a message will be written to the log and listing.
  - `PF4=3`: The same as `PF4=2` with an additional execution error issued if the solution reports a level value greater than 100 for any integer variable with a default bound of +INF.

  Setting `PF4` values to 2 and 3 is a convenient way to test if the application relies on the previous default bounds of 100. Future releases may use `PF4=0` as the default.

  An historic note: when GAMS was first introduced, some MIP solvers could only handle binary variable and GAMS applied transformations to simulate integers by using a power expansions. In addition, MIP solvers with integer variables imposed different restrictions on the largest upper bound value. Finally, solvers like DICOPT introduced integer cuts during the solution process, needing some finite upper bound on integer variables. What was once a good choice, turned over time into a source of confusion or resulted in unexpected model behavior.

Default Iteration Limit

- The default iteration limit (`IterLim`) has been increased from 10000 to 2e9. Setting `IterLim` to `INF` will not work since it is treated as an integer by GAMS and many solvers. Some solver, e.g. GAMS/Gurobi, recognize 2e9 and set the solver iteration limit to infinity.

Enhanced Profiling Options

- In addition to `PROFILE=n` and `PROFILETOL=r`, a new GAMS parameter `PROFILEFILE=file` has been introduced to write profiling information to a text file with some fixed format which can easily be imported into a spreadsheet for further analysis.

```
    1        -1        0.000 ExecInit
  139        15        0.000 Assignment cnf
  139        13        0.000 Assignment cnf
    .         .         .    .
    .         .         .    .
    .         .         .    .
 3549       432        0.000 Equation   nbal
 3621     39489        0.032 Solve Fini wsisn
 3621        -1        0.062 GAMS Fini
    1        -1        0.000 ExecInit
 3621        -1        0.047 Solve Read wsisn
 3621        -1        0.000 GAMS Fini
```

A summary report of the ten slowest execution steps will be written to the log and listing.

```
--- Profile Summary (184 records processed)
     0.062      3621 GAMS Fini
     0.047      3621 Solve Read wsisn
     0.046      3529 Equation   divcnlsea (86)
     0.032      3621 Solve Fini wsisn (39489)
     0.016      3274 Assignment wnr (2502)
     0.016      3447 Equation   cost (15)
     0.016      3475 Equation   laborc (180)
     0.016      3519 Equation   waterbaln (180)
     0.016      3546 Equation   subirrc (84)
     0.015      3030 Assignment gwtsa (273)
```

The timing on slow data definitions and gdx loads during compilation will be profiled as well.

Macros

- Added a line continuation character (\) for macro definitions. See test model **macro02**.

Other

- Faster reading of data statements that are not sorted
- Reduced memory usage for projections
- GDX handles IEEE special values

GAMS Data Utilities

GDXDIFF

- Added FldOnly and DiffOnly parameters to write different formats
- Protection against overwriting an input file

GDXVIEWER

- Fixed problem with cubed view

GDXXRW

- Added RWAIT parameter to specify a delay to open Excel to avoid not ready problems
- Added CheckDate option to regenerate output only if input is more recent than output file

SCENRED2

- A libinclude file runscenred2 has been added to make the communication with Scenred2 simpler. See example **srpchase**.

GAMS IDE

- Changed the interface for dealing with tabs
- Tabs expanded to spaces will mark the file as modified
- Added command to select text from current position to a text marker
- Added checks to see if IDE is visible on the monitor
- Move and Size are available on TaskBar icon

Documentation

- Updated McCarl GAMS User's Guide

3.44.1.4 Solvers

AlphaECP

- New libraries 1.75.03

Coin-OR

- New solver CoinCouenne
  - CoinCouenne is a global optimization solver for non-convex mixed integer non-linear programs, similar to the commercial solvers BARON and LindoGlobal. The solver is still in an experimental phase and is hidden in the GAMS system.
- New libraries
  - Cbc 2.3
  - Glpk 4.37
  - Ipopt 3.6
  - Scip now uses Clp 1.10

EMP

- Disjunctive programs can be solved via the following alternative automated reformulations without changes to the model
  - Convex Hull
  - BigM
  - CPLEX indicators
  - Examples
    - EMP Library **sequence (20)**: Sequencing on a single machine
    - Model Library **bilinear (346)**: Convexification of bilinear term binary times x

GUROBI

- New libraries 1.1

KNITRO

- New libraries 6.0
  - Introduces MINLP capability: binary and integer variables are supported. Two algorithms are available, a non-linear branch and bound method and an implementation of the hybrid Quesada-Grossman method for convex MINLP. The Knitro MINLP code is designed for convex mixed integer programming and is a heuristic for nonconvex problems.
  - General performance improvements for both the active-set and interior-point/barrier solvers
  - Reorganized options into groups: general, barrier and MINLP.

MOSEK

- New libraries 5.0 rev 127

In-core communication solver links

- Support of COINBONMIND, COINCBCD, COINIPOPTD and LGOD as in-core communication solvers.

  Where a traditional link already exists, the newer in-core link version has a "D" appended to the name (D for DLL). These in-core links are very similar to their traditional predecessors. They may lack some functionality but offer in-core communication between GAMS and the solver, making potentially large model scratch files unnecessary. This can save time if you solve many models in your GAMS program.

3.44.1.5 Model Libraries

GAMS Data Library

- **Portfolio (68)**: Determines an efficient frontier in Excel, using the GDX DLL (vba API) and the GAMS executable
- **Sudoku (69)**: Solve a Sudoku in Excel, using the GDX DLL (vba API) and the GAMS executable
- **Samurai (70)**: Solve a Samurai Sudoku in Excel, using the GDX DLL (vba API) and the GAMS executable
- Samurai2 (71): Solve a Samurai Sudoku in Excel, using the GDX and GAMSX DLL (vba API) *[not available anymore]*
- **CHP (72)**: Optimize combined heat and power generation in Excel, using vba API (gamsx and opt) and GDXXRW
- **CHP2 (73)**: Optimize combined heat and power generation in Excel, using vba API (gamsx, opt and gdx)

GAMS EMP Library

- **zerofunc (18)**: Match unmatched vars with zero functions in VI
- **traffic2 (19)**: Traffic Assignment Model
- **sequence (20)**: Sequencing on a single machine

GAMS Model Library

- **secure (343)**: Secure Work Files - Example 2
- **srpchase (344)**: Scenario Tree Construction Example
- **tsp5 (345)**: Traveling Salesman Problem - Five
- **bilinear (346)**: Convexification of bilinear term binary times x
- **solveopt (347)**: option solveopt explained

GAMS Test Library

- **minos01 (433)**: MINOS test suite - anti-cycling via expand frequency
- **scnred02 (434)**: Scenred2 test - tree reduction and zero values
- **empvi05 (435)**: Test for EMP's treatment of an odd variable
- **nlcode3 (436)**: Wrong NL code generation for odd case
- **set8 (437)**: Test the `set.id` `set#id` `#id` data macros
- **nlcode4 (438)**: More NL testing when code is deleted
- **compile7 (439)**: Test and document some system.XXX macros
- **mcp09 (440)**: Test inequalities with infinite bounds
- **empadj01 (441)**: Test for EMPs AdjustEqu/NYslp option
- **emp05 (442)**: Test for handling of constant equations
- **examin03 (443)**: EXAMINER test suite - test returnInitPoint option
- **compile8 (444)**: Test for a mapping error
- **sl4mip01 (445)**: Test for known XPRESS bug, with solvelink=1,2,4,5
- **macro02 (446)**: Multi-line macro test
- **put10 (447)**: $on/offput inserted outside the loop
- **set9 (448)**: fast shifting of set elements

3.44.1.6 Solver/Platform Availability Matrix

3.44.2 23.1.2 Maintenance release (July 23, 2009)

- MPSGE: MPSGE ignored the `m.workspace` and `m.workfactor` settings
- GAMSCHK: Fixed a problem with the layout of the reports in some models

3.45 23.0 Distribution

3.45.1 23.0.2 Major release (February 14, 2009)

3.45.1.1 Acknowledgments

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Surendu Korgaokar, Tom Rutherford, Stefan Vigerske and Rich Roberts.

3.45.1.2 GAMS System

GAMS Data Utilities

GDXXRW

- Fix to not limit the size of the block read to 65536 lines

Documentation

- Updated McCarl GAMS User's Guide

Other

As announced earlier we dropped the following systems as of this GAMS Distribution.

- GAMSBAS: The newer savepoint/loadpoint facility is easier to use and more robust.
- MPSWRITE: The newer CONVERT utility includes this capability

3.45.1.3 Model Libraries

GAMS Model Library

- **poutil (342)**: Portfolio Optimization for Electric Utilities

GAMS Test Library

- **privat01 (432)**: Test private workfile behavior
- **emp04 (431)**: Compare results of EMP runs using different options for nlp problem
- **local01 (430)**: Test .local in different context $on/offlocal
- **empdisj1 (429)**: Test EMP Disjunction - Minimize the makespan
- **empecs02 (428)**: Test for EMP-Embedded Complemenarity System
- **empecs01 (427)**: Test for EMP-Embedded Complemenarity System
- **empvi04 (426)**: Test for EMP Variational Inequalities
- **empvi03 (425)**: Compares alternative EMP-VI models
- **empvi02 (424)**: Compares manual and EMP-VI models
- **empvi01 (423)**: Simple test for variational inequalities through EMP
- **empbp03 (422)**: Simple bilevel model, both inner and outer problems have constraints
- **empbp02 (421)**: Simple bilevel model, outer problem consist of objective only
- **empbp01 (420)**: Simple bilevel model, inner problem consist of objective only
- **convert7 (419)**: CONVERT test suite - test hessian info correctness
- **lp14 (418)**: Simple test on one equation AFTER reformulation
- **card01 (417)**: Test extended card and ord functions
- **suffix01 (416)**: Test suffix options on controlling sets
- **nlp01 (415)**: Test of correctness for levels & marginals of NLP

3.45.1.4 Solvers

As announced in the 22.9 release notes we dropped the following solvers as of this GAMS Distribution.

- CONOPT1 (old version of CONOPT). The default CONOPT version, CONOPT 3, will not be dropped.
- MILESOLD (old version of MILES). The default MILES version will not be dropped.
- OSL1 and OSL2 (old versions of OSL). The default OSL version will not be dropped.
- PATHOLD (old version of PATH). The default PATH version will not be dropped. Due to user requests, and despite our earlier announcement, we did *not* drop the following components:
- CONOPT 2 (old version of CONOPT).
- DEA

CPLEX

- New libraries 11.2.1

GUROBI

- The new Gurobi solver provides state-of-the-art simplex-based linear programming (LP) and mixed-integer programming (MIP) capability.
- The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation will be deterministic: two separate runs on the same model will produce identical solution paths.
- The Gurobi solver is available for the 32-bit and 64-bit versions of Windows and Linux. Please contact us for an evaluation license.

KNITRO

- New libraries 5.2.0

LINDOGLOBAL

- New libraries 5.0.1.345

MOSEK

- New libraries 5.0 rev 112

XPRESS

- New libraries 19.00.04
- The primal algorithm has been improved and is now 40% faster on XPRESS' LP benchmark test set and 20% faster on XPRESS' large LP benchmark test set.
- The dual algorithm has been improved and is now 22% faster on XPRESS' LP benchmark test set and 45% faster on XPRESS' large LP benchmark test set.
- The MIP performance has been improved and is now 40% faster on XPRESS' MIP benchmark test set
- Improvements to the Quadratic Primal algorithm to avoid cycling problems

In-core communication solver links

- Support of GUROBI as an in-core communication solvers.
- Experimental in-core link MINOSD
- Enhancements and bug fixes for the in-core communication solver links BDMLPD, CONOPTD, and CPLEXD Where a traditional link already exists, the newer in-core link version has a ”D” appended to the name (D for DLL). These in-core links are very similar to their traditional predecessors. They may lack some functionality but offer in-core communication between GAMS and the solver, making potentially large model scratch files unecessary. This can save time if you solve many models in your GAMS program.

Solver/Platform Availability Matrix

3.46 22.9 Distribution

3.46.1 22.9.2 Major release (December 01, 2008)

3.46.1.1 Acknowledgements

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Stefan Boeters, Wolfgang Britz, Alexander Gocht, Josef Kallrath, Erwin Kalvelagen, Todd Munson, Yiqi Zhu.

3.46.1.2 Future Deprecation

We are planning to drop the following systems from our next GAMS Distribution 23.0. If you object please submit an email to support@gams.com!

- CONOPT1 and CONOPT 2 (old versions of CONOPT)
- DEA (capability will be offered by a more general scenario framework)
- GAMSBAS
- MPSWRITE
- MILESOLD (old version of MILES)
- MINOS5 (old version of MINOS)
- OSL1 and OSL2 (old versions of OSL)
- PATHOLD (old version of PATH)

3.46.1.3 GAMS System

GAMS

- **GAMS scratch file extension**. Starting with distribution 22.9 the default file extension of intermediate files located in the `225?` directories is `.dat`. The scratch extension is a parameter that can be changed with the GAMS option `ScrExt`, e.g `gams trnsport scrext=tmp`. Within GAMS code you get the scratch extension using `%gams.scrext%`.
- GAMS Parameter **ETlim** (Elapsed Time limit). A GAMS job will terminate if the elapsed time in seconds exceeds the value of ETlim. The job elapsed time is checked before the execution of a $call, execute or solve statement. The system will terminate with a compilation or execution error if the limit (default INF) is reached.
- GAMS Parameter **AppendExpand** (short form **AE**). Allows to control the file opening of the expand file. A value of 0 will open with rewrite, a value of 1, the default, will open with append.
- New **solveopt=clear** option. A third option has been added to option `solveopt=xxx`. The related model attribute <`model`>`.solveopt=n` has been adjusted to match the new option values. If the model attribute is set to `NA` (default), the setting of the corresponding option statement will be used to determine the way solutions values are loaded back into the GAMS data space. The possible values are (the numeric value for the model attribute are given in parenthesis):

  - `replace` (0) all equations appearing in the model list will be completely replaced by the new model results. Variables are only replaced if they appear in the final model.
  - `merge` (1) the new model results are merged into the existing structures. This is the default.
  - `clear` (2) similar to the replace option; in addition, variables appearing in the symbolic equations but squeezed out in the final model, are removed.

- New **solPrint=silent** option. A third option has been added to option `solPrint=xxx`. The related model attribute <`model`>`.solPrint=n` and the GAMS parameter `solPrint=n` have been adjusted to match the new option values. If the model attribute is set to `NA` (default), the setting of the corresponding option statement will be used to control the printing of model generation and solution information. Note that the GAMS parameter initializes the corresponding option statement values. The possible values are (the numeric value for the model attribute are given in parenthesis):
  - `off` (0) detailed solution output is suppressed.
  - `on` (1) the most detailed solution output. This is the default.
  - `silent` (2) all solve related output is suppressed.

- New model attributes returning **elapsed time** information of the solution process:
  - `ETsolve` total elapsed time to execute a solve statement
  - `ETsolver` elapsed time that can be attributed to the solver only
  - `ETalg` elapsed time that can be attributed to the core algorithm

- New suffix for **abort.noerror**. When using the suffix `.noerror` with the `$abort` statement, the error count will NOT be increased. When a save file is written, all remaining unexecuted code will be flushed. This allows effective reuse of the save file.

- Added a test and error message to see if a restart file has not been closed yet when we try to use the file.

The GAMS Macro Facility

The GAMS macro facility has been inspired by the GAMS-F preprocessor for function definition developed by Ferris, Rutherford and Starkweather, 1998, 2005. The GAMS macro facility incorporates the major features of the GAMS-F preprocessor into the standard GAMS release. The GAMS macros acts like a standard macro when defined, however, its recognition for expansion is GAMS syntax driven.

Macros are widely used in computer science to define and automate structured text replacements. The GAMS macro processors functions similar to the popular C/C++ macro preprocessor. The definition takes the form

```
$macro name  macro body
$macro name(arg1,arg3,arg2,..) macro body with tokens arg1,..
```

The name of the macro has to be unique, similar to other GAMS data types like sets and parameters. A (following immediately the macro name starts the list of replacement arguments. The macro body is not further analyzed after removing leading and trailing spaces.

The recognition and following expansion is directed by GAMS syntax. The tokens in the macro body to be replaced by the actual macro arguments follow the standard GAMS identifier conventions. For example:

```
$macro diff(y) system.cosh(y) - cosh(y)
$macro cosh(x) (exp(x) + exp(-x))/2
scalar z; z = diff(2/3); display z;
```

will expand into:

```
scalar z; z = system.cosh(2/3) - (exp(2/3) + exp(-2/3))/2; display z;
```

This expansion takes place in two steps, first GAMS recognizes diff as a macro and changes the input text to:

```
scalar z; z = system.cosh(2/3) - cosh(2/3); display z;
```

Gams will continue to process the modified input text. The first occurrence of cosh is not recognized by gams as a macro, only the second reference to `cosh(2/3)` will result in a second and final expansion.

The recognition of macros and expansion of arguments can further be controlled by the use of ampersands (&) in the macro body. A single ampersand (&) is used as a concatenation or separation symbol to recognize tokens to be replaced. Two ampersands (&&) immediately preceding a token will drop the most outer matching single or double quotes of the replacement argument. For example:

```
$macro  f(i)  sum(j, x(i,j))
$macro equ(q)  equation equ_&q; equ_&q.. q =e= 0;
equ(f(i))
```

will expand into:

```
equation equ_f(i); equ_f(i).. sum(j, x(i,j)) =e= 0;
```

The first step of the above expansion is shown below. GAMS will then only recognize the third occurrence of `f(i)` as a macro which will be expanded to give the above result.:

```
equation equ_f(i); equ_f(i).. f(i) =e= 0;
```

The actual calling arguments of macros can contain complex expressions and other macro calls. Multiple arguments are separated by commas. Pairs of parenthesis and quotes can be used freely to protect the separating comma.

```
$macro many(a,b) scalar x; x=a/&&b); display x;
many((3/5),'(mod(3,2)')
```

Note that the second argument has unbalanced parenthesis and therefor needs to be enclosed in quotes to give the result below.

```
scalar x; x=(3/5)/(mod(3,2)); display x;
```

Some macro use can result in an expansion of infinite length. For example:

```
$macro a  b,a
display a;
```

will expand into:

```
display b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,...
```

GAMS will eventually refuse to do more substitutions and issue a compilation error.

The use of deeply nested macros may require the use of aliased sets in indexed operations like sum and prod. A minor syntax extension allows the implicit use of aliases. The suffix `.local` on a controlling set will use an implicit alias within the scope of the indexed operation. For example:

```
$macro qq(i) sum(i.local, b(i))
c(q) = qq(q);
```

will expand into:

```
c(q) = sum(q.local, b(q));
```

The use of the `.local` modifier is not limited to macros and can be used in any context.

Another feature motivated by the use of macros is the implicit use of the `.l` suffix in data manipulation statements. This allows using the same algebra in model definitions and assignment statements. The following code snippet illustrates this feature:

```
$macro D(s,h) INCOME(h)*alpha[s,h]*sum{s.local, alpha[s,h]*P[s]*
.  .  .
cmkt(s)..    Y(s) =g= sum(h, D(s,h));
. .  .
$onDotL

dl(s,h) = d(s,h); display dl;
```

where `INCOME` is another macro which calls a number of other nested macros. `D(s,h)` is now used to define the equation `cmky` and is also used in an assignment after the model has been solved. The `$ondotl` enables the implicit `.l` suffix for variables. This feature was introduced to make macros more useful and is not limited to be used in macro bodies. Since this a new feature it has to be enabled. The matching `$offdotl` will disable this feature.

Three more switches are relevant to macros. The `$show` will list any GAMS macros defined. The `$onmacro`/`$offmacro` will enable or disable the expansion of macros; the default is `$onmacro`. Finally, the `$on/offexpand` will change the processing of macros appearing in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch `$onexpand` enables the recognition and expansion of macros in the macro argument list. `$offexpand` will restore the default behavior.

Macro definitions are preserved in a save/restart file and are available again when performing a continued compilation.

Comparison of GAMS-F with GAMS macros

The GAMS-F preprocessor combines aspects of traditional macros with that of functions and objects. The example 7 from the GAMS-F documentation using macros can be found in the GAMS model library under the name **two3mac**. Some of the main differences between GAMS macros and the GAMS-F preprocessor are:

- A macro is a new GAMS data type and shares the name space of GAMS symbols, like sets, parameters, variables,etc.

- Instead of using <id> == <body>; to recognize a macro, we use $macro <id> <body> with the $ in position 1.
- Surrounding parenthesis are not automatically added to the macro body when the macro is expanded.
- A macro is recognized and expanded anywhere a proper GAMS identifier can be used. This can be suppressed by $onmacro/offmacro.
- No automatic definition of new aliased sets and their use in controlled index positions. The .local feature has been added to ensure local use and eliminates the need for new alias definitions.
- No automatic equation definitions for MCP models.
- The body of the macro is only used during expansion. Hence, the macro definitions are not order dependent.
- Variables in macro bodies will have an implicit .L when using in assignment statements. This is not a macro expansion feature, but a new GAMS feature that needs to be activated by $onDotL/$offDotL

GAMSIDE

- The model library open file no longer converts the file name to lowercase
- The ViewClose command for IDECmds now allows closing for any file
- Added option to execute command scripts (.cmd files) from the IDE
- Pressing F3 (search again) after a search/replace will restart the replace dialog
- Output of the put_utility 'title' option will be shown in the process window

GAMS Data Utilities

- CHOLESKY
  - Compute the Cholesky factors of a symmetric positive-definite matrix
- EIGENVALUE
  - Compute the eigenvalues of a symmetric matrix
- EIGENVECTOR
  - Compute the eigenvalues and eigenvectors of a symmetric matrix
- GDXCOPY
  - When we convert a gdx file using the **-Replace** option, and the file is open in the IDE, the IDE will be signaled to close the file and reopen the file after the conversion is complete.
- GDXDUMP
  - Updated documentation.
- GDXVIEWER
  - Allow resizing of the column width
- GDXDCLIB
  - Reading an aliased set could return too many elements
- GDXMERGE
  - Fixed a problem when we specify a list of identifiers

SCENRED2

Scenred2 is an updated and expanded version of the scenred utility for scenario reduction. Scenred2 is intended to be a replacement for the existing scenred, but we have made the newer version available as scenred2 due to some differences in the options used to control scenred's behavior. Having both available also facilitates comparisons between the two.

New features in scenred2 include:

- Tree construction. Scenred could only reduce existing scenario trees, while scenred2 can create trees from collections of independent scenarios (i.e. from scenario "fans").
- Visualization. Scenred2 contains new options to create input files for GNUPLOT.
- Improved metrics. Tree reduction can now be carried out w.r.t. the Fortet-Mourier metric, instead of the upper bounds given by the Monge-Kantorovich metric.

3.46.1.4 Documentation

- Updated McCarl GAMS User's Guide

3.46.1.5 Other

- Semidefinite Programming Solver CSDP available for Windows, Linux and Macintosh on Intel distributions. Some examples are included in the GAMS Model Library:
  - trnssdp (340)
  - **gqapsdp (339)**
  - **maxcut (338)**
- Additional Lahey Fortan version of all API files does not depend on a C compiler

3.46.1.6 Model Libraries

GAMS Model Library

- **two3mac (341)**: Simple 2 x 2 x 2 General Equilibrium Model Using Macros
- trnssdp (340): Solving the Transportation LP Problem using SDP
- **gqapsdp (339)**: SDP Convexifications of the Generalized Quadratic Assignment Problem
- **maxcut (338)**: Goemans/Williamson Randomized Approximation Algorithm for MaxCut

GAMS Test Library

- **load4 (414)**: Tests various file opening options for GDX files
- **eigvec02 (413)**: Test Eigenvector/Eigenvalue utilities
- **choles02 (412)**: Test 2 Cholesky utility
- **choles01 (411)**: Test 1 Cholesky utility
- **eigvec01 (410)**: Test Eigenvector utility
- **eigval01 (409)**: Test Eigenvalue utility
- **scnred01 (408)**: Basic Scenred2 test - tree construction
- **scensol1 (407)**: Basic Scenario Solver Test
- **gdxmerg2 (406)**: Bad acronym merge in gdxmerge
- **gdx8 (405)**: Bad acronym mapping
- **gdx7 (404)**: Bad acronym loading
- **macro01 (403)**: Several macro tests
- **assign2 (402)**: Test for bad assignment with .fx
- **mcp08 (401)**: Test MCP that turfs up PATH preprocessing error
- **convert6 (400)**: CONVERT test suite - hessian.gdx

3.46.1.7 Solvers

Coin-OR

- New libraries
  - Bonmin 0.100
  - Cbc 2.2
  - Glpk 4.32
  - Ipopt 3.5
- Scip 1.1
- Mumps 4.8.3 (used by Ipopt and Bonmin)
- CoinCbc can now use multiple threads (see new option "threads"). This option is available for all platforms other than Windows.
- Scip supports special ordered set of type 1 and 2 (SOS1 and SOS2). Further, a new heuristic and a new cutting plane separator has been added, the preprocessing has been improved, the Clp interface revised, and bugs were fixed.

Cplex

- New libraries 11.2

  CPLEX 11.2 offers finer control for solution polishing. In previous versions, the only stopping criterion for solution polishing was set by the parameter `PolishTime` to limit time spent polishing a solution. General stopping criteria, such as the time limit, absolute MIP gap, relative MIP gap, MIP node limit or MIP integer solution limit did not apply to solution polishing.

  Now, however, CPLEX 11.2 allows the user to control more finely when solution polishing terminates. In other words, the usual tolerances (`EpAGap` and `EpGapinitialized` with GAMS parameters `OptCA` and `OptCr`) and limits (`IntSolLim`, `NodeLim` and `TiLim` initialized with GAMS paramter `NodLim` and `ResLim`) now apply to solution polishing.

  In addition to those existing parameters that now control the termination of solution polishing, there are also new parameters specific to the starting conditions for solution polishing.

  With these new parameters, a user can tell CPLEX when to switch from branch & cut to solution polishing. CPLEX is able to switch after it has found a feasible solution and put into place the MIP structures needed for solution polishing. When these two conditions are met (feasible solution and structures in place), CPLEX stops branch & cut and switches to solution polishing whenever the first of these starting conditions is met:

  - when CPLEX achieves a specified absolute MIP gap (PolishAfterEpAGap)
  - when CPLEX achieves a specified relative MIP gap (PolishAfterEpGap)
  - when CPLEX finds a specified number of integer solutions (PolishAfterIntSol)
  - when CPLEX processes a specified number of nodes (PolishAfterNode)
  - when CPLEX reaches a specified time limit on time spent in optimization (PolishAfterTime)

  The new parameters are incompatible with the deprecated option PolishTime. If you use them together in an option file you will see an error like this:

```
Reading parameter(s) from "C:\tmp\cplex.opt"

>>  polishtime 5
    Warning line 1: deprecated option "polishtime"; Use option polishafter... for finer soluti
>>  polishafterintsol 1
Finished reading from "cplex.opt"
CPLEX Error  1807: Incompatible parameters.
polishafterintsol: current = 2100000000, default = 2100000000, minimum = 1, maximum = 2100000
```

  A few examples with the corresponding GAMS/CPLEX option file:

  - As an example of how to manage time spent polishing a feasible solution, suppose the user wants to solve a problem by spending 100 seconds in branch & cut and an additional 200 seconds in polishing:

```
TiLim 300
PolishAfterTime 100
```

  - Switch to polishing after first feasible solution:

```
PolishAfterIntSol 1
```

  - For example, the following procedure applies branch & cut until it reaches a 10% gap. Then it starts solution polishing until it narrows the gap to 2%.

```
PolishAfterEpGap 0.1
EpGap 0.02
```

Lindoglobal

- New libraries 5.0.1.292 now also for Sun Sparc Solaris

PATH

- New libraries 4.7.01 fix preprocessing bug for both MCP and NLP front ends

BDMLPD, CPLEXD and CONOPTD

- Enhancements and bug fixes for the three experimental in-core communication solver links BDMLPD, CPLEXD and CONOPTD.

3.46.1.8 Solver/Platform Availability Matrix

3.47 22.8 Distribution

3.47.1 22.8.1 Major release (August 01, 2008)

3.47.1.1 Acknowledgements

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Andrea Consiglio, Anton Eremeev, Mustafa Esen,Josef Kallrath, Erwin Kalvelagen, Todd Munson, Rich Roberts, and Andres Ramos.

3.47.1.2 GAMS System

GAMS

- GAMS scratch file extension: Starting with distribution **22.9** the default file extension of intermediate files located in the 225? directories will change from `.scr` to `.dat`. **For distribution 22.8 it is still** `.scr`. The scratch extension now is a parameter that can be changed with the GAMS option `ScrExt`, e.g `gams trnsport screxT=tmp`. Within GAMS code you get the scratch extension using `%gams.screxT%`.
- Increased the maximum input line length to 40,000 characters and the maximum number of columns in a table to 10,000.
- Checking for an interrupt is now also done inside a GAMS looping constructs.
- The `$LOAD` directive can read the universe from a gdx file by specifying `$LOAD id=`$*$.
- Certain gdx file read operations are now faster and use less memory.
- GAMS parameters `gdxcompress` and `gdxconvert`
  - Allow new gdx files to be written in an older format of the gdx file.
  - Usage through an environment variables (valid for all gdx related operations): `gdxconvert = <value>` and `gdxcompress = <value>` or via GAMS command line parameters: `gdxconvert = <value>` and `gdxcompress = <value>`.
  - Possible values for gdxcompress
    - `0` (or empty): no compression
    - `1`: compression turned on
  - Possible values for gdxconvert:
    - V7: version 7

    - V6: version 6
    - V5: version 5
  - E. g. to get a compressed v6 gdx file enter: `gams <model_name> gdxconvert=v6 gdxcompress=1 gdx=<gdx_file>`.
  - Note:
    - With GAMS 22.8 the default format for gdx files is V7 uncompressed.
    - Only V6 and V7 support compression.
    - V7 formatted files were introduced with version 22.6 of GAMS; V6 formatted files were introduced with version 22.3 of GAMS. GAMS platforms that were introduced after 22.3/22.6 (e.g. Mac Intel or SunSparc64) do not support V5/V6.
    - The command line options have a higher precedence as the environment variables with the same name.

Utilities

- New
  - `gdx2xls`: Converts an entire gdx data container to a Microsoft Excel spread sheet.
  - `invert`: Calculates the inverse of a matrix provided as a gdx file (for more information see gdxutils documentation).
  - `msappavail`: Checks which Microsoft Office programs are installed.
  - `xlstalk`: Allows some interaction with Excel to open/close/save Excel files.
- Extended/Updated
  - `gdxcopy`: New option to replace existing gdx files.
  - `gdxdiff`: New id option to compare specified ids only.
  - `gdxmerge`: New optional output parameter to specify the name of the output file.
  - `gdxviewer`: Fixed problem with cube view.

GAMSIDE

- Fixed problem when moving a column to the plane in the gdx data viewer.

3.47.1.3 Model Libraries

GAMS Test Library

- **New models**, including tests for
  - invert utility
  - ls solver
  - poly function
  - gdxconvert and gdxcompress parameters

GAMS Data Utilities

- GAMS introduces the new model library 'GAMS Data Utilities' containing models that demonstrate the various utilities to interface GAMS with other applications.

PRACTICAL FINANCIAL OPTIMIZATION Models

- The models of the forthcoming book PRACTICAL FINANCIAL OPTIMIZATION - A Library of GAMS Models by Andrea Coniglio, Soren Nielsen, and Stavros A. Zenios have been included in the GAMS distribution. It is a companion volume to the book `Practical Financial Optimization` by Stavros A. Zenios.

3.47.1.4 Solvers

BARON

- New libraries 8.1.5 for Windows, Linux, and AIX

BDMLPD

- GAMS 22.8 introduces a third experimental solver BDMLPD besides CONOPTD and CPLEXD. They are very similar compared to their professional brothers BDMLP, CONOPT and CPLEX. They lack some functionality (e.g. CPLEXD does not solve QCP models) but offer in-core communication between GAMS and the solver. No large model scratch files need to be written to disk which can save time if you solve many models in your GAMS program. This in-core execution is activated by setting <`modelname`>`.solvelink=5;` before the solve statement.

Coin-OR

- CoinBonmin
  - New libraries 0.99
  - Support of user-defined cut generators and heuristics via BCH (Branch and Cut Heuristic)
- CoinCbc new libraries 2.1
- CoinScip supports user-defined cut generators, heuristics, and incumbent report callbacks via BCH

Convert

- New option `hessian` to dump the Hessian matrix into a GDX file. Similar to the option `jacobian`.

Cplex

- New libraries 11.1.1

Lindoglobal

- New libraries 5.0.1.292. Sun Sparc Solaris 5.0.1.274.

LS

- Linear Least Squares Solver

Mosek

- New libraries 5 Rev 90

XA

- New libraries for Windows

3.47.1.5 Solver/Platform Availability Matrix

3.48 22.7 Distribution

3.48.1 22.7.1 Major release (May 01, 2008)

3.48.1.1 Acknowledgements

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Jens Baudach, Michael Ferris, Josef Kallrath, Aldo Vecchietti and Stefan Vigerske.

3.48.1.2 GAMS System

GAMS

- Enhanced Data Statements

  The data statements have been enhanced to allow initial values for equations and variables in addition to set and parameter data. Those new data statements follow the syntax for list and table data statement for parameters by adding an additional dimension to specify the specific data attribute. The variable and equation suffixes can be used in this additional dimension. For example, the solution values for the transport example could be written as:

```
variable x(i,j) / (seattle.(new-york 50, chicago 300)
                   san-diego.(new-york,topeka) 275   ).l
                  seattle.topeka.m     0.36
                  san-diego.chicago.m  0.009 /;
variable z / l 153.6750 /;

equation demand(j) / (new-york 0.2250
                      chicago  0.1530
                      topeka   0.1260).m /;
```

  Only the non default values need to be specified. The following attributes can be used according to the variable type:

```
.l      level
.m      marginal
.lo     lower bound
.up     upper bound
.scale  scale value
.prior  priority for discrete variables
.fx     shorthand for setting .l,.lo and .up to the same value
```

With table style input all index positions have to appear in the row definition. For example we could write the above list oriented statement in table form as:

```
variable table x(i,j) initial values
                    l       m
seattle. new-york   50
seattle. chicago   300
seattle. topeka          0.36
san-diego.new-york 275
san-diego.topeka   275
san-diego.chicago       0.009
```

As with other data statements, `$onmulti`, `$ondelim` and `$onempty` can be used as well.

- The Matching Operator

  Mappings between n-tuples can be clumsy to enter via data statements and are often difficult to compute. Similar to the product operator (.) we introduced a match operator (:). For example, the two set data statements give the same result:

```
Set I / t1*t6:s3*s5 /
Set j / t1.s3,t2.s4,t3.s5 /
```

  In another example we may want to count "tuples" in some fashion:

```
sets h /h1*h24/, d /d1*d365/, dh(d,h) /#d.#h/
sets t /t1*t8760/, tdh(t,d,h) /#t:#dh/
```

  The resulting set tdh will then have the values:

```
t1.d1.h1, t2.d1.h2, t3.d1.h3 ..
```

  Currently there is no general matching operator for assignment statement. To facilitate some otherwise very clumsy and expensive calculations, one can use the following option statement:

```
Set ijk(I,j,k), x(I,j,k,l) ..
Option ijk(i:j,k), x(ijk:l);
```

  This statement will first clear the set ijk, then apply the matching operators to i and j and finally apply a product operation to the matched (i,j) with k which results in:

```
i1.j1.k1, i1.j1.k2, ..
i2.j2.k1, i2.j2.k2, ..
```

  And x will match ijk with the set l resulting in:

```
i1.j1.k1.l1, i1.j1.k2.l2, ..
```

- Limit on Memory Use - HeapLimit

  In some applications it may be desirable to limit the amount of memory a GAMS job can use. HeapLimit (MB) is a new GAMS parameter and function/property that limits memory used to store dynamic data. If the dynamic data storage exceeds this limit, the job will be terminate with return code 10, out of memory. These features are especially useful in a server environment.

  - The GAMS parameter HeapLimit sets the limit of memory use at compile and execution time for a GAMS job
  - The function/property HeapLimit can be used to interrogate the current limit and allows it to be rest
  - The NLP solver CONOPT also has a HeapLimit option which ensures that the solver will not use more dynamic memory

- Other enhancements
  - A symbol can have up to 20 dimensions and identifiers and labels can have up to 63 characters
  - The GAMS executable understands the 'Keep' and 'CurDir' parameters
  - New derived variable/equation attribute `.range` is defined as `x.range=x.up-x.lo`. This provides a convenient way to see if a variable is fixed
  - Additional references in compile time constant expression in `$eval` and `$ife` statement can now reference scalar parameters and the function `card(id)`
  - Added the tuple text if we display with the format x:0:0:1 which puts single items on one line
  - In table statements under `$ondelim`, we can drop the dummy element in the column definition
  - As in execution time GDX loads, we can extract the domain information with a load statement like `$load setid=parameterid`

Gams Data Exchange (GDX)

- A symbol in a GDX file can have up to 20 dimensions and identifiers and labels can have up to 63 characters
- A GDX file can store domain information for a symbol
- A GDX file can save an aliased set

GDX Utilities

- GDXDUMP writes now data for variables and equations (no longer surrounded by `$ontext` / `$offtext`)
- MDB2GMS allows writing of an empty symbol
- GDXXRW
  - No longer interprets a text field as a numeric value (because of international notation issues); the value returned for such a cell is NA
  - Fixed a problem with the default value for a field for a VAR and EQU
- GDXMERGE did not merge an aliased set correctly
- GDXVIEWER can export all symbols to Excel in command-line mode by specifying `ID=*`

GAMSIDE

- Supports an optional file, 'idecfg.ini', to display additional items like:
  - Option to open a html document
  - Multiple model libraries
  - Display of an image in the process window
- Fixed a problem matching parenthesis on an empty line
- Fixed a problem deleting a 225 directory (when this was the last directory used for opening a file)

GAMS Model Library

- **GAMS Model Library** contains six examples from LogMIP User's Manual

GAMS Test Library

- **30 new models**, including
  - Tests for proper handling of domains larger than 10 and UEL names longer than 31 characters
  - Testing of gdxmerge with aliased sets
  - New tests for DECISC and DECISM
  - Several models testing EMP

Documentation

- Updated McCarl GAMS User's Guide

3.48.1.3 Solvers

BARON

- New libraries (version 8.1.4)

CPLEX

- New libraries (version 11.0.1)

COIN-OR

- New MIP solver SCIP from Zuse Institute Berlin
  - use as option mip=coinscip;
  - uses COIN-OR LP solver CLP
  - free for academic users.
- Support of Windows 64-bit platform
- CoinCbc supports user-defined cut generators and heuristics via BCH (Branch and Cut Heuristic)
- CoinIpopt and CoinBonmin support dynamic load of linear solvers MA27, MA57 (HSL), and Pardiso.
- Minor updates in the libraries and interfaces of CoinCbc, CoinGlpk, CoinIpopt, and CoinBonmin.

CONOPT

- new option Heaplimit (see also gams option)

EMP - Extended Mathematical Programming

- (Experimental) Framework for automated mathematical programming reformulations as
  - Bilevel Programs
  - Disjunctive Programs
  - Extended Nonlinear Programs
  - Embedded Optimization Complementarity Programs

  Thereby new upcoming model types are reformulated into established math programming classes in order to use mature solver technology. EMP comes free of charge with any licensed GAMS system but needs a subsolver to solve the generated models.

LOGMIP

- LogMIP 1.0 is a program for solving linear and nonlinear disjunctive programming problems involving binary variables and disjunction definitions for modeling discrete choices. While the modeling and solution of these disjunctive optimization problems has not yet reached the stage of maturity and reliability as LP, MIP and NLP modeling, these problems have a rich area of applications. LogMIP 1.0 has been developed by A. Vecchietti, J.J. Gil and L. Catania at INGAR (Santa Fe-Argentina) and Ignacio E. Grossmann at Carnegie Mellon University (Pittsburgh-USA) and is composed of:
  - a language compiler for the declaration and definition of disjunctions and logic constraints
  - solvers for linear and non-linear disjunctive models (lmbigm, lmchull, lmlboa)

  LogMIP comes free of charge with any licensed Windows GAMS system but needs a subsolver to solve the generated MIP/MINLP models. For more information see
  - LogMIP Website

MOSEK

- New libraries (version 5.0.0.79)

CPLEXD and CONOPTD

- GAMS 22.7 introduces two experimental solvers: CPLEXD and CONOPTD. They are very similar compared to their professional brothers CPLEX and CONOPT. They lack some functionality (e.g. CPLEXD does not solve QCP models) but offer in-core communication between GAMS and the solver. No large model scratch files need to be written to disk which can save time if you solve many models in your GAMS program. This in-core execution is activated by setting `modelname.solvelink=5;` before the solve statement.

3.48.1.4 Solver/Platform Availability Matrix

3.48.2 22.7.2 Maintenance release (May 13, 2008)

- DICOPT/ALPHAECP/LOGMIPLBOA: Fixed Cplex scaling bug
- GAMS/CPLEX: Small cosmetic Cplex bug fix. We got "CPLEX Error 3003: Not a mixed-integer problem." In case we cannot solve the fixed problem. This was due to some query calls about nodes and iteration.
- GAMSIDE: Fixed a bug that could cause an out of memory error when moving a row or column to the plane index in the gdx data viewer
- GAMS/DEA: Avoid writing zeros to GDX files
- GAMS/LGOLIB: Fixed a memory leak
- Minor documentation updates
- Note: AIX and Mac OS X PPC were not updated.

3.49 22.6 Major release (December 24, 2007)

3.49.1 Acknowledgements

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Anton Eremeev, Michael Ferris, Christian Gilow, Uwe Schneider, Monique Guignard-Spielberg and Stefan Vigerske.

3.49.2 New Platforms

- Solaris on Sparc64 (64-bit)
- Mac on Intel (32-bit)

3.49.3 GAMS System

3.49.3.1 GAMS

- `Complete detailed GAMS Release Notes`
- Dollar Control Options
  - `$ife`
  - `$ifthen, $elseif, $else and $endif`
  - `$eval, $evallocal and $evalglobal`
  - `Syntax for Constant Evaluation`
- Additional Licensing Information
  - `At Compile Time: LicenseStatus and LicenseStatusText`
  - `At Run Time: LicenseLevel and LicenseStatus`

- Put Writing Facility
  - `.tn attribute`
- Enhanced Functions on controlling Sets
  - `.len, .off, .ord, .pos, .uel, .val`
- Other Enhancements
  - Unlimited input string length for `$call`, `execute` and `put_utility exec/shell`.
  - The listing file can be redirected to the standard error stream by using the gams parameter ao=3
  - The model attribute .numvarproj was added to count bound projections during model generation. Some minor additions to MODEL STATISTICS and an enhanced column listing help identify potential problems (the variable attribute .infeas could also be used to display potential bound projections)

3.49.3.2 GAMS IDE

- GDXviewer supports Excel 2007 format xlsx
- GDXviewer has an integrated symbol search

3.49.3.3 GAMS Model Library

- New models
  - Asynchronous Jacobi Methods (jacobi)
  - Dantzig Wolfe Decomposition and Grid Computing (danwolfe)
  - Portfolio Modeling with Parallel Solutions (qmeanvag)
  - Cplex Solution Pool for a Simple Facility Location Problem (solnpool)
  - Mission Planning for Synthetic Aperture Radar Surveillance (swath)
  - TSP tour plotting with LaTeX xy-pic environment (tsp2ltx)
  - Min Cost Flow with an Instance generated by NETGEN (netgen)

3.49.3.4 GDXXRW

- Supports Excel 2007 format xlsx

3.49.3.5 GDX2HAR/HAR2GDX

- GAMS distributes and supports utilities for converting HAR (header array) files used by GEMPACK: **gdx2har** and **har2gdx**. Details about these utilities can be found at Tom Rutherford's page `here`.

3.49.4 Solvers

3.49.4.1 AlphaECP

- New libraries (version 1.63). AlphaECP can call an NLP solver during the optimization to improve convergence properties for continuous variables.

3.49.4.2 BARON

- New libraries (version 8.1.1). XPRESS is available as a subsolver for LPs (option "lpsol=7"). Identification of common linear subexpressions in nonlinear functions that speeds up BARON's parsing time considerably for certain problems.

3.49.4.3 COIN-OR Solvers

- New solver libraries of CBC/CLP (2.0), GLPK (4.22), IPOPT (3.3), BONMIN (0.9). The GAMS/CoinCBC link has been completely rewritten so that it offers the performance of the CBC standalone version. Many new options have been added. The GAMS/CoinGLPK link uses now the advanced B&B solver of GLPK. Support of Solaris 64bit Intel platform and Mac OS on Intel chips.

3.49.4.4 CPLEX

- New libraries (11.0). There are quite a few changes in Cplex 11 and the GAMS/CPLEX interface. Please check the detailed GAMS/CPLEX 11 `release notes`. The major enhancements are:
  - Improved Mixed Integer Programming (MIP) Performance
  - Enhanced Parallel MIP
  - Multiple MIP Solutions
  - Performance Tuning tool

3.49.4.5 LINDOGLOBAL

- New libraries (version 5.0.1.183). Support of Solaris Sparc64 platform (SOX) and Intel Mac (DII).

3.49.4.6 MOSEK

- New libraries (version 5.0.0.62). Support of Solaris Sparc64 platform (SOX) and Mac OS on Intel chips (DII).

3.49.4.7 XPRESS

- New libraries (version 18.00)

3.49.4.8 Solver/Platform Availability Matrix

3.50 22.5 Major release (June 01, 2007)

3.50.1 Acknowledgements

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

3.50.2 GAMS System

3.50.2.1 New functions and features

- Added functions `system.gamsrelease` and `gamsrelease`
- Added function `poly(x,a0,a1,a2,a3..) = sum(i=0 to n, ai*x^i)`
- `Appendout (ao)=2` will redirect the listing file to stdout. This feature is not available when running under the IDE.

3.50.2.2 GAMS IDE

- Open in new window creates now a single new window when multiple files have been selected
- Spelling menu entry added to Edit menu
- Gdxviewer can write a single symbol and all symbols to an Excel file

3.50.2.3 GAMS Model Library

- New models
  - eps-Constraint Method for Multiobjective Optimization (**epscm**)
  - Kissing Number Problem using Variable Neighborhood Search (**knp**)
  - Termination routine to ensure solvers stay with resource limit (**schulz**)
  - How to test for a GAMS version (**version1**)

3.50.3 Solvers

3.50.3.1 AlphaECP

- A new solver called GAMS/AlphaECP for mixed integer non-linear problems has been added to the GAMS solver portfolio. AlphaECP is an implementation of the Extended Cutting Plane method by Tapio Westerlund and Toni Lastusilta from Abo Akademi University, Finland. The GAMS/AlphaECP solver requires the presence of a licensed MIP solver only.

3.50.3.2 BARON

- New libraries (version 7.8) with improved local search routines.

3.50.3.3 COIN-OR Solvers

- New libraries for CoinCbc, CoinGlpk, and CoinIpOpt. The experimental MINLP solver CoinBonmin has been introduced for the GAMS Windows and Linux systems. Bonmin (Basic Open-source Nonlinear Mixed INteger programming) provides an implementation of an NLP-based branch-and-bound algorithm, an outer-approximation decomposition algorithm, an implementation of Quesada and Grossmann's branch-and-cut algorithm, and a hybrid outer-approximation based branch-and-cut algorithm.

3.50.3.4 CONOPT

- New libraries (version 3.14r).

3.50.3.5 CONVERT

- Added option 'NLP2MCP' that reformulates a non-integer program as a mixed complementarity problem (MCP)
- Added option 'AmplNLC' that converts a model into C code for evaluating objectives, constraints, and their derivatives.
- Added option 'Jacobian' that creates a GDX file containing the basic model data (matrix, initial point, evaluation of constraints at initial point and bounds)
- Added option 'LindoMPI' that creates an MPI file which is readable by Lindo.

3.50.3.6 CPLEX

- New libraries (version 10.20).

3.50.3.7 LINDOGLOBAL

- The Global Optimization Solver (GAMS solver name is LINDOGLOBAL) from Lindo Systems, Inc. has been added to the GAMS solver portfolio. LINDOGLOBAL finds proven optimal solutions to non-convex mixed integer non-linear problems. The LINDOGLOBAL solver requires a license for GAMS/CONOPT. The size of a model solved by LINDOGLOBAL is limited to 2,000 equations and 3,000 variables.

3.50.3.8 MOSEK

- New libraries (version 4.0.0.60)

3.50.3.9 XPRESS

- New libraries (version 17.10.12)

3.50.3.10 Solver/Platform Availability Matrix

3.51 22.4 Major release (February 12, 2007)

3.51.1 Acknowledgements

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Michael Ferris, Gary Goldstein, and Randy Wigle.

3.51.2 GAMS System

3.51.2.1 GAMS

- Compressed and Encrypted Input Files: Facilities to create and use encrypted and compressed input files have been added. The three new commands: `$compress`, `$decompress` and `$encrypt` are the respective utilities. For more details consult the User's Guide.
- `NonNegative Variable`: This new keyword is a synonym for `Positive Variable`.

3.51.2.2 GAMS IDE

- Added spelling checker
- Added search options for gdxviewer

3.51.2.3 Model Library

- **new models** (cefiles, encrypt) demonstrate the new GAMS feature of compressing and encrypting GAMS input files.

3.51.2.4 Testlib Library

- **new models** to test new features

3.51.3 Solvers

3.51.3.1 COIN

- new libraries and solver from the `GAMSlinks` project hosted at COIN-OR.
  - CoinGlpk: New COIN-OR and Glpk 4.9 libraries.
  - CoinCbc: New COIN-OR and Cbc 1.1.0 libraries.
  - CoinIpopt: A new interior point NLP solver from COIN-OR.
  - The GAMSlinks project (announced January 3 by Stefan Vigerske) was created to develop links between GAMS and the COIN-OR solvers.

3.51.3.2 CPLEX

- New libraries (version 10.1.1).

3.51.3.3 KNITRO

- New libraries (version 5.1).

3.51.3.4 MOSEK

- New libraries (version 4.0.0.59)

3.51.3.5 XPRESS

- New libraries (version 17.10.08)

3.52 22.3 Major release (November 27, 2006)

3.52.1 Acknowledgements

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz, Michael Ferris, Josef Kallrath, Antti Lehtila, Scott Malcolm, and Renger van Nieuwkoop.

3.52.2 GAMS System

3.52.2.1 GAMS

- GAMS Grid Facility: The GAMS language has been extended to take advantage of systems with multiple CPUs and High Performance Computing Grids. New language features facilitate the management of asynchronous submission and collection of model solution tasks in a platform independent fashion. A simple architecture, relying on existing operating system functionality allows for rapid introduction of new environments and provides for an open research architecture. For details please consult our documentation.
- `$LoadDC`: Enhanced $load with domain checking. Any domain violations will be reported and flagged as compilation errors. In contrast, $Load ignores all domain violations and loads only data that meets the domain restrictions.
- A new GAMS parameter, WorkSpace allows initializing all modelname.workspace attributes.
- New functions:
  - arccos, arcsin, tan
  - arctan2(y,x): 4-quadrant arctan function, like the atan2(y,x) in the C math library or Matlab, or Mathematica's ArcTan[x,y]
  - binomial(x,y): generalized binomial function, like Mathematica's Binomial[x,y]
  - GamsVersion: returns the current gams version, for example 146 for this release
  - HeapFree: free GAMS heap space in Mbytes
  - HandleStatus, HandleDelete, HandleSubmit, HandleCollect: Grid facility functions, see the grid facility document for details
  - Sleep(sec): suspends GAMS execution for sec seconds, where sec has millisecond resolution.
- GAMS solvers and other processes now inherit their priority from GAMS/Base or GAMSIDE: this is useful on heavily loaded machines where you want GAMS to run with below-normal priority.

3.52.2.2 GAMS IDE

- Removed size limit for the capacity of the clipboard for .lst viewer
- Added a shape graph type (rectangle and ellipse) and multi-line parametric graph type
- Added more context menus
- Option to open .lst file in editor
- Switching tabs or switching application will check for files that changed
- When gamside.ini is not found in the 'My Documents\gamsdir' directory, new directories and an initial project will be created. The IDE will start maximized, and the edit window and process window position and size are initialized.
- Added a GUI editor for solver option files. The editor also provides access to the solver specific help file.
- When searching for text in files (Find in Files) the search window remains open to make switching between editor and search results faster.

3.52.2.3 File Compression

- Every program creating a GDX file (e.g. gdxxrw or gams) will look at the environment variable GDXCOMPRESS; if value is not zero, data will be written compressed.
- Compressed workfiles can be created with parameter xsave.
- Decompression of GDX or workfiles is handled automatically.
- GDX and workfiles can exceed 2GB in size.
- The new utility gdxcopy allows conversion between different formats. For example, a compressed GDX file can be converted so it is understood by older GAMS systems. Of course, all file formats (GDX and workfile) can be read by a current GAMS system.

3.52.2.4 GDXMERGE

- Added ability to process named identifiers only (id=id1, id=id2)
- Added ability to process very large symbols separately by specifying a memory threshold to avoid memory issues.

3.52.2.5 GDXXRW

- String explanatory text for sets is now quoted. (Text starting with '=' created problems.)

3.52.2.6 Testlib Library

- **66 new models**, including
  - New solver-specific models/suites: 24 models, 21 suites
  - Tests for new intrinsic functions and utilities

3.52.3 Solvers

- The processing of solver option files has been centralized/standardized. This allows for a uniform way of documenting and working with solver and options. The GAMS IDE provides a utility for visual processing of options.

3.52.3.1 BARON

- New libraries (version 7.7) with improved range reduction for monomial functions.

3.52.3.2 CONOPT

- New libraries (version 3.14q) with some minor bug fixes and performance improvements.

3.52.3.3 CPLEX

- New libraries (version 10.1) with increased performance when solving models with indicator constraints.

3.52.3.4 KNITRO

- New libraries (version 5.0) with improved efficiency and robustness, especially in the active-set algorithm.

3.52.3.5 MOSEK

- New libraries (version 4.0)
  - There is a parallel extension (option MSK_IPAR_INTPNT_NUM_THREADS) for the interior solver. Moreover, it is possible to solve a problem concurrently with several of MOSEK's algorithms. The parallel extension comes free of charge.
  - Warm start capability for the mixed integer optimizer (option MSK_IPAR_MIO_CONSTRUCT_SOL) has been added.
  - A dual simplex optimizer has been added and the overall performance of the Simplex optimizers has been significantly improved. Moreover, an automatic (and transparent) dualizer has been added to the simplex optimizers.
  - The presolve has been improved especially for large stochastic optimization problems.

3.52.3.6 PATH

- New libraries (version 4.6.07) with some minor bug fixes and performance improvements.

3.52.3.7 SNOPT

- New libraries (version 7.2-4) with some minor bug fixes and performance improvements.

3.52.3.8 XA

- New libraries (version 15.07) with improved presolve and new MIP heuristics.

3.52.3.9 XPRESS

- New libraries (version 17.10)
  - GAMS/XPRESS now supports multiple threads for MIP and Barrier runs.
  - New option (loadmipsol) supports passing an integer feasible point to the MIP solver.
  - Improved factorization speed - yields faster primal and dual algorithms.
  - Improved MIP performance (via presolve and heuristics, e.g.)

3.52.3.10 Solver/Platform Availability Matrix

3.53 22.2 Minor release (April 21, 2006)

3.53.1 Acknowledgements

We would like to thank all of our users who have reported problems and made suggestions for improving this release. In particular, we thank Wolfgang Britz (Bonn University), Paritosh Desai (DemandTec), Michael Ferris (UW-Madison), Edgar Ramirez (at hotmail.com), and Rich Roberts (SRS Technologies).

3.53.2 GAMS System

3.53.2.1 GAMS

- The limit on nonlinear instructions in a single block has been raised from 16 million to 64 million instructions.
- Performance improvements for very large and complicated loop structures.
- International characters in file and path names are now handled correctly.

3.53.2.2 GAMS IDE

- GDX data browser is faster and can sort indices by name vs. entry order
- A symbol shown in the GDX data browser can be written to an Excel file

3.53.3 Solvers

3.53.3.1 CONOPT

- New libraries are included which address minor fixes

3.53.3.2 CPLEX

- New libraries (version 10.0.1, a maintenance release)

3.53.3.3 LGO

- New libraries
  - The built-in stochastic searches have been improved.
  - Some internal limits were increased to allow larger models to be solved.

3.54 22.1 Major release (March 15, 2006)

3.54.1 GAMS System

3.54.1.1 Relaxation of discrete variables (.prior=Inf)

- The priority attribute of a discrete variable can be used to relax a specific variable instance. The priority attribute `.prior` establishes in what order variables are to be fixed to integral values while searching for a solution. Variables with a specific `.prior` value will remain relaxed until all variables with a lower `.prior` values have been fixed. Setting the `.prior` value to +inf will relax this variable permanently. This relaxation is done independent of the model attribute `.prioropt`.

  This feature is useful in solving difficult discrete models. The Model Library model "Linear Recursive Sequence Optimization Model" (lrs.gms) illustrates the use of this feature that specifies that only the first n variables of k are binary, whereas the remaining ones are fractional. This is simply expressed as follows:

```
SET t time horizon /1*350/, f(t) first N steps /1*48/;
Binary Variable k(t);
...
k.prior(t) = inf; k.prior(f) = 1;
```

3.54.1.2 Derived Variable and Equation Attributes (.slacklo,.slackup,.slack,.infeas)

- Calculations involving slacks and infeasibilities have been simplified with four new derived attributes. These attributes are defined on equations and variables as follows:

```
x.slacklo = max(x.l-x.lo,0);
x.slackup = max(x.up-x.l,0);
x.slack = min(x.slacklo,x.slackup);
x.infeas = - min(x.l-x.lo,x.lu-x.l,0);
```

  Note that the slacks or infeasibilities will always be zero or positive.

3.54.1.3 Enhanced Model List

- The entries of a model list can now contain previously defined models. For example:

```
Model one / e1,e2,e3 /
two / one, e4 /
three / two, e5 /;
```

  The model two will now contain equations `e1`,`e2`,`e3`,`e4`. The model three will contain all of model two plus equation `e5`.

3.54.1.4 New model type

- `RMPEC`: The model type `MPEC` now allows discrete variables. The Relaxed Mathematical Program with Equilibrium Constraints (RMPEC) model relaxes the discrete variables as with RMIP and MIP models.

3.54.1.5 Enhanced $IF/IFI statement:

- A check for the existence of a directory has been added to the $if statement.

```
$IF DEXIST directory command
$IFI DEXIST directory command
```

3.54.1.6 GDX Utilities

GDXDIFF

- When a key value pair exists in one file and not in the other we use Eps to determine if the difference should be reported.

GDXDUMP

- All unique elements in the output are now quoted to avoid problems with reserved words etc.

GDXMERGE

- New options introduced to merge only specified identifiers, to merge very large data sets, and to read parameters from a file (@filename).

GDXXRW

- Fixed problem with merging when only the top-left corner of the range was specified.

3.54.1.7 GAMS IDE

- Added a viewer for lst files. In contrast to the editor, this viewer does not load the complete file in memory.
- Added a navigation tree for the lst file.
- Added the option to generate charts using a gdx file.
- All viewers, except the process window, are now organized as tabbed windows.
- The gdxviewer remembers that last symbol viewed for a gdx file and will select the symbol when opening the file.
- Expanded format for library file to use other entries in the library.

3.54.1.8 GAMS Model Library

- **new models**
  - IDE charting facility (**chartdat**)
  - Cplex option feasopt (**feasopt1**)

3.54.1.9 Testlib Library

- **new models**
  - Tests for new GAMS intrinsic functions `NCPVu***`
  - Tests for GDXdiff utility

3.54.1.10 McCarl GAMS User's Guide

- Extensive internal changes to allow better distribution formats
- Now distributed as a single .chm file and a single .pdf file with fully linked topics

3.54.2 Solvers

3.54.2.1 BARON

- XA can now be used as an LP solver inside BARON.
- For the LP solvers Cplex and XA the user can specify the LP algorithm (primal/dual simplex, barrier) using the BARON option `lpalg`.
- The user can control BARON's node selection scheme with the option `nodesel`. Available strategies include best bounds, DFS, minimum infeasibilities, and BARON's own mix.

3.54.2.2 CONOPT

- New libraries (version 3.140)
  - The scaling method has been improved to work better with models with small levels and large derivatives (e.g. sqrt(x) for x close to 0).
  - Minor problems with previous versions corrected.

3.54.2.3 CPLEX

- New libraries (version 10.0)
  - Dropped Platforms: ILOG dropped support for CPLEX 10.0 on the SGI platforms. Also support for Linux with glibc2.2 was dropped but will hopefully be available when GAMS 22.1 will be released. GAMS 22.1 for dropped platforms ships older versions of CPLEX.
  - Solution Polishing: Solution Polishing is appropriate for finding the best solutions to complex and difficult MIP models within a specified time. Solution Polishing is used to improve the best solution at the end of the branch-and-cut process if optimality has not been proven.
  - MIP Starts: The advanced restart capabilities of CPLEX have been improved to utilize initial solutions, partial solutions and partially correct solutions. If the user specifies values for only a portion of the discrete variables, CPLEX 10.0 will attempt to fill in the missing values or correct the wrong values in a way that leads to an integer-feasible solution, potentially reducing the time to solve the problem.
  - Infeasibility analysis tool: CPLEX provides an automatic approach to find the best feasible alternative to an infeasible model. This approach is turned on by an option called `FeasOpt` (for *feasible optimization*). With the `FeasOpt` option CPLEX accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section.
  - For details check the GAMS/CPLEX 10 Release Notes.

3.54.2.4 DICOPT

- New option infeasder: It allows for linearization of infeasible NLP subproblems.

3.54.2.5 Examiner

- Default behavior changed: if no action is chosen, Examiner prints a warning message and defaults to "examineInitPoint yes"
- Adjusted to handle new RMPEC model type. Currently MPEC models are not allowed.
- Report format for examination of points changed:
  - better labeling of the point being examined - consistent with the naming scheme and options used to choose the examination
  - use GAMS notation (e.g. `x.l(seattle,topeka), f.m(supply)`) to specify rows/columns
  - add option 'showSlacks' to print slacks/differences explicitly when the difference is small compared to the values themselves

3.54.2.6 XPRESS

- New libraries (version 16.10.03)
  - The dual pricing algorithm has been improved. For some difficult LP problems this has resulted in an order of magnitude speed improvement.
  - The speed and stability of the simplex factorization has been improved.
  - The barrier algorithm now requires less memory.
  - The cut generation has been improved.

3.55 22.0 Major release (August 01, 2005)

3.55.1 New platforms supported

- 64 bit Windows (aka Windows 64-Extended, Windows EM64T) version introduced. Windows licensing includes both 32 and 64 bit systems at a single platform cost. Some solvers exist only in 32 bit mode and are included as such on the 64 bit version. See the solver platform matrix.

3.55.2 Updated build for the Linux platform

- Previous 32-bit Linux GAMS Distributions (e.g. 21.X) were built using only an older Linux machine with version 2.2 of the GNU C Library. This distribution is compatible with both older and newer Linux systems. The installer for this software is located in the systems/lnx/lnx2.2 directory of the CD. The build code for this software is LXI (displayed on the solver audit lines, etc). This build is included in the current CD as it was before.

- This distribution includes a new 32-bit Linux build (build code LX3) that parallels the previously-existing LXI build. The LX3 system is built on a newer Linux machine with version 2.3 of the GNU C Library, and is not compatible with older systems using GNU LIBC 2.2. The installer for this software is located in the systems/lnx/lnx2.3 directory of the CD.

- Consider the following when choosing which Linux build to install:

  – Most Linux users will be using a system with GNU LIBC 2.3 and should use the newer build. You can check your LIBC version by running it. On my machine, I do "/lib/libc.so.6" but be aware your version number may vary.

  – These are builds of the same source, using different compilers. The "older" build uses older compilers, not older source. In cases where different vendor libraries are required, the same comment applies.

  – The different builds use a common GAMS license file.

  – While there is no scheduled end-of-life for the older build we expect that our algorithm vendors will stop supporting it at some point. When this happens, we can only ship updated versions of the new build.

  – If you wish to install both versions, install them in separate directories.

3.55.3 GAMS System

3.55.3.1 GDX Utilities

GDXDIFF

- Modified the comparison routine that determines if two numbers are equal wrt. an absolute or relative tolerance.

3.55.3.2 GAMS IDE

- Added option to save a file in Unix format.
- Added option to save and email the current file.
- Added option to launch Windows Explorer showing directory of the current file with the current file selected.
- gamsdir is now a sub-directory of 'My documents'. gamside.ini file now stored in the gamsdir directory.
- Reload file now also works for GDX files and RefFiles (in addition to regular source files).
- When starting the IDE or opening a project, a scan is made for GAMS temporary directories which can be deleted. This option was also added under the Utilities menu.
- Pressing F1 for help was broken.
- `$ONECHO`/`$OFFECHO` work now like `$OFFTEXT`/`$ONTEXT` for syntax coloring.

3.55.4 Solvers

3.55.4.1 CPLEX

- New libraries (version 9.1)
  - Option mipstart: It is no longer necessary to provide a compete integer solution as the starting point.
  - Option fraccuts: Settings of 1 or 2 will cause Gomory cuts to be generated for MIQCP models.
  - New Option lbheur: When set to 1 (default is 0), CPLEX will use a local branching heuristic to try to improve new incumbents.

3.55.4.2 KNITRO

- New libraries are included which address minor fixes.

3.55.4.3 XA

- New libraries are included which address minor fixes.

3.55.4.4 XPRESS

- New libraries are included which address minor fixes.

3.56 21.7 Major release (April 01, 2005)

3.56.1 Acknowledgements

We would like to thank all of the users who have reported bugs or made suggestions in improving this release.

3.56.2 New platforms supported

- 64 bit x86_64 (Linux) version introduced. Linux licensing includes both 32 and 64 bit at a single platform cost. Some solvers exist only in 32 bit mode and are included as 32 bit on the 64 bit version. See the solver platform matrix below.
- Macintosh PowerPC (Darwin) version introduced.

See our sales page for available solvers by platform.

3.56.3 GAMS System

3.56.3.1 Model Library

- Several new models have been added. See the **GAMS Model Library** for details.

3.56.3.2 New Workfile Format

- now a single file (G00 file extension). Files can be read that were created on different platforms. File is packed and only approx 50% of previous file size. Read and write is also substantially quicker.

3.56.3.3 Testlib Library:

- Updates to quality model
  - take advantage of solver updates & fixes
  - added `--QUICK=1` option to skip slow tests
  - test all solvers, using `system.solverNames` and filter list
  - allow skips of models in solver-specific test suites
- New models
  - convert solver test suite
  - new cmex & MIP tests, test new workfile format save/restart

  See the **testlib library** for details.

3.56.4 Solvers

3.56.4.1 BARON

- New libraries (version 7.2.5) which address minor bug fixes and enhancements

3.56.4.2 BENCH

- promoted from beta solver.

3.56.4.3 GAMS/AMPL Link

- a GAMS/AMPL link has been introduced.

3.56.4.4 GAMS/LINGO Link

- a GAMS/LINGO link has been introduced.

3.56.4.5 MOSEK

- New libraries (version 3.2) which address minor bug fixes and enhancements.

3.56.4.6 NLPEC

- Relaxed the test for complementarity of the solution proposed by the reformulated model to work better for poorly-scaled models

3.56.4.7 XA

- update to documentation (nodelimit, log output)
- numerous library updates to fix outstanding issues
- improved handling of solver status 4=TERMINATED BY SOLVER

3.56.4.8 XPRESS

- deal with "recoverable errors" on return from barrier method
- minor improvements and workarounds for library quirks in link

3.56.4.9 Solver/Platform Availability Matrix

3.57 21.6 Minor release (January 26, 2005)

3.57.1 Acknowledgements

We would like to thank all of the users, particularly Josef Kallrath, who have reported bugs or made suggestions in improving this release.

3.57.2 Solvers

3.57.2.1 BARON

- New libraries are included. Several new features which are documented in the BARON solver manual.

3.57.2.2 CONVERT

- New model instance format CoinFML, Coin style XML model file.
- LINGO format updated to include Version 9 features

3.57.2.3 COIN

- CoinGlpk: New COIN and GlpK 4.7 libraries.
- CoinSbb: This is now called CoinCbc (Coin Branch & Cut). The name CoinSbb will remain for this and the next distribution.
- Three new cuts classes: Clique, Flow Cover, Mixed Integer Rounding and all new libraries.

3.57.2.4 KNITRO

- New libraries are included which address minor fixes.

3.57.2.5 XPRESS

- New libraries are included which address minor fixes.

3.57.3 GAMS System

3.57.3.1 Model Library

- licememo: Generate your own license memorandum and solver matrix with this model.
- deploy: Generate a minimal deployment GAMS system with necessary components only.

3.57.3.2 Testlib

- 26 new quality assurance tests have been added to the suite.

3.57.3.3 Minor enhancements

- `$onecho` and `$onput` have new variants to specify verbatim or substitutions. The new names are: `$onechoV`, `$onechoS`, `$onputV`, `$onputS`.
- Manipulation of system environment variables and prefixing of the search path:

```
$SetEnv name value
$DropEnv name
$PrefixPath value

%SysEnv.name%

$if SetEnv name
```

3.57.3.4 fml2gms

- converts a CoinFML style XML file into GAMS (courtesy of Leo Lopes, University of Arizona). An example showing how to convert from GAMS to CoinFML format and back to GAMS is shown below:

```
> gamslib trnsport
> echo coinfml > convert.opt
> gams trnsport optfile=1 lp=convert
> fml2gms coinfml.xml
> gams fml2gms.gms
```

3.57.4 Documentation

3.57.4.1 Bruce McCarl User's Guide

- accessing the guide through the GAMS/IDE, will now launch the PDF file directly instead of through a web browser. This is due to possible incompatibility issues with Windows XP Service Pack 2 (securtiy settings). To launch via the web, users can go to File > Options and check the box "Launch docs using Explorer."

3.58 21.5 Minor release (November 11, 2004)

Distribution 21.5 includes newly available solver libraries and minor enhancements in the GAMS system, as well as introduces the interior-point NLP solver KNITRO from Ziena Optimization.

3.58.1 Acknowledgements

We would like to thank all of the users who have reported bugs or made suggestions in improving this release, especially Richard Roberts for catching an IDE bug.

3.58.2 Solvers

3.58.2.1 CONOPT

- New libraries (14e) are included which address minor fixes.

3.58.2.2 CPLEX

- New libraries (9.0.2) are included which address minor fixes.

3.58.2.3 KNITRO

- New interior-point nonlinear programming solver added to the GAMS NLP solver portfolio.

3.58.2.4 MOSEK

- New libraries (3.1) are included which address minor fixes.

3.58.2.5 XA

- New libraries are included which address minor fixes.

3.58.2.6 XPRESS

- New libraries are included which address minor fixes.

3.58.3 GAMS System

3.58.3.1 Model Library

- A model showing a successive recalibration algorithm for solving general equilibrium models has been added (**decomphh.gms**).
- A model illustrating numerical integration using trapezoidal approximations for Herves (transposable element) activity calculations is included (**herves.gms**).

3.58.3.2 Testlib

- 27 new quality check models have been added. See the **Testlib** page for details.

3.58.3.3 New functions

- sinh, cosh, tanh

3.58.3.4 $ Dollar control options

- `$on/offVerbatim`
  - The `$on/offVerbatim` commands are used in conjunction with the GAMS parameter `DUMPOPT` to suppress the input preprocessing for input lines that are copied to the dmp file. This feature is mainly used to maintain different versions of related models in a central environment.
    The `$on/offVerbatim` commands are only recognized for `DUMPOPT >= 10` and apply only to lines in the file the commands appeared.
    The use of $goto and `$on/offVerbatim` are incompatible and may produce unexpected results.
- `$on/offPut`
  - This option simplifies the writing of put statements that copy text. Previously, to copy text into a put file one may have to write many lines similar to the ones below:

```
put 'text A with a double quote "' "text A with a single quote '" /
put 'text B with a double quote "' "text B with a single quote '" /
put 'text C with a double quote "' "text C with a single quote '" /
```

This may be cumbersome. The use of `$on/offPut` will result in the same text written to the put file without having to specify the 'put' explicitly for each line and allows you to specify mixed quotes in a single statement. Consider the following example:

```
file fx;
put fx 'text with " and ' "' required a split of the text" /
$onput
With the new on/off put we can freely
mix and match ' and " characters.
$offput
;
```

It also works outside of the put statement:

```
$onput
It even works outisde the put statement,
as long as a put file is assigned.
$offput
```

- `$setDDlist`
  - This statement catches misspelled 'double dash' GAMS parameters. For example, in the program below (ein.gms), the only 'double dash' options are 'one', 'two', 'three' and 'four' (note the use of the string macro for `%two%` automatically makes it part of the allowed list of double dash parameters):

```
$if NOT set one $set one default value
display '%two%';
$setddlist three four
```

The following GAMS invocation will cause an error since `--five` is not a valid 'double dash' option.

```
> gams ein.gms --two=twovalue --five=20

1 Display 'twovalue'
**** --five=20

3 $setddlist three four $620
```

3.58.3.5 Other new features

- alias(∗,u1,u2,..)
  - The universe is assumed to be ordered and ordered operators like lag, leads and ORD can be applied.
- Faster execution of GAMS statements using set(s) containing a single element.
- SortedUels(∗,∗)
  - A new predefined tuple to report sets in sorted order. For example, to write in sorted order:

```
alias(*,u);
loop(SortedUels(u,i),
  put / i.tl i.te(i) );
```

3.59 21.4 Major release (September 06, 2004)

3.59.1 Acknowledgements

We would like to thank all of the users who have reported bugs or made suggestions in improving this release.

3.59.2 GAMS System

3.59.2.1 Model Library

- The model library keeps on growing! A model that implements a column generation scheme for the cutting stock problem (cutstock) has been added. There are two examples that demonstrate the multiple solution feature in BARON (mhw4dxx and bchfcnet).
- The model library also has the new category "GAMS Tools". Models in this category show how to use tools designed to help modelers in their daily work. Three models (awkqap, seders, and awktsp) demonstrate the routine process to take a raw file from a web site and manipulate the text file into a format suitable to be read by GAMS using Unix-style tools SED & AWK.
- Five new models (mingamma, quantum, procmean, mlbeta, and mlgamma) have been added to demonstrate the new statistical functions (discussed below).

3.59.2.2 Testlib

- 33 new models were added to the test library.
- Addition of solver-specific test suites. The models in a test suite are run not for all solvers but for the set of solvers associated with that particular suite.
- Most of the new models test features in GAMS/Base.
- A series of new models tests MCP syntax and matching rules.

3.59.2.3 Windows Setup.exe

- A new Windows setup.exe installation program for GAMS is used adding new functionality and a more user friendly interface.
- The McCarl GAMS User Guide is now part of the Windows installation (for UNIX platforms this must still be installed manually).

3.59.2.4 GDX Utilities

- GDXrank, a standalone sorting utility for GDX files, is now part of the distribution (all platforms)
- A wrapper for GDXrank, called rank.gms, written by Tom Rutherford, is included with the distribution and can be called using the $LIBINCLUDE directive.

3.59.2.5 GAMS IDE

- The IDE recognizes https as www string
- The menu Help | About shows current license file
- If there is a GAMS license file on the ClipBoard, selecting Help | About will prompt to write the ClipBoard to a license file.
- Library files can now have multiple files with a numeric extension; previously,it was the first file only.
- A library file can have have an extension of '.ignore' which will be removed so we can protect '.zip' files.
- After running gdxdiff, an empty difference file will not be shown.

3.59.2.6 Additional functions

- gMilliSec(DayTime): milli seconds of a DayTime number
- MaxExecError: a read/write access to the ExecError values
- TimeElapsed: elapsed time in seconds since the start of a GAMS run
- Gamma(a): Gamma function (DNLP)
- LogGamma(a): Log Gamma function
- GammaReg(x,a): Regularized gamma function
- Beta(a,b): Beta function (DNLP)
- LogBeta(a,b): Log Beta function
- BetaReg(x,a,b): Regularized Beta function
  The definitions and notation for incomplete and regularized gamma and beta functions are not consistent, For example, note the differences with the definitions used in Mathematica:

```
GAMS Mathematica

Gamma(a) Gamma[a]
LogGamma(a) LogGamma[a]
GammaReg(x,a) GammaRegularized[a,0,x]
Beta(a,b) Beta[a,b]
LogBeta(a,b) Log[Beta[a,b]]
BetaReg(x,a,b) BetaRegularized[x,a,b]
```

3.59.2.7 Speed-up for expressions containing constant indices or indices that are not in the natural order

- Speed-up for expressions containing constant indices or indices that are not in the natural order
  - The option sys11 controls this new feature. When we use option sys11=1, GAMS will execute without speedup features as in previous versions; using option sys11=0 will cause GAMS to use procedures that execute some statements faster at the cost of increased memory use (This is now the default value). The following GAMS code illustrates some of the speedups.

```
Sets i / i1*i700 /
     j / j1*j700 /
     k / k1*k500 /
     ik(i,k);

Parameters aij(i,j) bji(j,i), cjk(j,k), dij(i,j);

ik(i,k)$(uniform(0,1) < 0.01) = yes;
aij(i,j) = uniform(0,1);

bji(j,i) = aij(i,j);

cjk(j,k) = sum(ik(i,k), aij(i,j));
dij(i,j) $(aij(i,'j700') > 0.5) = bji(j, 'i700')
```

  A comparison of the execution times in seconds:

```
Assignment sys11=1 sys11=0
Bji 2.7 0.36
Cjk 25.0 2.38
Dij 4.7 0.39
```

– Faster execution of lag / lead operators. This improvement is visible for large sets only as illustrated in the following GAMS code:

```
Set i /i1*i2000/,
    j /j1*j20000/;

Parameter Ai(i),Bj(j);
Ai(i - 1) = Ord(i);
Bj(j - 1) = Ord(j);
```

A comparison of execution times in seconds:

```
Assignment previous current
Ai 0.02 0.0
Bj 3.2 0.0
```

3.59.2.8 Embedded set text and parameter values

- The `$on/offembedded` option enables the use of embedded values in parameter and set data statements. For sets, the final text is concatenated with blank separators. For example, the element texts for the set and j will be identical:

```
Set i(k,l) / a.a 'aaaa cccc dddd', b.a 'bbbb cccc dddd' /;
$onembedded
Set j(k,l) / (a aaaa, b bbbb).(a cccc) dddd /;
```

For parameters, the final value will be the product of the embedded values. If no value is specified, a value of 1 is assumed. For example, the values for x and y will be the same:

```
Parameter x(k,l) / a.a 24, b.a 12, c.a 4, c.b 4, d.a 6, d.b 6 /;
$onembedded
Parameter y(k,l) / (a 2, b).(a 3) 4, (c 2,d 3).((a,b) 2) /;
```

3.59.2.9 New suffixes for functions

- New suffixes Grad and Hess have been introduced to get exact point derivatives from any function. These function suffixes are mainly intended for future testing of functions and cannot be used in equations.
- The first argument gives the position of the element of the Hessian or gradient element desired in the form i or i:j, where i is the row element and j the column element.
- The symbol ':' is used to separate the element position specification from the function argument list.
- For example, the following will return the second derivative for the second and fourth argument, where 1,2,3,4,5 are the normal function arguments:

```
h = EDist.hess(2:4:1,2,3,4,5);
```

If the needed element position index is one, we can drop the argument as shown below:

```
g = exp.grad(1:5) or g = exp.grad(5);
h = log.hess(1:1:3) or h = log.hess(3);

hess(i,j) = betareg.hess(ord(i):ord(j):expr1,expr2,expr3);
```

3.59.2.10 Some new minor and exotic features

- New `%system.xxx%` and `system.xxx` for put statements
  - `system.date1`: different date format: Feb 04, 2004
  - `system.tab`: inserst a tab character
- New File attribute
  - `file.silent`: will suppress the logging of put files
- New `$IF` option allows testing for existing solver at compile time
- `$IF SOLVER` command
- `$if NOT solver baron $goto nobaron`
- Underflow Control
- Release 21.0 introduced new math routines that work over a much wider numerical range than the older systems. The pre 21.0 systems rounded small function return values (less than 1e-30) to zero.
- Some older models may use this rounding to zero feature and will now give slightly different results.
- The new GAMS parameter ZeroRes=real allows you to change the threshold value for internal rounding in the GAMS calculation and the GAMS parameter ZeroResRep=1 will cause GAMS to issue warnings whenever such a rounding occurs.

3.59.3 Pricing

3.59.3.1 Global Packages

- Similar to the NLP Packages (NLP-1 and NLP-2), GAMS now offers Global Packages: If you buy at least two of the Global solvers BARON, LGO, and MSNLP, the prices are reduced by 25%. Please check our pricelist or contact sales@gams.com.

3.59.4 Solvers

3.59.4.1 BARON

- New libraries (version 7.2)
- Branch-and-cut implementation (available only when using CPLEX as the LP solver)
- Automatic exploitation of convexity
- Modeling construct permits user to supply convexity information to the solver
- Modeling construct permits user to have relaxation-only constraints
- Improved local search implementation for large-scale models
- GDX dumps can be used to provide output for the K best solutions
- Several algorithmic fixes and improvements

3.59.4.2 COIN

- GAMS has added a link to the Computational Infrastructure Operations Research (COIN-OR). The COIN-OR project is an initiative to spur the development of open-source software for the operations research community.
- The GAMS/COIN-OR link allows GAMS users to connect their customized solution approaches using the COIN-OR Open Solver Interface (OSI) in a seamless manner.
- The GAMS/COIN-OR Link for LP and MIP problems is available in source and free of charge to any licensed GAMS system.
- Potentially all solvers connected to the COIN-OR/OSI can be made available through the GAMS/COIN-OR link. Currently,
  - CoinGlpk: Gnu Linear Programming Kit
  - CoinSbb: simple branch and bound, a branch and cut code
- are included in the latest Windows and Linux distributions. Please visit the GAMS/COIN-OR web page at https://github.com/coin-or/GAMSlinks for details.

3.59.4.3 CONOPT

- New libraries are included which address minor fixes and improvements.

3.59.4.4 MINOS

- New libraries (MINOS 5.51 June 2004)
- Fixed issues with scaled/unscaled infeasibilities/nonoptimal

3.59.4.5 MSNLP

- MSNLP (Multi-Start NLP) is another stochastic search algorithm from Optimal Methods, Inc for global optimization problems. Like it's bigger brother OQNLP, MSNLP uses a point generator to create candidate starting points for a local NLP solver. Algorithm performance depends strongly on the starting point generator. MSNLP implements a generator creating uniformly distributed points and the Smart Random Generator. This generator uses an initial coarse search to define a promising region within which random starting points are concentrated. Two variants of Smart Random are currently implemented, one using univariate normal distributions, the other using triangular distributions. MSNLP also comes with the local NLP solver LSGRG. MSNLP is available in the Global Packages.

3.59.4.6 NLPEC

- NLPEC has been promoted from beta solver status and has a write up in the solver documents.
- The NLPEC solver for MPECs automates the process of model reformulation. NLPEC reformulates the original MPEC as an NLP (using one of 23 different reformulation strategies), solves the resulting NLP, and translates the results back for return as an MPEC solution.

3.59.4.7 PATHNLP

- PATHNLP has been promoted from beta solver status. The PATHNLP solver suitable is for NLP programs. PATHNLP solves an NLP by internally constructing the Karush-Kuhn-Tucker (KKT) system of first-order optimality conditions associated with the NLP and solving this system using the PATH solver for complementarity problems.
- Some improvements from the beta version include:
  - In some cases where PATHNLP fails to find a solution, it can now return a feasible point where before it would return no solution information whatsoever.
  - Information about the Hessian and it's inverse is now available.

3.59.4.8 OQNLP

- OQNLP's merit and distance filters, which are respondible for starting the NLP solver at a small fraction of the candidate starting points, while still finding the global solution to most problems, have been improved. The *dynamic merit filter logic* and the *basin overlap fix* provide mechanisms for decreasing the radii of some attraction basins, focusing on those which reject points most often and those which overlap. These dynamic filters lead to a substantial improvement in OQNLP's ability to obtain a global optimum, with some increase in the number of solver calls. The OptQuest point generator has been supplemented by two new point generators (see MSNLP).

3.59.4.9 XPRESS

- Updated to use XPRESS 2004 libraries - highlights include:
  - MIP heuristics
  - Lift-and-project cuts to give improved bound information
  - Strong branching
  - Extensive benchmarking & resultant performance improvements

3.60 21.3 Major release (January 19, 2004)

3.60.1 GAMS System

3.60.1.1 Model Library

- Models from the "Handbook of Computible General Equilibrium Modeling", University of Tokyo Press, Tokyo (to appear) have been added to the model library.
- Several QCP models have been added
- A series of models that illustrate the BCH (Branch-and-Cut-and-Heuristic) Facility have been added

3.60.1.2 TESTLIB

- New library of models developed for testing and quality control:
  - Solver correctness
  - Base module features
  - Performance
- Designed for use by GAMS staff and our solver providers
- Allows any user to reproduce our tests
- To retrieve models from testlib: > `testlib ...`.

3.60.1.3 GAMS IDE

- The tabs showing the file names in the editor are maintainded in sorted order using the full path name.
- Support for regular expressions in search, search and replace and search in files
- When searching text using Ctrl-F, the word under the cursor will be used as the text to search for.
- Added a viewer for reference files. A reference file is created using the rf option in the gams call. The IDE assumes that the file extension for the reference file is '.ref'
- An option was added to emulate a subset of the Epsilon keyboard mapping.

3.60.1.4 GDX Utilities

- GDXMerge added; a utility to combine multiple gdx files into a single gdx file.
- GDX2Veda added; a utility to export GAMS data into the [VErsatile Data Analyst (VEDA)] (http://www.kanors-emr.org/software).

3.60.1.5 GAMS Branch-and-Cut-and-Heuristic (BCH) facility

- The GAMS Branch-and-Cut-and-Heuristic (BCH) facility allows GAMS users to interact with a running MIP/MINLP solver by supplying specialized GAMS programs to generate *cutting planes* and good *integer feasible solutions*. This allows GAMS users to apply complex solution stragies without having to have intimate knowledge about the inner workings of a specific MIP system. BCH strategies can now be implemented rapidly and reliably within a matter of days rather than weeks. Details and examples can be found here.

3.60.1.6 Quadratically Constrained Program Types QCP, MIQCP and RMIQCP

- New model types for quadratically constrained problems are:
  - QCP: Quadratically Constrained Programs
  - MIQCP: Mixed Integer Quadratically Constrained Programs
  - RMIQCP: Relaxed Mixed Integer Quadratically Constrained Programs
- These model types are like NLP, MINLP and RMINLP with nonlinearities restricted to be quadratic forms. QCP problems can be solved with existing nonlinear solvers and large-scale LP solvers that offer quadratic extensions. For examples, look at the models qalan, qcp1, qdemo7, and qsambal in the model library.

3.60.1.7 Solver Link Options SOLVELINK

- This new option allows you to control the way solver's or subsystems are invoked. This may be helpful when solving a large number of models that are relatively small compared to the size of the overall database.
- The new options values are:
  - 0: Make an automatic save/restart for each solve (default - old behavior)
  - 1: Calls the subsystem using the shell method
  - 2: Call the subsystem using the spawn method
- This option can be specified as a gams parameter (> `gams ... solvelink=n`), as part of an option statement (`option solvelink=n;`) or specified as a model attribute (`mymodel.solvelink=n;`) Some demo limits removed
- The demo version size restriction on the number of symbols and number of unique set elements has been removed. Previously, the demo/student versions were limited to 2000 symbols.

3.60.1.8 Save Point Options SAVEPOINT

- This new option directs GAMS to write solutions to GDX files for later use in the same or other programs. This may be helpful in cases when we want to provide good starting points or process large number of solutions at a later stage. The new option values are:
  - 0: Do not write point files (default)
  - 1: Write the solution the GDX file <`workdir`><`modelname`>`_p.gdx`
  - 2: Write the solution the GDX file <`workdir`><`modelname`>`_p`<`solvenumber`>`.gdx`
- This option can be specified as a gams parameter (> `gams ... savepoint=n`), as part of an option statement (`option savepoint=n;`) or specified as a model attribute (`mymodel.savepoint=n;`). The `execute_loadpoint` allows you to merge solution points into any GAMS database.

3.60.1.9 Relaxed MCP and MPEC syntax

- A variable can now appear in more than one `equ.var` pair in the model list, however, in the final instance of the model, all mappings have to be unique.

3.60.1.10 Enhancements to EXECUTE_LOAD and EXECUTE_UNLOAD

- execute_load
  - `gamspar = gdxvar.xx`: `xx` can be `l m` ...
  - `gamsvar.xx = gdxvar.xx`
- execute_unload:
  - no arguments means dump all
  - allow multiple occurrence of the same symbol `x.l x.up x a a`
  - `x.l=a`: `a` is the complete `x` variable

3.60.1.11 New EXECUTE_LOADPOINT

- This new command `execute_loadpoint` is similar to `EXECUTE_LOAD`, however, the new values are merged with the old values. If no arguments besides the name of the GDX file are given, all variables and equations that match variables and equations of the calling GAMS programs will be merged with the GDX level and marginal values. Bounds, scales and priorities will remain unchanged. This can be very useful in connection with the `SAVEPOINT` facility.

3.60.1.12 Nested GAMS parameter PF=xxx

- The command line include parameter can now handle nested includes

3.60.1.13 New Functions

A number of new functions have been added.

- `TimeStart`: Accumulated restart time
- `TimeComp`: Compilation time in seconds
- `TimeExec`: Execution time in seconds
- `TimeClose`: accumulated save time
- `Frac`: frac(x) is the fractional part of x
- `ErrorLevel`: Returns code of the most recently called external program
- `HeapSize`: Current Heap size in Mbytes
- `Fact`: Factorial
- `UniformInt`: Uniform integer random number between `UniformInt(low,high)`
- `PI`: The famous constant 3.14....
- `NcpF`: `NcpF(x,y,c) = sqrt(sqr(x) + sqr(y) + 2*c) - x - y` Fisher
- `NcpCM`: `NcpCM(x,y,c) = x - c*ln(1+exp((x-y)/c)` Chen-Mangasarian
- `Entropy`: `entropy(x) = -x*log(x)`
- `Sigmoid`: `sigmoid(x) = 1/(1+exp(-x))`
- `Log2`: logarithm base 2
- `IfThen`: `ifthen(condition,true expression,false expression)`
- `Edist`: `Edist(x,y,..) = sqrt(sqr(x)+sqr(y)+..)`
- `CEntropy`: `centropy(x,c) = x*ln(x/c)` Cross Entropy

3.60.1.14 Set Table

- Allows to enter set data in tabular form similar to parameter data. The data entries can be numbers and special values including yes/no. To allow data tables to be interpreted as Parameters or sets, the special values of NO and YES will be interpreted as 0 and 1 respectively.

3.60.1.15 Some new minor and exotic features

- Dynamic reinterpretation of real powers. When using automatic translation it may be helpful to treat a real power as an integer power if the exponent is constant and within `e-12` of an integer value. This can be enabled by setting Option `sys10=1;` and disabled by setting option `sys10=0;` (default).
- File Name Casing. File names as passed to the operating system maintain the original casing. This can be changed with the GAMS parameter `FILECASE=n`, where
  - `n=0` means to retain original casing (default)
  - `n=1` uppercase the filename, and
  - `n=2` will lowercase the filename.
- `$CLEARERROR` clear all compilation errors
- `$TERMINATE` terminates compilation and skips executions
- `$SETNAMES filename filepath filename fileextension` separates the string filename into its three components and stores then as GAMS environment variables. For example, `$setname d:\gams\xxx.txt fp fn fe` creates/updates three string variables `fp`, `fn`, `fe` which will combine into the original filename string by `%fp%%fn%%fe%`.
- `$SETCOMPS s1.s2.s3.. first second third` separates the string `s1.s2...` into its components and stores them in the environment variables `first`, `second`,..
- `$IF WARNINGS` command
- `$REMARK` comment with string substitution
- `$ON/OFFUNDF` allows the use of the special value `UNDF` in data statements and expression.
- `System.xxx` and `%system.xxx%` have been enhanced and unified. The complete list is:

```
DATE, TIME, TITLE, PAGE, SFILE, RTIME, RDATE, RFILE, IFILE, OFILE
,VERSION PLATFORM, LP, NLP, ...all model types, LICE1, LICE2, GSTRING,
SSTRING, PFILE ILINE, OPAGE, VERID, ELAPSED, MEMORY, TSTART, TCLOSE,
TCOMP, TEXEC, INCPARENT INCPARENTL, INCNAME, INCLINE, LINE, LISTLINE,
FILESYS, PRLINE, PRPAGE, FE, FN FP, REDIRLOG, ERRORLEVEL
```

  The `system.xxx` form is used in `PUT` statements, the `%system.xxx%` form is used for string substitutions during compilation.
- Fast Projections and Aggregations The GAMS executions engine performs poorly on certain large data structures. This can be overcome with telling GAMS to 'pipeline' certain operations via an `OPTION` statement.

  ```
  Option left < right, left <= right,..;
  ```

  Where left and right are GAMS identifiers with conforming domain declarations. The dimensionality of the left symbol has to be equal or less than the dimensionality of the right side. If the left dimensionality is less than the right one, the operation performed is an aggregation or projection depending on the data type of the left side. In all cases, indices are permuted according to the domain definitions. If a symbol has identical domain definitions they are permuted right to left (<) or left to right (<=). For example, assume we have

  ```
  Set i, fromto(i,i), tofrom(i,i);
  alias(i,ii);
  parameter in(i), out(i);

  option tofrom < fromto, in < fromto, out <= fromto;
  ```

is equivalent to

```
tofrom(i,ii) = fromto(ii,i);
in(i) = sum(fromto(ii,i),1);
out(i) = sum(fromto(i,ii), 1);
```

3.60.2 Solvers

3.60.2.1 BARON 6.0

- Improved memory management
- Default bounds are new +/- infinity
- CONOPT run not required to get marginals
- Several algorithmic improvements

3.60.2.2 CPLEX 9.0

- GAMS/Cplex is now based on Cplex 9.0. In addition to performance improvements and enhancements of existing features, support is provided for the new GAMS model types QCP and MIQCP. Because of some compatibility issues, we still ship CPLEX 8.1 for Digital Unix. Details can be found here.

3.60.2.3 Large scale QCP Solvers

- MOSEK handles QCP and MIQCP models
- CPLEX handles QCP and MIQCP models.
- XPRESS handles QCP with quadratic term in the objective only.
- SBB handles MIQCP models.
- PATHNLP takes advantage of quadratic forms for QCP models.

3.60.2.4 NLP solvers capable of solving QCPs

- All NLP solvers (CONOPT, MINOS, SNOPT, ...) handle QCP and RMIQCP models.

3.60.2.5 AMPL and Lingo Wrappers are now part of the GAMS distribution

3.60.2.6 MOSEK 3.0

3.60.2.7 MPSGE

- Improved reporting of formulation errors
- There are new rules for choosing proper labels and identifiers in MPSGE models. The MPSGE engine was initially designed around the old 10-character limit for labels and identifiers imposed by GAMS/Base. When GAMS/Base relaxed these limits to 31 characters, a flag was added to check for compatibility with the old rules to avoid breaking MPSGE. We've now removed the check in GAMS/Base - the MPSGE engine makes its own check now. This means we can use long labels and identifiers under the new rules.
- The testlib models `mpsge01` $*$ `mpsge10` document and test the rules for identifiers in MPSGE models. E.g. do

```
testlib mpsge01
gams mpsge01
```

3.61 21.2 Maintenance release (September 03, 2003)

3.62 21.1 Maintenance release (June 02, 2003)

3.63 21.0 Major release (May 15, 2003)

3.63.1 GAMS System

3.63.1.1 GAMS Model Library

- We added quite a few global optimization models to the GAMS model library that demonstrate the power of the new suite of Global Optimizers available in GAMS: BARON, LGO, and OQNLP. In addition, we added a model that calculates the US holiday schedule (HOLIDAY) and a model for the Five Leaper Tour problem (FIVELEAP). The model NASH gives an example of a Mathematical Program with Equilibrium Constraints (MPEC) which can be solved by the beta solver NLPEC.

3.63.1.2 IDE

- GDX data viewer: Symbols can be sorted by various attributes, and the display uses two grids to display the data. The top grid shows the plane index, the bottom grid the current data. The indices can be arranged using drag and drop. The arrangement is preserved in the project file. The data viewer can write a single symbol or all symbols to an html file.
- Removed Setup button from the print dialog screen. Printer setup is now available from the window that follows this dialog. This allows more printer options to be selected, such as duplex printing.
- Added option to update the GAMS system using a ZIP file in the GAMS system directory. (File | Options | Execute | Update)
- Added entries on the help menu for easy access to some documentation files.
- Revised the on-line help and added a number of screen shots.
- Added `file:///` as a recognized hyperlink.

3.63.1.3 GDX Utilities

- New documentation includes GDX facilities in GAMS.
- GDXXRW (Excel interface)
  - Added 'usage' output
  - Removed SSET as a type, and added Values option for a set.
- GDXDIFF
  - Added option to specify a single field for comparison (for variables and equations)
  - Added option to ignore comparison of associated text
  - Added RelEps option for relative comparisons
- GDXDUMP
  - Added UelTable option

3.63.2 Solvers

3.63.2.1 New Global Solvers

- NLP and MINLP problems frequently have multiple local optima. The three new solvers, BARON, LGO and OQNLP aim at finding the best possible local optima, the global optimum solution. The solvers differ in the methods they use, in whether they find globally optimal solution with proven optimality, in the size of models they can handle, and in the functional forms they accept.

3.63.2.2 BARON

- BARON is from The Optimization Firm, LLC and the University of Illinois at Urbana-Champaign. The Branch And Reduce Optimization Navigator derives its name from its combining interval analysis and duality in its reduce arsenal with enhanced branch and bound concepts as it winds its way through the hills and valleys of complex optimization problems in search of global solutions.

3.63.2.3 LGO

- LGO is from Pinter Consulting Services, Inc. This algorithm combines rigorous statistical methods with traditional mathematical programming methods to find solutions within well defined bounds. LGO stand for Lipschitz Global Optimization.

3.63.2.4 OQNLP

- OQNLP from Optimal Methods, Inc and OptTek Systems, Inc. is a solver for global optimization of non-linear problems with discrete and continuous variables. This multi start solver combines robust nonlinear optimization technologies with state-of-the-art meta heuristics like scatter search. In this official release of OQNLP, nonlinear subproblems with different starting points can be solved by any GAMS NLP solver as well as the build-in LSGRG solver.

3.63.2.5 CONOPT

- The CONOPT family of NLP solvers now have three members, CONOPT1, CONOPT2, and CONOPT3. The alias CONOPT which used to point to CONOPT2 will now call CONOPT3. In case you need to run an older version of CONOPT you must specify CONOPT1 or CONOPT2. CONOPT3 is an enhancement of CONOPT2. An additional solver component based on sequential quadratic programming (SQP) principles has been added and it uses the new 2nd order facilities in GAMS. In addition, an improved automatic scaling routines is now used as default.
- CONOPT3 is a true multi-method solver that combines its many solver components (SQP, SLP or sequential linear programming, Quasi-Newton, and Steepest Descend) with dynamic selection of the best component based on performance statistics.
- CONOPT2 is, apart from minor bug fixes, identical to CONOPT2 from the previous release. CONOPT1 has not been changed and it will not be updated any more. We encourage all CONOPT1 user to move on to CONOPT3 or CONOPT2.

3.63.2.6 CPLEX 8.0

- Find detailed release notes here

3.63.2.7 MINOS

- The new GAMS/MINOS, which is based on MINOS 5.51 uses a new LU factorization with Threshold Rook Pivoting. It also has better memory estimation and improved reporting of infeasibilities.

3.63.2.8 MOSEK

- MOSEK from MOSEK ApS, Copenhagen Denmark is a large scale system for solving linear, mixed-integer linear, and convex nonlinear mathematical optimization problems. MOSEK is particularly well suited for solving sparse large-scale problems using an extremely efficient interior point algorithm. This official release of MOSEK using library version 2.5 also solves second order conic programs.

3.63.2.9 OSL

- The OSL optimizer family is now represented by its newest member, OSL3. You can still run OSL2 and OSL1 (where available) by specifying them explicitly, but the default OSL is OSL3.
- Improvements to OSL3 over past OSL versions include:
  - Better performance for the simplex, barrier, and MIP codes
  - Better memory management
  - More robustness, especially in the barrier code

3.63.2.10 PATH

- The PATH presolve has been extended and strengthened.

3.63.2.11 SNOPT

- The new GAMS/SNOPT, which is based on SNOPT 6.2-1 uses a new LU factorization with Threshold Rook Pivoting. Memory estimation, the reporting of infeasibilities, and the handling of LP's has been improved.

3.63.2.12 XPRESS

- GAMS/XPRESS now links to XPRESS 2003, the latest optimizer from DASH. Optimizer improvements include:
  - Completely redesigned and rewritten branch and bound framework
  - New code for the Newton Barrier optimizer
  - Enhanced branching technology, cut strategies and heuristics
  - Better memory management

3.63.3 Documentation

3.63.3.1 McCarl's GAMS User Guide

- The new McCarl's GAMS User Guide will be installed automatically with CD installation. If you download the GAMS system from our web site, you have to install it separately, see *[not available anymore]*.

3.63.3.2 Solver Manuals

- There is a revised version for most of the solver manuals, all of them are in a uniform format. A composite manual, The Solver Manuals, is also available for printing.

3.64 20.7 Maintenance release (June 14, 2002)

3.65 20.6 Major release (May 25, 2002)

3.65.1 GAMS System

3.65.1.1 MODLIB

- We have two new models (clearlak and srkandw) that demonstrate the use of scenred, an interesting MIP model for scheduling TV commercials, and an example that shows the use of some GDX utilities.

3.65.1.2 MPS2GMS

- This utility converts MPS files into a GAMS program making use of the GAMS GDX facility. This replaces the contributed utility mps2gams.

3.65.1.3 POSIX UTILITIES

- Starting with this distribution the GAMS system for Windows includes a collection of Posix utilities which are usually available for the different Unix systems and therefore help to write platform independent scripts. More information here.

3.65.1.4 SCENRED

- ScenRed (a new addition to the GAMS system) allows GAMS users easy access to the scenario reduction algorithms found in [ScenRed] (http://www.mathematik.hu-berlin.de/%7Eromisch/projects/GAMS/scenre
Given the event tree for a stochastic program, ScenRed determines a subset of scenarios and the optimal redistribution of probabilities for the preserved scenarios. This is useful when the stochastic program that results from using the original (complete) event tree is too large to solve. Making use of the new execution-time GAMS data interface (execute_load/execute_unload), GAMS/ScenRed takes the original tree from the modeler, along with parameters controlling the reduction, and returns a reduced tree for use in subsequent solves or data manipulation.

3.65.2 Solvers

3.65.2.1 SBB

- There are two new options (dfsstay and acceptnonopt) in SBB that can help to find good solutions more quickly as well as handle almost optimal subproblems which are ignored by default.

3.65.2.2 XA

- The GAMS/XA link has been modified to work with the new generation (Version 13) of XA libraries. All XA-supported architectures are now using this new version, and so will include a Newton-barrier capability.

3.65.3 Beta Solvers

3.65.3.1 BARON

- Complete primal and dual solutions values are now reported by using CONOPT as a post processor. The preprocessing has been enhanced to allow free (=N=) equations.

3.65.3.2 MOSEK

- MOSEK from MOSEK ApS is a large scale system for solving problems of the following classes:
  - Linear optimization
  - Mixed integer linear optimization
  - Convex quadratic optimization
  - Conic quadratic optimization
- The released version of MOSEK will also solve
  - Quadratically constrained convex optimization
  - Convex optimization
- More information about MOSEK can be found at [www.mosek.com] (http://www.mosek.com).

3.65.3.3 OQNLP

- OQNLP from Optimal Methods, Inc is a solver for global optimization of smooth constrained problems with either all continuous variables or a mixture of discrete and continuous variables. This multi start method combines mathematical programming approaches with meta heuristics like tabu search.

3.65.3.4 NLPEC

- NLPEC is a solver for MPEC models that works by reformulating the MPEC model as an NLP, solving the NLP using one of the GAMS NLP solvers, and then extracting the MPEC solution from the NLP solution. All of this happens automatically, although it is possible to access the intermediate NLP model. Like the CONVERT solver, the reformulated models NLPEC produces are in scalar form. Many different reformulations (currently around 20) are supported by the NLPEC solver. MPEC models are notorious for their difficultly, but the combination of different reformulations and NLP solvers give users a good chance to solve them.

3.65.3.5 PATHNLP

- PATHNLP is now set up to allow the PATHLIB presolver to perform additional model reductions.
- PATHNLP now uses the same libraries as PATH. Previous versions used an experimental version of PATHLIB (ver 5.X). Since the supported PATHLIB is now 4.6, it looks like PATHNLP now uses an older PATHLIB; this is not the case.

3.66 20.5 Maintenance release (January 28, 2002)

3.67 20.4 Maintenance release (January 21, 2002)

3.68 20.3 Major release (December 24, 2001)

3.68.1 GAMS System

3.68.1.1 GAMS

- Restrictions of number of symbols, set elements and execution code size have been completely removed.
- Performance enhancements for very large models with millions of rows and columns.
- Minor language enhancements: $on/offecho copies a block of lines to a file. The text is subject to string macro processing as well as inline and eol comment removal. For example:

```
$onecho > myfile
...
$offecho
```

- To append to a file use ">>".

3.68.1.2 IDE

- Contributed solver installation files (e.g. QPWRAP, DEA, ... see [User Contributed Software and Tools] (https://forum.gamsworld.org/viewforum.php?f=16&sid=60f52f2a7983d94c0202a0834f780778)) can now be installed using the IDE. Place the installation file (file extension ∗.zip or ∗.pck) in the GAMS system directory and click on the "Update" button in File->Options->Execute

3.68.2 Solvers

3.68.2.1 Uniform Solver Naming Conventions

- Since solvers are distributed with past, production, and beta versions (e.g. OSL, CONOPT, ...) a consistent naming convention has been adoped.
- The following simple rules apply:
  1. The base name of the solver does not change (e.g. CPLEX, OSL, CONOPT, ...) and refers to the current production version.
  2. Past and beta versions have names derived from the base name. For example, in this release we distribute CONOPT1 (past), CONOPT2 (production), and CONOPT3 (future). The base name CONOPT refers to CONOPT2.
- If you select the base name of a solver from now on you will get the most recent production version of the solver. For the following solvers the naming convention has changed:

New Name (20.3-)	Old Name (-20.2)
CONOPT or CONOPT2	CONOPT2
CONOPT3	CONOPT3
CONOPT1	CONOPT
OSL or OSL2	OSL2
OSL3	OSL3
OSL1	OSL
MINOS or MINOS55	MINOS
MINOS5	MINOS5
PATH or PATHC	PATHC
PATHOLD	PATH
MILES or MILESE	MILESE
MILESOLD	MILES

3.68.2.2 CONOPT

- The future version CONOPT3 is also available on Linux.

3.68.2.3 CONVERT

- For BARON and AMPL output GAMS/CONVERT eleminates the objective variable and uses a "true" objective function.

3.68.2.4 SBB

- NLP solvers sometimes have difficulties solving particular nodes and using up all the resources in this node. SBB provides options (see *subres*, *subiter*) to overcome these instances, but options have to be set in advance. Now, SBB keeps track of how much time is spend in the nodes, builds an average over time, and automatically controls the time spend in each node. The option *avgresmult* allows the user to customize this new feature.

3.68.2.5 XPRESS

The XPRESS licensing for Windows, Linux, and Solaris is now much simplified; there is no XPRESS-specific licensing procedure for the user to follow. The GAMS/XPRESS solver takes care of licensing the XPRESS DLL, and does this automatically. Of course, as with other solvers, the GAMS/XPRESS must be licensed with your GAMS system.

3.69 20.2 Maintenance release (November 22, 2001)

3.69.1 Solvers

3.69.1.1 CPLEX 7.5

- Find detailed release notes here

3.69.1.2 XPRESS 13.01

- The GAMS/XPRESS link has been updated to use the new 13.01 solver libraries from XPRESS.

3.70 20.1 Major release (October 31, 2001)

3.70.1 GAMS System

3.70.1.1 GAMS

- A new licensed Privacy and Security feature which allows to purge, hide or protect model data and model components from unauthorized use. Models and data that are saved with the privacy feature enabled are locked to a specific license and protected. For more details consult the corresponding appendix of the GAMS Users Guide.
- A new GAMS parameter EXECMODE can be used to reconfigure certain features when operating in an application service environment. $call/execute, $echo/put and $includes can be disabled or restricted to operate only on certain sub trees of the file system.
- Model options can be initialized from the command line or the GAMS parameter window in the IDE (bratio, iterlim, domlim, reslim, optcr, optca, sysout, solprint, optfile and all solvers for model types). For example:

  ```
  gams myfile LP=BDMLP RESLIM=10
  ```

  will override the default solver to be BDMLP and force an interrupted after 10 seconds.
- Other minor language enhancements are:
  - Factorial function FACT
  - Implication operator (-> or `imp`)
  - Equivalence operator (<=> or `eqv`)
  - $ECHON and echo without EOL
  - $IFi which makes case insensitive string comparisons
- A number of restrictions have been relaxed:
  - NLP model can have up to 8 million constants.
  - A GAMS program can have up to 1 million GAMS symbols or unique elements.
  - The internal code space has been increased to 32 million instructions.
  - The model size is limited by hardware addressing space only.
- GAMS environment variables can now be initialized or reset from the command line or parameter window. The prefix `--` signals an environment variable. For example:

  ```
  gams myfile --mypath="d:a b"
  ```

  is equivalent to inserting `$set mypath d:a b` in the GAMS input file. The GAMS parameter SETTYPE (ST) values of 0 (default), 1 and 2 change the implied $set command to $set/setlocal/setglobal respectively. The evaluation of nonlinear expressions inside solvers of the form <`expression`>**<`integer constant=""`> are now interpreted more restrictive for the values of -1,0,1 and 2. The solver will issue domain violations when the value of an expression becomes negative. For example, x**0 will now require that x is always greater or equal to zero. The function power(expression,integer constant) should be used if this is not intended.
- Derivatives for 0**0 are not defined and may cause problems during the solution process. Warnings are issued when this case arises.

IDE

- Alternate license dialog was disabled
- Selecting 'GAMS model library' will no longer add this entry to the recently used list of libraries.
- Find/Replace dialogs no longer default to 'Selected Text' when text was selected in the editor.

3.70.1.2 GAMS Model Library

- The references for existing models have been updated which allows an indexing by author.
- We added a couple of MINLP models:
  - CSCHED Cyclic Scheduling of Continuous Parallel Units
  - GASNET Optimal Design of a Gas Transmission Network
  - WINDFAC Winding Factor of Electrical Machines
  - STOCKCC Minimizing Total Average Cycle Stock
  - NSHARPX Synthesis of General Distillation Sequences
  - MINLPHIX Heat Integrated Distillation Sequences
- The new model CESAM illustrates a cross entropy technique for estimating the cells of a consistent Social Accounting Matrix (SAM) assuming that the initial data are inconsistent and measured with error.
- The COPS models have been added:
  - POLYGON Largest small polygon COPS 2.0 #1
  - ELEC Distribution of electrons on a sphere COPS 2.0 #2
  - CHAIN Hanging Chain COPS 2.0 #3
  - CAMSHAPE Shape optimization of a cam COPS 2.0 #4
  - PINENE Isometrization of alpha-pinene COPS 2.0 #5
  - POPDYNM Marine population dynamics COPS 2.0 #6
  - FLOWCHAN Flow in a channel COPS 2.0 #7. Uses new function fact
  - ROBOT Robot arm COPS 2.0 #8
  - LNTS Particle steering COPS 2.0 #9
  - ROCKET Goddard rocket COPS 2.0 #10
  - GLIDER Hang glider COPS 2.0 #11
  - GASOIL Catalytic cracking of gas oil COPS 2.0 #12
  - METHANOL Methanol to hydrocarbons COPS 2.0 #13
  - CATMIX Catalyst Mixing COPS 2.0 #14
  - TORSION Elastic-plastic torsion COPS 2.0 #15
  - JBEARING Journal bearing COPS 2.0 #16
  - MINSURF Minimal surface with obstacle COPS 2.0 #17

3.70.2 Solvers

3.70.2.1 CONVERT

- This solver converts a GAMS model instance into a format used by other modeling and solutions systems. The transformed models have only scalar equations and variables with all original naming removed. This 'scalar' format hides the original model and data development and allows one to exchange confidential models for solver tuning and research. Currently, CONVERT can translate GAMS models into the following formats:
  - AMPL
  - BARON
  - CplexLP
  - CplexMPS
  - GAMS
  - LGO
  - LINGO
  - MINOPT
- This solver is also available as an e-mail based service from the GAMS World.

3.70.2.2 CPLEX 7.1

- Find detailed release notes here

3.70.2.3 MPSGE/MCP Solver

- There has been a reorganization of the MCP/MPSGE solvers. Central to this is the introduction of a new core library for MPSGE models. There are two MCP solvers compatible with this new scheme: PATHC and MILESE. Both of these solvers implement the same algorithms as their predecessors PATH and MILES: only their internal interface is changed. The old solvers, using the old interface, are still available for the time being. However, the new solvers offer certain advantages; for example, PATHC contains optional preprocessing of the MCP model that was not possible because of the structure of the old interface. The new interface also relies on the GAMS Base module to perform many more consistency checks on the MCP formulation and gives more immediate and extensive diagnostic information when modeling errors occur. Finally, the new interface will allow one to "plug in" an alternate version of the MPSGE core library to experiment with different MPSGE functional forms.
- It's important to note that backward compatibility is maintained, and that all four solvers (PATH, PATHC, MILES ,and MILESE) will accept the same GAMS files as input. The interface changes referred to above are all internal in nature.

3.70.2.4 OSL3

- The GAMS/OSL link has been updated to use the new OSL3 libraries from IBM.

3.70.2.5 PATHNLP

- PATHNLP is the latest NLP solver at GAMS. Essentially, it implements an SQP algorithm by automatically reformulating an NLP problem as a complementarity problem and solving this using the proven, reliable PATH solver. This allows second order information to be used in the solution of the model, which often results in greater solution efficiency. In addition, the marginal values are sometimes more exact than those provided by first-order methods. The advantages of this approach are most apparent on very large, sparse models with many nonlinear variables and degrees of freedom. In these cases, the superbasics limit of other NLP codes limits their effectiveness but is no barrier when using PATHNLP. The new PATHNLP solver allows the solution of certain previously unsolvable models (e.g. maximum entropy models).

3.70.2.6 SBB

- SBB derives an implicit absolute termination tolerance if the model has a discrete objective row. This may speed up the overall time if the user has tight termination tolerances (optca, optcr).
- SBB passes indicies of rows with domain violations back to the LST file. All domain violation from the root node as well as all sub nodes are reported and the user might take advantage of this information to overcome these violations.

3.70.2.7 XPRESS

- The GAMS/XPRESS link has been updated to use the new 12.50 solver libraries from XPRESS. Benefits include an improved pricing algorithm in dual simplex, improved sparse/dense data handling in all simplex codes, and a general 30% reduction in simplex solve time. The barrier solver incorporates row-wise Cholesky factorization, which gives improved performance on many problems, and has more efficient memory usage. The presolve has more efficient memory management, and the eliminator phase has been improved. The branch and bound algorithm has improved cut selection and faster clique cut generation.
- Improvements to the link itself allow for better detection and reporting of unbounded or infeasible models.

3.70.3 Solvers in Beta Version

3.70.3.1 BARON

- BARON is a computational system for solving non convex optimization problems to global optimality. Purely continuous, purely integer, and mixed-integer nonlinear problems can be solved with the software. The Branch And Reduce Optimization Navigator derives its name from its combining interval analysis and duality in its reduce arsenal with enhanced branch and bound concepts as it winds its way through the hills and valleys of complex optimization problems in search of global solutions.
- The demo size for this solver is reduced to 10 variables and 10 equations. The beta version of BARON is available on Windows, Linux and AIX.

3.70.3.2 CONOPT3

A new version of CONOPT is available for beta testing, find release notes here. It is available on Windows only.

Chapter 4

User's Guide

This documentation guides GAMS users through several topics in the GAMS system.

4.1 Installation and Licensing

This part leads step by step through the installation process on three main platforms and describes the GAMS licensing system:

- Supported Platforms
- Installation Notes for macOS
- Installation Notes for Unix
- Installation Notes for Windows
- Standard Locations
- Licensing

4.2 Tutorials and Examples

This part describes step by step through several selected tutorials and a small number of examples.

- A GAMS Tutorial by Richard E. Rosenthal
- Quick Start Tutorial
- Good Coding Practices
- Fixing Compilation Errors
- Finding and Fixing Execution Errors and Performance Problems
- Comparative Analyses with GAMS
- Good NLP Formulations
- Data Exchange with Other Applications

  - Data Exchange with Text Files
  - Data Exchange with Microsoft Excel
  - Data Exchange with Databases
- Executing GAMS from other Environments
- Using GAMS Studio

There are also tutorials and examples of the Application Programming Interfaces

- .NET Tutorial and `Examples`
- C++ Tutorial and `Examples`
- Java Tutorial and `Examples`
- Python Tutorial and `Examples`

4.3 GAMS Language and Environment

This part introduces the components of the GAMS language in an ordered way, interspersed with detailed examples that are often drawn from the model library.

- Introduction - an introduction to the GAMS User's Guide.
- GAMS Programs - The structure of the GAMS language and its components
- Set Definition - The declaration and initialization of sets, subsets, and domain checking.
- Dynamic Sets - The membership assignment, the usage of dollar controls, and set operations.
- Sets as Sequences: Ordered Sets - Special features used to deal with a set as if it were a sequence.
- Data Manipulations with Parameters - The declaration and assignment of GAMS parameters.
- Data Entry: Parameters, Scalars and Tables - Three basic forms of GAMS data types : Parameters, Scalars and Tables.
- Variables - The declaration and attributes of GAMS variables.
- Equations - The definition and declaration of GAMS equations.
- Model and Solve Statements Model - The specificiation of a GAMS model and how to solve it.
- Conditional Expressions, Assignments and Equations - The conditional assignments, expressions and equations in GAMS.
- The Display Statement - The syntax, control, and label order in display.
- Programming Flow Control Features - The GAMS programing flow control features : loop, if-elseif, while, and for statements.
- The Option Statement - The list and detailed description of options.
- System Attributes - The system attributes
- The Grid and Multi-Threading Solve Facility - The basic concepts and Grid Features.
- Special Features for Mathematical Programs - Special features in GAMS that do not translate across solvers, or are specific to certain model types.

The following discusses the execution of GAMS, the use of special features, and other miscellaneous topics.

- GAMS Output - The control of GAMS compilation output, execution output, output produced by a solve statement, and error reporting.
- GAMS Log
- The GAMS Call and Command Line Parameters - The list and detailed description of GAMS command line parameters.
- Dollar Control Options - The list and detailed description of dollar control options.
- The Put Writing Facility - The put writing facility of the GAMS language.
- Solver Usage - Controlling solvers.
- The Save and Restart Feature - The GAMS save and restart feature and the work file.
- Embedded Code Facility - The Embedded Code Facility (e.g. how to embed Python code into GAMS).
- GAMS Connect - The GAMS Connect Framework.
- Extrinsic Functions - The extrinsic function library and comparison with external equations.
- External Equations - A facility for connecting code written in different programming languages to equations and variables in a GAMS model.
- GAMS Return Codes - The structure of error codes, the return codes of the GAMS compiler and execution system, and the driver return codes.
- GAMS Data eXchange (GDX) - GAMS Data eXchange (GDX) facilities and utilities for Binary Data Exchange.
- Extended Mathematical Programming (EMP) - Extended Mathematical Programming (EMP).
- Accessing Model Libraries - Introduction of GAMS Model Library.
- Mathematical Programming System for General Equilibrium analysis (MPSGE) - A mathematical programming system for general equilibrium analysis which operates as a subsystem within GAMS.
  - Introduction to MPSGE
  - MPSGE Models in GAMS
  - Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE
  - Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae
  - `A Library of Small Examples for Self-Study`
  - `Linking Implan Social Accounts to MPSGE`
  - The MPSGE guide is also available as `PDF`

4.4 Glossary

An alphabetical list of GAMS terms is available in the Glossary.

4.5 Supported Platforms

GAMS supports the following platforms:

Platform	Description
x86 64 bit Linux	AMD- or Intel-based 64-bit (x86_64) Linux systems with glibc 2.17 or higher.
arm 64 bit macOS	Apple Silicon (arm64) Macintosh system with macOS 13 (Ventura) or macOS 14 (Sonoma).
x86 64 bit macOS	Intel-based 64-bit (x86_64) Macintosh system with macOS 12 (Monterey), macOS 13 (Ventura), or macOS 14 (Sonoma).
x86 64 bit MS Windows	Windows 10 or newer on AMD- or Intel-based (x86_64) architectures.

Note

Compared to most of the GAMS system, GAMS Studio has some additional system requirements which are detailed here.

While the GAMS execution system itself is available on all supported platforms, for certain solvers, tools, and APIs, different availabilities can apply. For details, see

- Supported Platforms for Solvers
- Supported Platforms for Tools
- Supported Platforms for High-level APIs
- Supported Platforms for Low-level APIs

4.6 Installation Notes for macOS

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure. Furthermore, we recommend to use the PKG installer for macOS because it includes the GAMS Studio and it integrates GAMS into macOS, e.g. it is possible to open the GAMS Studio via the Launchpad.

Two installation procedures are available for GAMS on macOS:

- Installation using the PKG installer (GAMS46.4.1.pkg)
- Installation using the self-extracting archive (osx_x64_64_sfx.exe or osx_arm64_sfx.exe)

4.6.1 Installation using the PKG installer (GAMS46.4.1.pkg)

1. Obtain the GAMS PKG file for your CPU architecture (`x86_64` or `arm64`), which is available from `http://www.gams.com/download`.
2. Double click the package and follow the instructions. By clicking on `Customise` in the tab `Installation Type` you can decide to add GAMS to the PATH as well as to reject the installation of GAMS Studio. Note that adding GAMS to the PATH is done by modifying your shell profile file (`~/.zprofile`, `~/.bash_profile` etc.). A backup of your old profile is saved (`~/.zprofile.gamsbackup`, `~/.bash_profile.gamsbackup` etc.).
3. In order to test the GAMS installation with `GAMS Studio` open the `Launchpad` and click the `GAMS Studio 46` icon to open the application. Alternatively, go to Applications and open the `GAMS Studio 46` application.

4. Install your license via the corresponding GAMS Studio dialog. The license file is nowadays sent via email, with instructions. You can also request a demo license from http://www.gams.com/download.

   Optionally, you can create the license file 'gamslice.txt' in a directory that GAMS searches to find a license. GAMS searches a couple of system wide and user specific standard locations for a license file.

5. Open the `Model Library Explorer` and open the `TRNSPORT` model (sequence number 001). Run the model and check the contents of the process log, which should be similar to the Terminal output listed in the next bullet point.

6. In order to test the GAMS installation without using `GAMS Studio` open a Terminal window. Execute the following commands to see if everything works as expected:

```
gamslib trnsport
gams trnsport
```

   The output should be similar to this:

```
--- Job trnsport Start 06/26/14 11:24:56 24.3.1 r46409 DEX-DEG Mac x86_64/Darwin
GAMS 24.3.1   Copyright (C) 1987-2014 GAMS Development. All rights reserved
Licensee: ...
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.024
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
---   6 rows  7 columns  19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.114

IBM ILOG CPLEX   24.3.1 ... DEG Mac x86_64/Darwin
Cplex 12.6.0.0

Reading data...
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.02 sec. (0.00 ticks)

Iteration      Dual Objective            In Variable           Out Variable
     1             73.125000    x(seattle.new-york) demand(new-york) slack
     2            119.025000     x(seattle.chicago)  demand(chicago) slack
     3            153.675000    x(san-diego.topeka)   demand(topeka) slack
     4            153.675000  x(san-diego.new-york)  supply(seattle) slack
LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective :         153.675000

--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 06/26/14 11:24:57 elapsed 0:00:00.487
```

4.6.2 Uninstall PKG installation

To uninstall a GAMS installation that was installed using the PKG installer, run the following command from the terminal: `sudo bash /Library/Frameworks/GAMS.framework/Versions/46/Resources/uninstall.sh`. Additionally, you can remove GAMS from the PATH in your shell profile file (`~/.zprofile`, `~/.bash_profile`) if GAMS was added to the PATH during installation.

4.6.3 Installation using the self-extracting archive (osx_x64_64_sfx.exe or osx_arm64_sfx.exe)

The following instructions are for the macOS system on Intel/AMD CPUs (`x86_64`). For a macOS system on ARM64 CPUs, the same instructions apply, but with `x64_64` replaced by `arm64` in all file names.

1. Obtain the GAMS distribution file, which is available from http://www.gams.com/download, in one large self-extracting zip archive with a `_sfx.exe` file extension, e.g., `osx_x64_64_sfx.exe`. Check that it has the execute permission set. If you are not sure how to do this, just type in the command `chmod 755 osx_x64_64_sfx.exe` .

   Attention

   The common way to install GAMS on a Mac is the PKG installer. When one tries to run the sfx installer (e.g. for unattended installation) under macOS 10.15 (Catalina) or newer, the installer and several related files will be tagged with the `com.apple.quarantine` flag. There are different solutions to this problem:

   - The flag can be removed by the following command:
     `xattr -rd com.apple.quarantine osx_x64_64_sfx.exe`
   - Instead of downloading the sfx installer through the browser, use a command line tool such as curl (note that by downloading the software, you agree to the License Agreement). The download link may need to be adjusted, depending on the distribution that should be downloaded.
     `curl https://d37drm4t2jghv5.cloudfront.net/distributions/29.1.0/macosx/osx_x64_64_sfx.e`

2. Choose a location where you want to create the GAMS system directory (the GAMS system directory is the directory where the GAMS system files should reside). At this location the GAMS installer will create a subdirectory with a name that indicates the distribution of GAMS you are installing. For example, if you are installing the 24.3 distribution in `/Applications/GAMS`, the installer will create the GAMS system directory `/Applications/GAMS/gams24.3_osx_x64_64_sfx`. If the directory where you want to install GAMS is not below your home directory, you may need to have root privileges on the machine.

3. Create the directory that should contain the GAMS system directory, for instance `/Applications/GAMS`. Change to this directory (`cd /Applications/GAMS`). Make sure `pwd` returns the name of this directory correctly.

4. Run the distribution file, either from its current location or after transfering it to the directory that should contain the GAMS system directory. By executing the distribution file, the GAMS distribution should be extracted. For example, if you downloaded the distribution file into your home directory, you might execute the following commands:

   ```
   mkdir /Applications/GAMS
   cd    /Applications/GAMS
   ~/osx_x64_64_sfx.exe
   ```

5. Create the license file `gamslice.txt` in a directory GAMS searches to find a license. The license file is nowadays sent via email, with instructions. You can also request a demo license from http://www.gams.com/download.

Attention

Do not store the `gamslice.txt` in the GAMS system directory. This invalidates the code signature and cause Gatekeeper to reject the bundle!

GAMS searches a couple of system wide and user specific locations for a license file. For macOS these locations include `/Library/Application Support/GAMS` and `/Users/username/Library/Application Support/GAMS`. The locations can vary due to different system configuration. One can get an ordered list of data directories GAMS searches for `gamslice.txt` by running the program `./gamsinst -listdirs` from the GAMS system directory. Even though this list might contain locations in the system directory, e.g. `/Applications/GAMS31.1` we strongly discourage to place `gamslice.txt` here.

6. Change to the GAMS system directory and run the program `./gamsinst`. It will prompt you for default solvers to be used for each class of models. If possible, choose solvers you have licensed since unlicensed solvers will only run in demonstration mode. These solver defaults can be changed or overridden by:

   a. rerunning `./gamsinst` and resetting the default values

   b. setting a command line default, e.g., `gams trnsport lp=soplex`

   c. an option statement in the GAMS model, e.g: `option lp=soplex;`

7. Add the GAMS system directory to your path (see below).

8. To test the installation, log in as a normal user and run a few models from your home directory, but not the GAMS system directory:

```
LP:    trnsport (objective value:  153.675)
NLP:   chenery  (objective value:  1058.9)
MIP:   bid      (optimal solution: 15210109.512)
MINLP: procsel  (optimal solution: 1.9231)
MCP:   scarfmcp (no objective function)
MPSGE: scarfmge (no objective function)
```

9. If you move the GAMS system to another directory, remember to rerun `./gamsinst`. It is also good practice to rerun `./gamsinst` when you change your license file if this has changed the set of licensed solvers.

4.6.3.1 Access to GAMS

To run GAMS you must be able to execute the GAMS programs located in the GAMS system directory. There are several ways to do this. Remember that the GAMS system directory in the examples below may not correspond to the directory where you have installed your GAMS system.

1. If you are using the C shell (`csh`) and its variants you can modify your `.cshrc` file by adding the second of the two lines given below:

```
set path = (/your/previous/path/setting )
set path = ( $path /Applications/GAMS/gams24.3_osx_x64_64_sfx ) # new
```

2. Those of you using the Bourne (`sh`) or Korn (`ksh`) shells and their variants can modify their `.profile` file by adding the second of the three lines below:

```
PATH=/your/previous/path/setting
PATH=$PATH:/Applications/GAMS/gams24.3_osx_x64_64_sfx  # new
export PATH
```

   If the `.profile` file does not exist yet, it needs to be created. You should log out and log in again after you have made any changes to your path.

3. You may prefer to use an alias for the names of the programs instead of modifying the path as described above. C shell users can use the following commands on the command line or in their `.cshrc` file:

   ```
   alias gams /Applications/GAMS/gams24.3_osx_x64_64_sfx/gams
   alias gamslib /Applications/GAMS/gams24.3_osx_x64_64_sfx/gamslib
   ```

   The correct Bourne or Korn shell syntax (either command line or `.profile`) is:

   ```
   alias gams=/Applications/GAMS/gams24.3_osx_x64_64_sfx/gams
   alias gamslib=/Applications/GAMS/gams24.3_osx_x64_64_sfx/gamslib
   ```

   Again, you should log out and log in in order to put the alias settings in `.cshrc` or `.profile` into effect.

4. Casual users can always type the absolute path names of the GAMS programs, e.g.:

   ```
   /Applications/GAMS/gams24.3_osx_x64_64_sfx/gams trnsport
   ```

4.6.3.2 Example

The following shows the log of a session, where a user downloads a GAMS 24.3.1 system and installs it under `Applications/GAMS/gams24.3_osx_x64_64_sfx`. It is assumed that a GAMS license file has been stored as `/Users/doe/gamslice.txt`.

```
doe@mac:/Users/doe$ curl -L -k -O \
  http://d37drm4t2jghv5.cloudfront.net/distributions/24.3.1/macosx/osx_x64_64_sfx.exe
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  148M  100  148M    0     0  4374k      0  0:00:34  0:00:34 --:--:-- 5906k


doe@mac:/Users/doe$ chmod 755 osx_x64_64_sfx.exe


doe@mac:/Users/doe$cd /Applications/GAMS


doe@mac:/Applications/GAMS$ ~/osx_x64_64_sfx.exe
UnZipSFX 5.52 of 28 February 2005, by Info-ZIP (http://www.info-zip.org).
   creating: gams24.3_osx_x64_64_sfx/
  inflating: gams24.3_osx_x64_64_sfx/sp2full.m
  inflating: gams24.3_osx_x64_64_sfx/optpathnlp.def
  inflating: gams24.3_osx_x64_64_sfx/MessageReceiverWindow.exe
  inflating: gams24.3_osx_x64_64_sfx/datalib
  inflating: gams24.3_osx_x64_64_sfx/empsyntax.txt
  inflating: gams24.3_osx_x64_64_sfx/optlindoglobal.html
...
  inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/DomainChecking/DomainChecking.cs
  inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/DomainChecking/DomainChecking.csproj
  inflating: gams24.3_osx_x64_64_sfx/apifiles/CSharp/xp_example2.cs
  inflating: gams24.3_osx_x64_64_sfx/optdicopt.html

doe@mac:/Applications/GAMS$cd gams24.3_osx_x64_64_sfx
doe@mac:/Applications/GAMS/gams24.3_osx_x64_64_sfx$./gamsinst -license ~/gamslice.txt

-------------------------------------------------------
```

```
gamsinst run on Wed Jun 25 19:25:29 2014
GAMS sysdir is "/Applications/GAMS/gams24.3_osx_x64_64_sfx"

LP (Linear Programming) models can be solved by:
   1.           CBC  (demo or student license)
   2.        CONOPT  (demo or student license)
   3.         CPLEX  (demo or student license)
   4.        GUROBI  (demo or student license)
   5.         IPOPT  (demo or student license)
   6.        IPOPTH  (demo or student license)

...

Installed defaults:

         LP: CPLEX
        MIP: CPLEX
       RMIP: CPLEX
        NLP: CONOPT
        MCP: PATH
       MPEC: NLPEC
      RMPEC: NLPEC
        CNS: CONOPT
       DNLP: CONOPT
     RMINLP: CONOPT
      MINLP: DICOPT
        QCP: CONOPT
      MIQCP: SBB
     RMIQCP: CONOPT
        EMP: JAMS

We are now prepared to set read and execute permissions on the
GAMS system files.  If you are not sure which option to choose,
we recommend option 3.
You can set read (and execute) permission for:
 1. user only.
 2. user and group.
 3. user, group, and world.
 Enter your choice now



3



The files "gams", "gamslib", etc., are now executable.
You can run these commands in a number of ways:
  1. Call them using an absolute path (i.e. /Applications/GAMS/gams24.3_osx_x64_64_sfx/gams).
  2. Create your own aliases for them.
  3. Add the GAMS system directory "/Applications/GAMS/gams24.3_osx_x64_64_sfx" to your path.
Method 3. is recommended.



doe@mac:/Applications/GAMS/gams24.3_osx_x64_64_sfx$cd



doe@mac:/Users/doe$/Applications/GAMS/gams24.3_osx_x64_64_sfx/gamslib trnsport
 Copy ASCII: trnsport.gms
```

```
doe@mac:/Users/doe$/Applications/GAMS/gams24.3_osx_x64_64_sfx/gams trnsport.gms  license ~/gamslice.t
--- Job trnsport Start 06/26/14 11:24:56 24.3.1 r46409 DEX-DEG Mac x86_64/Darwin
GAMS 24.3.1   Copyright (C) 1987-2014 GAMS Development. All rights reserved
Licensee: ...
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.024
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
---   6 rows  7 columns  19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.114

IBM ILOG CPLEX   24.3.1 ... DEG Mac x86_64/Darwin
Cplex 12.6.0.0

Reading data...
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.02 sec. (0.00 ticks)

Iteration      Dual Objective            In Variable           Out Variable
     1             73.125000    x(seattle.new-york) demand(new-york) slack
     2            119.025000     x(seattle.chicago)  demand(chicago) slack
     3            153.675000    x(san-diego.topeka)   demand(topeka) slack
     4            153.675000  x(san-diego.new-york)  supply(seattle) slack
LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective :         153.675000

--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 06/26/14 11:24:57 elapsed 0:00:00.487
```

4.7 Installation Notes for Unix

4.7.1 Installation

To install GAMS, please follow the steps below as closely as possible. We advise you to read this entire document before beginning the installation procedure. Additionally, a video on how to install GAMS on Linux is available at https://www.youtube.com/watch?v=Mx_tYI3wyP4.

1. Obtain the GAMS distribution file, which is available from http://www.gams.com/latest, in one large self-extracting zip archive with a `_sfx.exe` file extension, e.g., `linux_x64_64_sfx.exe` on a Linux 64bit system. Check that it has the execute permission set. If you are not sure how to do this, just type in the command, e.g., `chmod 755 linux_x64_64_sfx.exe`.

2. Choose a location where you want to create the GAMS system directory (the GAMS system directory is the directory where the GAMS system files should reside). At this location the GAMS installer will create a subdirectory with a name that indicates the distribution of GAMS you are installing. For example, if you are installing the 24.3 distribution in `/opt/gams`, the installer will create the GAMS system directory `/opt/gams/gams24.3_linux_x64_64_sfx`. If the directory where you want to install GAMS is not below your home directory, you may need to have root privileges on the machine.

3. Create the directory that should contain the GAMS system directory, for instance `/opt/gams`. Change to this directory (`cd /opt/gams`). Make sure `pwd` returns the name of this directory correctly.

4. Run the distribution file, either from its current location or after transferring it to the directory that should contain the GAMS system directory. By executing the distribution file, the GAMS distribution should be extracted. For example, if you downloaded the distribution file into your home directory, you might execute the following commands:

```
mkdir /opt/gams
cd /opt/gams
~/linux_x64_64_sfx.exe
```

5. Create the license file `gamslice.txt` in a directory GAMS searches to find a license or use the corresponding GAMS Studio dialog. The license file is nowadays sent via email, with instructions. You can also request a demo license from http://www.gams.com/download.

   GAMS searches a couple of system wide and user specific standard locations for a license file.

6. Change to the GAMS system directory and run the program `./gamsinst`. It will prompt you for default solvers to be used for each class of models. If possible, choose solvers you have licensed since unlicensed solvers will only run in demonstration mode. These solver defaults can be changed or overridden by:

   a. rerunning `./gamsinst` and resetting the default values

   b. setting a command line default, e.g., `gams trnsport lp=soplex`

   c. an option statement in the GAMS model, e.g: `option lp=soplex;`

7. Add the GAMS system directory to your path (see below).

8. To test the installation, log in as a normal user and run a few models from your home directory, but not the GAMS system directory:

```
LP:    trnsport (objective value:  153.675)
NLP:   chenery  (objective value:  1058.9)
MIP:   bid      (optimal solution: 15210109.512)
MINLP: procsel  (optimal solution: 1.9231)
MCP:   scarfmcp (no objective function)
MPSGE: scarfmge (no objective function)
```

9. If you move the GAMS system to another directory, remember to rerun `./gamsinst`. It is also good practice to rerun `./gamsinst` when you change your license file if this has changed the set of licensed solvers.

4.7.2 Access to GAMS

To run GAMS you must be able to execute the GAMS programs located in the GAMS system directory. There are several ways to do this. Remember that the GAMS system directory in the examples below may not correspond to the directory where you have installed your GAMS system.

1. If you are using the C shell (`csh`) and its variants you can modify your `.cshrc` file by adding the line

```
set path = ( $path /opt/gams/gams24.3_linux_x64_64_sfx )
```

2. Those of you using the Bourne (`sh`) or Korn (`ksh`) shells and their variants can modify their `.profile` or `.bashrc` file by adding the line

   ```
   PATH=$PATH:/opt/gams/gams24.3_linux_x64_64_sfx
   ```

   If neither `.profile` nor `.bashrc` exist yet, `.profile` needs to be created. You should log out and log in again after you have made any changes to your path.

3. You may prefer to use an alias for the names of the programs instead of modifying the path as described above. C shell users can use the following commands on the command line or in their `.cshrc` file:

   ```
   alias gams /opt/gams/gams24.3_linux_x64_64_sfx/gams
   alias gamslib /opt/gams/gams24.3_linux_x64_64_sfx/gamslib
   ```

   The correct Bourne or Korn shell syntax (either command line or `.profile`) is:

   ```
   alias gams=/opt/gams/gams24.3_linux_x64_64_sfx/gams
   alias gamslib=/opt/gams/gams24.3_linux_x64_64_sfx/gamslib
   ```

   Again, you should log out and log in in order to put the alias settings in `.cshrc` or `.profile` into effect.

4. Casual users can always type the absolute path names of the GAMS programs, e.g.:

   ```
   /opt/gams/gams24.3_linux_x64_64_sfx/gams trnsport
   ```

4.8 Installation Notes for Windows

4.8.1 Installation

1. Run `windows_x64_64.exe` (Windows 64bit): This file is available from https://www.gams.com/latest. Please note that the installation may require administrative privileges on your machine.

   You have two options to run the installer: In default or advanced mode. In the default mode, the installer will prompt you for the name of the directory in which to install GAMS. We call this directory the `GAMS directory`. You may accept the default choice or pick another directory. Please remember: if you want to install two different versions of GAMS, they must be in separate directories.

   If you choose to use the advanced mode, the installer will also ask you for a name of a start menu folder, if GAMS should be installed for all users, if the `GAMS directory` should be added to the `PATH` environment variable and which desktop icons should be created.

   The `GAMS License` page allows to select a GAMS license. The installer tries to find GAMS licenses in different locations and automatically selects a license to be used with the following precedence:

   - Clipboard
   - `C:\Users\username\Documents\GAMS`
   - Previous GAMS installation found from the registry (User)
   - Previous GAMS installation found from the registry (System)

In addition it is possible to browse for a license file (`gamslice.txt`) or to skip all license related actions (`No License`) and install one after the installation process. Note that the use of GAMS requires a license. A free demo license can be requested at https://www.gams.com/download. A license file is written to `C:\Users\username\Documents\GAMS` when installing for `Current User` (default). Installing for `All Users` or checking `Write License to System Directory` will write the license to the GAMS system directory instead.

For automating the installation of GAMS, it is possible to provide the command line parameters `/SP- /SILENT`. This will install GAMS in a non-interactive manner using default settings. Note that depending on the security settings, the User Account Control asking for permission might still be active. The following command line parameters are available to further customize the non-interactive installation:

- `/installAllUsers=yes|no` (default:`no`): Can be used in order to install for `All Users` instead of `Current User`.
- `/noLicense=yes|no` (default:`no`): Can be used in order to prevent any GAMS license from being written.
- `/noIcons` can be used in order to check the `Don't create a Start Menu folder` checkbox on the `Select Start Menu Folder` page.
- `/desktopIcons=yes|no` (default:`undef`) can be used in order to turn on/off the creation of Desktop icons. The default (`undef`) is to create a desktop icon for `GAMS Studio` but not for `GAMS IDE`.
- `/allUsers`: Perform installation in administrative install mode.
- `/currentUser`: Perform installation in non administrative install mode. If specified, `/installAllUsers=yes` has no effect.

Attention

Under Windows 10 the Windows Installer package may not update system environment variables. To work around this issue, log off of Windows after the installation, and then log on again.

Note

The installer can be run unattended with the switches `/SILENT` or `/VERYSILENT` and might require some post installation task (like copying the license file to an appropriate location). This unattended mode also skips execution of some code in the installer that can cause failures on exotically configured systems.

2. During the installation of GAMS, the setup process helps in setting up the license to be used with GAMS. In case no license has been selected during this process it is required to set this up manually. The license file is nowadays sent via email, with instructions. You can also request a demo license from https://www.gams.com/download. If you don't have a license file yet, choose `No license` when asked for the GAMS license options. You can install a license later by either manually creating the license file `gamslice.txt` in a directory GAMS searches to find a license or use the corresponding GAMS Studio dialog.

   GAMS searches a couple of system wide and user specific standard locations for a license file.

3. Choose the default solvers: Run GAMS Studio by double clicking `studio.exe` from the GAMS system subdirectory `Studio`. To view or edit the default solvers, open the GAMS Configuration Editor in Studio. You can also skip this and thus accept the existing defaults if you wish, but most users want to select new default solvers for each model type.

4. Run a few models to test the GAMS system: The Model Library Explorer allows you to pick some models from the GAMS model library easily. After selecting one, press `F9` to run it, and view the solution. To test your installation, run the following models from the GAMS model library:

```
LP:    trnsport (objective value:  153.675)
NLP:   chenery  (objective value:  1058.9)
```

```
MIP:   bid      (optimal solution: 15210109.512)
MINLP: procsel  (optimal solution: 1.9231)
MCP:   scarfmcp (no objective function)
MPSGE: scarfmge (no objective function)
```

4.8.2 Visual C++ Redistributable Dependency

- Some solvers in the system as well as GAMS Studio have dependencies on certain Visual C++ libraries. These are present on most Windows systems but are missing on some. If you get a complaint about missing libraries on startup of GAMS Studio or when solving a model, please run the installer `vcredist_x64.exe` for these libraries, which can be found in the GAMS installation folder in directory `[GAMS system]\studio`.

4.8.3 Command Line Use of GAMS

Users wishing to use GAMS from the command line (aka the console mode) may want to perform the following steps. These steps are not necessary to run GAMS via Studio.

1. We recommend to add the GAMS directory to your environment path in order to avoid having to type in an absolute path name each time you run GAMS. Run the installer in advanced mode and mark the check-box `Add GAMS directory to PATH environment variable`.

2. Run the program `gamsinst`: `gamsinst` is a command line program used to configure GAMS. It prompts the user for default solvers to be used for each model type. If possible, choose solvers you have licensed, since unlicensed solvers will only run in demonstration mode. The solver defaults can be changed by:

   a. rerunning `gamsinst` and resetting the default values

   b. setting a command line default, e.g. `gams trnsport lp=soplex`

   c. by an option statement in the GAMS model, e.g: `option lp=soplex;`

   The system wide solver defaults are shared by the command line and GAMS Studio, so you can also choose to set these defaults using GAMS Studio.

4.8.4 Warning from Microsoft SmartScreen Filter

The SmartScreen is a feature that is used in several Microsoft products, including the Internet Explorer, Microsoft Edge, as well as in Windows 8 and newer versions of Windows. It is supposed to help to protect your computer against unwanted software. It uses a reputation-based system to rate files downloaded from the internet. That means that a file gets a better reputation when it gets download more often. As a result you might get a warning like this, especially for GAMS distributions that were recently released:

clicking on “More info” will display an option to continue the installation process.

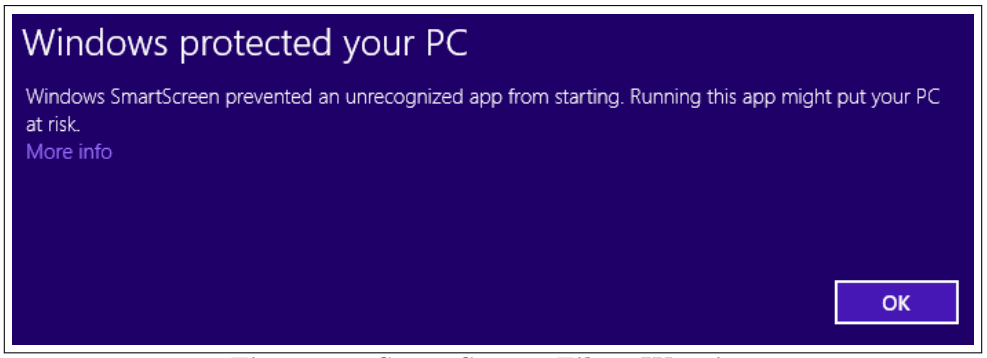


Figure 4.1 SmartScreen Filter Warning

4.9 Standard Locations

GAMS looks for specific files in some system-wide and user-specific locations which are independent of the GAMS installation. This allows the user to store these files in one place without the need to adjust anything when moving from one GAMS version to another. These folders are referred to as *standard locations* and are split into two groups: *configuration directories* and *data directories*. Here are some examples of files, which will be found in the data directories (or subfolders of these):

- The GAMS license file
- LibInclude files
- External Equation libraries
- Extrinsic Function libraries

The GAMS configuration file is expected in one of the configuration directories.

The exact location depends on the operating system used and user specific settings. Running `gamsinst -listdirs` on the command line shows the folders searched explicitly.

4.9.1 Standard Locations on macOS

4.9.1.1 Configuration Directories

These folders get searched on macOS (in the listed order):

- The GAMS system directory
- The subfolder `GAMS` in `$HOME/Library/Preferences` (`$HOME` usually contains the user's home directory)

Here is an example output of `gamsinst -listdirs`:

```
Config directories (searched for e.g. gamsconfig.yaml):
  /Users/JaneDoe/GAMS32
  /Users/JaneDoe/Library/Preferences/GAMS
```

4.9.1.2 Data Directories

These folders get searched on macOS (in the listed order):

- The subfolder `GAMS` in `$HOME/Library/Application Support` (`$HOME` usually contains the user's home directory)
- The subfolder `GAMS` in `/Library/Application Support`
- The subfolder `Resources` next to the GAMS system directory
- The GAMS system directory

Here is an example output of `gamsinst -listdirs`:

```
Data directories (searched for e.g. gamslice.txt):
  /Users/JaneDoe/Library/Application Support/GAMS
  /Library/Application Support/GAMS
  /Users/JaneDoe/Resources
  /Users/JaneDoe/GAMS32
```

4.9.2 Standard Locations on Unix

4.9.2.1 Configuration Directories

On Unix the environment variables `XDG_CONFIG_HOME` and `XDG_CONFIG_DIRS` (see XDG Base Directory Specification) influence the search directories. These folders get searched on Unix (in the listed order):

- The GAMS system directory
- The subfolder `GAMS` of folders from `$XDG_CONFIG_DIRS` or the subfolder `GAMS` in `/etc/xdg`
- The folder `$XDG_CONFIG_HOME` or the subfolder `GAMS` in `$HOME/.config` (`$HOME` usually contains the user's home directory)

Here is an example output of `gamsinst -listdirs`:

```
Config directories (searched for e.g. gamsconfig.yaml):
  /home/JohnDoe/GAMS32
  /etc/xdg/GAMS
  /home/JohnDoe/.config/GAMS
```

4.9.2.2 Data Directories

On Unix the environment variables `XDG_DATA_HOME` and `XDG_DATA_DIRS` (see XDG Base Directory Specification) influences the search directories. These folders get searched on Unix (in the listed order):

- The folder `$XDG_DATA_HOME` or the subfolder `GAMS` in `$HOME/.local/share` (`$HOME` usually contains the user's home directory)
- The subfolder `GAMS` of folders from `$XDG_DATA_DIRS` or the subfolder `GAMS` in `/usr/local/share` and the subfolder `GAMS` in `/usr/share`
- The GAMS system directory

Here is an example output of `gamsinst -listdirs`:

```
Data directories (searched for e.g. gamslice.txt):
  /home/JohnDoe/.local/share/GAMS
  /usr/local/share/GAMS
  /usr/share/GAMS
  /home/JohnDoe/GAMS32
```

4.9.3 Standard Locations on Windows

4.9.3.1 Configuration Directories

These folders get searched on Windows (in the listed order):

- The GAMS system directory
- The subfolder `GAMS` in `C:\ProgramData`
- The subfolder `GAMS` in `AppData\Local` in the User's directory
- The subfolder `GAMS` in `Documents` in the User's directory

Here is an example output of `gamsinst -listdirs`:

```
Config directories (searched for e.g. gamsconfig.yaml):
  C:\GAMS\32
  C:\ProgramData\GAMS
  C:\Users\JohnDoe\AppData\Local\GAMS
  C:\Users\JohnDoe\Documents\GAMS
```

4.9.3.2 Data Directories

These folders get searched on Windows (in the listed order):

- The subfolder `GAMS` in `Documents` in the User's directory
- The subfolder `GAMS` in `AppData\Roaming` in the User's directory
- The subfolder `GAMS` in `C:\ProgramData`
- The subfolder `data` in the GAMS system directory
- The subfolder `data\GAMS` in the GAMS system directory
- The GAMS system directory

Here is an example output of `gamsinst -listdirs`:

```
Data directories (searched for e.g. gamslice.txt):
  C:\Users\JohnDoe\Documents\GAMS
  C:\Users\JohnDoe\AppData\Roaming\GAMS
  C:\ProgramData\GAMS
  C:\GAMS\32\data
  C:\GAMS\32\data\GAMS
  C:\GAMS\32
```

Note: When running as a service, the "User's directory" mentioned above is not `C:\Users\UserName` but resolves to something like `C:\Windows\system32\config\systemprofile`. So the above output could look like this:

```
Data directories (searched for e.g. gamslice.txt):
  C:\Windows\system32\config\systemprofile\Documents\GAMS
  C:\Windows\system32\config\systemprofile\AppData\Roaming\GAMS
  C:\ProgramData\GAMS
  C:\GAMS\32\data
  C:\GAMS\32\data\GAMS
  C:\GAMS\32
```

4.10 Licensing

4.10.1 General Information

The GAMS system and all connected solvers can be downloaded from the GAMS website at https://www.gams.com/download. The use of the software is governed by the GAMS End User License Agreement, which is available on our website and can also be found in the GAMS System directory (eula.pdf). Without a valid GAMS license the system will not run and produces the following output:

```
*** No license specified and no gamslice.txt found in standard locations.
```

A free demo license time-limited for 5 months is included with the GAMS distribution. The demo license lets you generate and solve small models, see also Additional Limits for the Demo and Community License. On expiration you need to update your GAMS system (that comes with a new demo license) or request a demo license time-limited for 12 months at https://www.gams.com/download. Please note, a valid free demo license requires a recent GAMS version (not older than 18 months). The demo license of MIRO connector allows up to 10 indexed input and output symbols.

GAMS will terminate with an error message, if it hits one of the limits above:

```
*** Status: Terminated due to a licensing error
***         Inspect listing file for more information
```

When the time-limited demo license expires you need to update your GAMS system or request a new demo license at https://www.gams.com/download. Moreover, a valid free demo license requires a recent GAMS version (not older than 18 months).

4.10.2 GAMS Community License

Users can request a free community license from community@gams.com. The community license lets you generate and solve linear models (LP, MIP, and RMIP) that do not exceed 5000 variables and 5000 constraints. For all other model types the model cannot be larger than 2500 variables and 2500 constraints. Restrictions you experience with the demo license, i.e. additional limits enforced by some solvers, time-limitation of 12 months, and a recent GAMS system also apply to the community license. The free demo and community license cannot be combined with a professional license. These free demo and community licenses are for demonstration and evaluation but not for commercial and production work.

4.10.3 GAMS Licenses

GAMS licenses beyond the free demo and community license require a license for the GAMS Base Module and for most of the GAMS/Solvers and the GAMS/Solver-Links:

- The **GAMS Base Module** includes the GAMS Language Compiler, GAMS-APIs, many utilities, some solvers without size restrictions, and most solvers in "demo" mode with a model size limitation. See details below.
- A **GAMS/Solver** connects the GAMS Base module to a particular solver and includes a license for this solver to be used through GAMS. It is not necessary to install additional software. A **GAMS/Solver-Link** connects the GAMS Base Module to a particular solver, but does not include a license for the solver. It may be necessary to install additional software before the solver can be used.

We distinguish between academic and commercial licenses. Academic licenses can only be used for teaching and research at degree granting institutions. Moreover, we distinguish between time-limited and perpetual licenses. For perpetual licenses the first year of maintenance, support and updates is included in the initial purchase of the software. Maintained licenses qualify for free updates, adding components, platform-switching without additional charge, and multi-copy discounts on the same platform. After the first year maintenance can be extended by paying a fee. Licenses are typically limited to a single computer platform type (e.g. Windows or Linux), but there are also licenses available, which will work on all major platform supported by GAMS. Free academic licenses are available for certain solvers. Please visit the GAMS website for further information about the pricing of an appropriate license.

4.10.4 Installing or updating a license file

A GAMS license file is an ASCII file of six lines, which was sent to you via e-mail. Please copy all six lines into a file gamslice.txt and place this into a location searched by GAMS. Details can be found in the installation notes of the specific operating system:

- GAMS Installation Notes for Windows
- GAMS Installation Notes for macOS
- GAMS Installation Notes for Unix

Before installing a new GAMS version, please check whether it will work with your current GAMS license. The "Check for Update" functionality of GAMS Studio can be used to see if a newer GAMS version is available and what the newest version is that can be used with a given license.

Below is a sample output for a license, which is still under maintenance:

```
*** Processing GAMS system directory C:\gams\win64\24.8
*** Reading license file C:\ProgramData\GAMS\gamslice.txt
*** Your system is up to date
```

If your license is no longer under maintenance, you will receive a message like the one below:

```
*** Processing GAMS system directory C:\GAMS\win64\23.9
*** Reading license file C:\ProgramData\GAMS\gamslice.txt
*** Your license is 1276 days too old to run with the most recent system
*** The version of your GAMS system is 23.9.5
*** The last GAMS version you can use is 24.1.3
*** Please request the download of an older version from sales@gams.com
*** For ordering an update to use the most recent version (24.8.5)
*** please contact your distributor
```

4.10.5 License Problems

Errors or warnings triggered by a problem with the GAMS license file (gamslice.txt) are reported both in the log file and in the lst file. Below are some typical error and warning messages and instructions how to overcome them.

4.10.5.1 No License File present

Without a license file GAMS will not work. If you have received a license file (gamslice.txt), follow the instruction from the previous section. Do not try to rename it or to modify the contents of the license file. Without a license GAMS will give you an error message like the one below:

```
*** No license specified and no gamslice.txt found in standard locations.
```

4.10.5.2 License File Invalid or Corrupted

Running GAMS with an invalid license will give you an error message like the one below:

```
*** License File validation failed
```

Do not try to modify your license file, any change will break it. If the original license file is no longer available, please contact sales@gams.com and ask for a copy of your license file. Please attach the license file you are currently using. This error also pops up, if you are using a license file, which was issued for a different platform.

4.10.5.3 License File expired

If are trying to run an old license file with a newer GAMS distribution, you may get an error message stating:

```
*** License file too old for this version of GAMS.
*** Maintenance expired 2840 days ago.
*** More than 60 days since expiration, sorry...
*** Run an older GAMS system or renew the license
```

Replace that GAMS system with the one you got together with your license file (use the "Check for Update" utility mentioned above to find out the latest version you can use with the current license; old systems are available on request from sales@gams.com) or update your license to the current version.

4.10.5.4 No License (for a particular Solver) found

If one of the solvers you are trying to use (or is selected as the default solver) is not included in your license file and the size of the model exceed the limits of the free demo or community version, you will get an error stating:

```
*** No license found
*** To update your license, please contact your distributor.
```

Check whether the default solvers for a particular problem class are included in the license. Please note that some of the solvers call other solvers as a sub-solver, e.g.:

- GAMS/DICOPT requires an NLP and a MIP solver
- GAMS/SBB requires an NLP solver
- MPSGE: If GAMS/PATH is not included in your license, please select GAMS/MILES, which is included in the base system as the default solver for MCP problems.

4.10.6 Warnings

4.10.6.1 License file too old for this version of GAMS

The complete message is:

```
*** License file too old for this version of GAMS.
*** Maintenance expired xx days ago.
```

If GAMS continues working, you can ignore this warning. However, any forthcoming GAMS distribution will most likely not work, but downgrade to the limits of the free demo system.

4.10.6.2 License File has expired xx days ago

This warning (without further error messages) indicates that the time limited license (e.g. an evaluation license) will stop working soon.

4.10.7 The GAMS/BASE Module

The GAMS/BASE module includes:

- **Language Compiler and Execution System**
- GAMS IDE**(Integrated Development Environment) (Windows only)**
  - Project Management
  - Editor, Syntax coloring, Spell checking
  - Launching and monitoring of (multiple) GAMS processes
  - Listing file / Tree view / Syntax-error navigation
  - Solver selection / option selection
  - GDX Viewer: Data cube and export (e.g. to MS Excel), charting facilities
- **GAMS Studio (platform independent, Beta version)**
  - Editor, Syntax coloring
  - Launching and monitoring of (multiple) GAMS processes
  - Listing file / Tree view / Syntax-error navigation
  - GDX Viewer
- **GAMS Free Solvers and Links**
  - Open Source (COIN-OR): CBC, Ipopt, SHOT
  - CONVERT, JAMS and LOGMIP, NLPEC
  - MILES
  - EXAMINER, GAMSCHK
  - Academic licenses only: ODHeuristic (requires a GAMS/CPLEX or GAMS/CPLEX-Link license), SCIP, Soplex
  - GAMS/KESTREL for using the NEOS Server with a local GAMS system

- **Most other solvers in limited versions for demonstration and evaluation, but no commercial use:** 2000 constraints and variables for linear models, and 1000 constraints and variables for other model types. For licensed academic users the size limits are increased: 5000 constraints and variables for linear models, and 2500 constraints and variables for other model types. The solvers might enforce additional limits.
- **EMP (Extended Mathematical Programming Framework)**
- **Posix Utilities**
- Academic licenses only: **MIRO Connector** for using external input and output symbols.
- **GDX (GAMSData eXchange) and related GDX Utilities**
  - Binary data exchange between application, GAMS, and the solver (fast, saves disk space, tailored for large sparse matrices, platform independent, direct GDX interfaces, API support for high-level programming languages)
  - GDX Utilities: GDX Viewer, GDXMERGE, GDXDUMP, GDXCOPY, GDXDIFF, MDB2GMS, GDXMRW, GDXRRW, GDXXRW, GDX2XLS, XLSDump
- **Various other tools**
- GAMS APIs
  - Expert-Level GAMS APIs
    - GAMS
    - GDX
    - Option
    - Supported Languages: C, C++, C#, Delphi, Fortran, Java, Python, VBA, VB.Net
  - Object-Oriented GAMS APIs
    - Supported Languages: C++, C#, Java, Python, VB.Net
- **Documentation**
- **Model Libraries**

4.10.8 Additional Limits for the Demo and Community License

The model size limits enforced by GAMS are:

- 2000 variables and 2000 constraints for linear (LP, RMIP, and MIP) models with a demo license
- 1000 variables and 1000 constraints for all other model types with a demo license
- 5000 variables and 5000 constraints for linear (LP, RMIP, and MIP) models with a community license
- 2500 variables and 2500 constraints for all other model types with a community license

In addition to the GAMS model size limits, the solvers might impose stricter limits when running with a demo or community license. We use m for the number constraints, n for the number variables, nd for the number of discrete variables, nz for the number nonzeros, and $nlnz$ for the number nonlinear nonzeros:

- Antigone, Baron, and LindoGlobal require $m \leq 50$, $n \leq 50$, and $nlnz \leq 50$ with a demo license.
- Antigone, Baron, and LindoGlobal require $m \leq 300$, $n \leq 300$, and $nlnz \leq 100$ with a community license.
- Cplex and ODHCPLEX require $m \leq 2000$ and $n \leq 2000$ with a *pro-bono* community license.
- Decis and Knitro require $m \leq 300$, $n \leq 300$, $nd \leq 50$, $nz \leq 2000$, and $nlnz \leq 1000$ with a demo and community license.
- Xpress requires $m+n \leq 5000$ with a community license.
- In addition Gurobi requires a license file from the vendor to operate.

4.11 A GAMS Tutorial by Richard E. Rosenthal

4.11.1 Introduction

Richard E. Rosenthal of the Naval Postgraduate School in Monterey, California wrote a detailed example of the use of GAMS for formulating, solving, and analyzing a small and simple optimization problem. The example is a quick overview of GAMS and its main features. Many references are made to other parts of the documentation, but they are only to tell you where to look for more details; the material here can be read profitably without reference to the rest of the documentation.

The example is an instance of the transportation problem of linear programming, which has historically served as a *'laboratory animal'* in the development of optimization technology. [See, for example, Dantzig (1963) [1].] It is a good choice for illustrating the power of algebraic modeling languages like GAMS because the transportation problem, no matter how large the instance at hand, possesses a simple, exploitable algebraic structure. You will see that almost all of the statements in the GAMS input file we are about to present would remain unchanged if a much larger transportation problem were considered.

In the familiar transportation problem, we are given the supplies at several plants and the demands at several markets for a single commodity, and we are given the unit costs of shipping the commodity from plants to markets. The economic question is: how much shipment should there be between each plant and each market so as to minimize total transport cost?

The algebraic representation of this problem is usually presented in a format similar to the following.

Indices:

i = plants
j = markets

Given Data:

a_i = supply of commodity of plant i (in cases)
b_j = demand for commodity at market j
c_{ij} = cost per unit shipment between plant i and market j

Decision Variables:

x_{ij} = amount of commodity to ship from plant i to market j
where $x_{ij} \geq 0$, for all i, j

Constraints:

Observe supply limit at plant i: $\sum_j x_{ij} \leq a_i$ for all i (cases)
Satisfy demand at market j: $\sum_i x_{ij} \geq b_j$ for all j (cases)
Objective Function: Minimize $\sum_i \sum_j c_{ij} x_{ij}$ ($K)

[1] Dantzig, George B. (1963). *Linear Programming and Extensions*, Princeton University Press, Princeton N.J.

Note that this simple example reveals some modeling practices that we regard as good habits in general and that are consistent with the design of GAMS. First, all the entities of the model are identified (and grouped) by type. Second, the ordering of entities is chosen so that no symbol is referred to before it is defined. Third, the units of all entities are specified, and, fourth, the units are chosen to a scale such that the numerical values to be encountered by the optimizer have relatively small absolute orders of magnitude. (The symbol $K here means thousands of dollars.)

The names of the types of entities may differ among modelers. For example, economists use the terms *exogenous variable* and *endogenous variable* for *given data* and *decision variable*, respectively. In GAMS, the terminology adopted is as follows: indices are called `sets`, given data are called `parameters`, decision variables are called `variables`, and constraints and the objective function are called `equations`.

The GAMS representation of the transportation problem closely resembles the algebraic representation above. The most important difference, however, is that the GAMS version can be read and processed by a computer.

Table 1: Data for the transportation problem (adapted from Dantzig, 1963) illustrates Shipping Distances from Plants to Markets (1000 miles) as well as Market Demands and Plant Supplies.

Plants ↓	**New York**	**Chicago**	**Topeka**	←***Markets***
Seattle	2.5	1.7	1.8	350
San Diego	2.5	1.8	1.4	600
Demands →	325	300	275	*Supplies* ↑

As an instance of the transportation problem, suppose there are two canning plants and three markets, with the data given in table Table 1. Shipping distances are in thousands of miles, and shipping costs are assumed to be $90.00 per case per thousand miles. The GAMS representation of this problem is as follows:

```
$title a transportation model
Sets
     i   canning plants   / seattle, san-diego /
     j   markets          / new-york, chicago, topeka / ;

Parameters

     a(i)  capacity of plant i in cases
       /    seattle     350
            san-diego   600  /

     b(j)  demand at market j in cases
       /    new-york    325
            chicago     300
            topeka      275  / ;

Table d(i,j)  distance in thousands of miles
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4  ;

Scalar f  freight in dollars per case per thousand miles  /90/ ;

Parameter c(i,j)  transport cost in thousands of dollars per case ;

          c(i,j) = f * d(i,j) / 1000 ;

Variables
```

```
     x(i,j)  shipment quantities in cases
     z       total transportation costs in thousands of dollars ;

Positive Variable x ;

Equations
     cost        define objective function
     supply(i)   observe supply limit at plant i
     demand(j)   satisfy demand at market j ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;
```

If you submit a file containing the statements above as input to GAMS , the transportation model will be formulated and solved. Details vary on how to invoke GAMS on different of computers, but the simplest ('*no frills*') way to call GAMS is to enter the word GAMS followed by the input file's name. You will see a number of terse lines describing the progress GAMS is making, including the name of the file onto which the output is being written. When GAMS has finished, examine this file, and if all has gone well the optimal shipments will be displayed at the bottom as follows.

```
             new-york      chicago       topeka
seattle        50.000      300.000
san-diego     275.000                   275.000
```

You will also receive the marginal costs (simplex multipliers) below.

```
              chicago       topeka
seattle                      0.036
san-diego       0.009
```

These results indicate, for example, that it is optimal to send nothing from Seattle to Topeka, but if you insist on sending one case it will add .036 $K (or $36.00) to the optimal cost.

4.11.2 Structure of a GAMS Model

For the remainder of the tutorial, we will discuss the basic components of a GAMS model, with reference to the example above. The basic components are listed in table Table 2.

Table 2: The basic components of a GAMS model

Type	Component
Inputs	Sets Declaration Assignment of members
	Data (`Parameters, Tables, Scalars`) Declaration Assignment of values
	Variables Declaration Assignment of type
	Assignment of Variable Bounds and/or Initial Values (optional)
	Equations Declaration Definition
	Model and Solve Statements
	Display Statements (optional)
Outputs	Echo Prints
	Reference Maps
	Equation Listings
	Status Reports
	Solution Reports

There are optional input components, such as edit checks for bad data and requests for customized reports of results. Other optional advanced features include saving and restoring old models, and creating multiple models in a single run, but this tutorial will discuss only the basic components.

Before treating the individual components, we give a few general remarks.

1. A GAMS model is a collection of statements in the GAMS Language. The only rule governing the ordering of statements is that an entity of the model cannot be referenced before it is declared to exist.
2. GAMS statements may be laid out typographically in almost any style that is appealing to the user. Multiple lines per statement, embedded blank lines, and multiple statements per line are allowed. You will get a good idea of what is allowed from the examples in this tutorial, but precise rules of the road are given in the next Chapter.
3. When you are a beginning GAMS user, you should terminate every statement with a semicolon, as in our examples. The GAMS compiler does not distinguish between upper-and lowercase letters, so you are free to use either.
4. Documentation is crucial to the usefulness of mathematical models. It is more useful (and most likely to be accurate) if it is embedded within the model itself rather than written up separately. There are at least two ways to insert documentation within a GAMS model. First, any line that starts with an asterisk in column 1 is disregarded as a comment line by the GAMS compiler. Second, perhaps more important, documentary text can be inserted within specific GAMS statements.
5. As you can see from the list of input components above, the creation of GAMS entities involves two steps: a declaration and an assignment or definition. *Declaration* means declaring the existence of something and giving it a name. *Assignment* or *definition* means giving something a specific value or form. In the case of equations, you must make the declaration and definition in separate GAMS statements. For all other GAMS entities, however, you have the option of making declarations and assignments in the same statement or separately.
6. The names given to the entities of the model must start with a letter and can be followed by up to 62 more letters or digits.

4.11.3 Sets

Sets are the basic building blocks of a GAMS model, corresponding exactly to the indices in the algebraic representations of models. The Transportation example above contains just one `Set` statement:

```
Sets
     i   canning plants   / seattle, san-diego /
     j   markets          / new-york, chicago, topeka / ;
```

The effect of this statement is probably self-evident. We declared two sets and gave them the names `i` and `j`. We also assigned members to the sets as follows:

i = {Seattle, San Diego}
j = {New York, Chicago, Topeka}.

You should note the typographical differences between the GAMS format and the usual mathematical format for listing the elements of a set. GAMS uses slashes '/' rather than curly braces '{}' to delineate the set. Note also that multiword names like 'New York' have to be quoted ((e.g. 'New York' or "New York") or you must use hyphens instead of blanks (e.g. New-York').

The words after the set names in the `sets` statement above are called *text*. Text is optional. It is there only for internal documentation, serving no formal purpose in the model. The GAMS compiler makes no attempt to interpret the text, but it saves the text and '*parrots*' it back to you at various times for your convenience.

It was not necessary to combine the creation of sets i and j in one statement. We could have put them into separate statements as follows:

```
Set    i   canning plants   / seattle, san-diego / ;
Set    j   markets          / new-york, chicago, topeka / ;
```

The placement of blank spaces and lines (as well as the choice of upper- or lowercase) is up to you. Each GAMS user tends to develop individual stylistic conventions. (The use of the singular `set` is also up to you. Using `set` in a statement that makes a single declaration and `sets` in one that makes several is good English, but GAMS treats the singular and plural synonymously.)

A convenient feature to use when you are assigning members to a set is the asterisk. It applies to cases when the elements follow a sequence. For example, the following are valid `set` statements in GAMS.

```
Set    t   time periods         /1991*2000/;
Set    m   machines             /mach1*mach24/;
```

Here the effect is to assign

t = {1991,1992,1993, ..., 2000}
m = {mach $_1$, mach $_2$,....., mach $_{24}$ }.

Note that set elements are stored as character strings, so the elements of t are not numbers.

Another convenient feature is the alias statement, which is used to give another name to a previously declared set. In the following example:

```
Alias (t,tp);
```

the name `tp` is like a t' in mathematical notation. It is useful in models that are concerned with the interactions of elements within the same set.

The sets `i`, `j`, `t`, and `m` in the statements above are examples of static sets, i.e., they are assigned their members directly by the user and do not change. GAMS has several capabilities for creating dynamic sets, which acquire their members through the execution of set-theoretic and logical operations. Dynamic sets are discussed in chapter Dynamic Sets. Another valuable advanced feature is multidimensional sets, which are discussed in section Multi-Dimensional Sets.

4.11.4 Data

The GAMS model of the transportation problem demonstrates three of the different formats that are allowable for entering data. The three formats are:

- Lists
- Tables
- Direct assignments

The next three subsections will discuss each of these formats in turn.

4.11.4.1 Data Entry by Lists

The first format is illustrated by the first Parameters statement of the example, which is repeated below.

```
Parameters

     a(i)  capacity of plant i in cases
       /    seattle     350
            san-diego   600  /

     b(j)  demand at market j in cases
       /    new-york    325
            chicago     300
            topeka      275  / ;
```

This statement has several effects. Again, they may be self-evident, but it is worthwhile to analyze them in detail. The statement declares the existence of two parameters, gives them the names `a` and `b`, and declares their *domains* to be `i` and `j`, respectively. (A domain is the set, or tuple of sets, over which a parameter, variable, or equation is defined.) The statement also gives documentary text for each parameter and assigns values of `a(i)` and `b(j)` for each element of `i` and `j`. It is perfectly acceptable to break this one statement into two, if you prefer, as follows.

```
Parameters a(i)  capacity of plant i in cases
     / seattle     350
       san-diego   600  / ;

Parameters b(j)  demand at market j in cases
    / new-york    325
      chicago     300
      topeka      275  / ;
```

Here are some points to remember when using the list format.

1. The list of domain elements and their respective parameter values can be laid out in almost any way you like. The only rules are that the entire list must be enclosed in slashes and that the element-value pairs must be separated by commas or entered on separate lines.

2. There is no semicolon separating the element-value list from the name, domain, and text that precede it. This is because the same statement is being used for declaration and assignment when you use the list format. (An element-value list by itself is not interpretable by GAMS and will result in an error message.)

3. The GAMS compiler has an unusual feature called domain checking, which verifies that each domain element in the list is in fact a member of the appropriate set. For example, if you were to spell 'Seattle' correctly in the statement declaring `Set i` but misspell it as 'Seatle' in a subsequent element-value list, the GAMS compiler would give you an error message that the element 'Seatle' does not belong to the set `i`.

4. Zero is the default value for all parameters. Therefore, you only need to include the nonzero entries in the element-value list, and these can be entered in any order.

5. A scalar is regarded as a parameter that has no domain. It can be declared and assigned with a `Scalar` statement containing a *degenerate* list of only one value, as in the following statement from the transportation model.

```
Scalar f freight in dollars per case per thousand miles /90/;
```

If the domain of a parameter has two or more dimensions, it can still have its values entered by the list format. This is very useful for entering arrays that are sparse (having few non-zeros) and super-sparse (having few distinct non-zeros).

4.11.4.2 Data Entry by Tables

Optimization practitioners have noticed for some time that many of the input data for a large model are derived from relatively small tables of numbers. Thus, it is very useful to have the table format for data entry. An example of a two-dimensional table (or matrix) is provided in the transportation model:

```
Table d(i,j)  distance in thousands of miles
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4  ;
```

The effect of this statement is to declare the parameter `d` and to specify its domain as the set of ordered pairs in the Cartesian product of `i` and `j`. The values of `d` are also given in this statement under the appropriate heading. If there are blank entries in the table, they are interpreted as zeroes.

As in the list format, GAMS will perform domain checking to make sure that the row and column names of the table are members of the appropriate sets. Formats for entering tables with more columns than you can fit on one line and for entering tables with more than two dimensions are given in Chapter Data Entry: Parameters, Scalars and Tables.

4.11.4.3 Data Entry by Direct Assignment

The direct assignment method of data entry differs from the list and table methods in that it divides the tasks of parameter declaration and parameter assignment between separate statements. The transportation model contains the following example of this method.

```
Parameter c(i,j)  transport cost in thousands of dollars per case ;
          c(i,j) = f * d(i,j) / 1000 ;
```

It is important to emphasize the presence of the semicolon at the end of the first line. Without it, the GAMS compiler would attempt to interpret both lines as parts of the same statement. (GAMS would fail to discern a valid interpretation, so it would send you an error message.)

The effects of the first statement above are to declare the parameter `c`, to specify the domain `(i,j)`, and to provide some documentary text. The second statement assigns to `c(i,j)` the product of the values of the parameters `f` and `d(i,j)`. Naturally, this is legal in GAMS only if you have already assigned values to `f` and `d(i,j)` in previous statements.

The direct assignment above applies to all `(i,j)` pairs in the domain of `c`. If you wish to make assignments for specific elements in the domain, you enclose the element names in quotes. For example,

```
c('Seattle','New-York')  = 0.40;
```

is a valid GAMS assignment statement.

The same parameter can be assigned a value more than once. Each assignment statement takes effect immediately and overrides any previous values. (In contrast, the same parameter may not be declared more than once. This is a GAMS error check to keep you from accidentally using the same name for two different things.)

The right-hand side of an assignment statement can contain a great variety of mathematical expressions and built-in functions. If you are familiar with a scientific programming language such as C, you will have no trouble in becoming comfortable writing assignment statements in GAMS. (Notice, however, that GAMS has some efficiencies not shared by C. For example, we were able to assign `c(i,j)` values for all `(i,j)` pairs without constructing '*loops*'.)

The GAMS standard operations and supplied functions are given later. Here are some examples of valid assignments. In all cases, assume the left-hand-side parameter has already been declared and the right-hand-side parameters have already been assigned values in previous statements.

```
csquared       = sqr(c);
e              = m*csquared;
w              = 1/lamda;
eoq(i)         = sqrt( 2*demand(i)*ordcost(i)/holdcost(i));
t(i)           = min(p(i), q(i)/r(i), log(s(i)));
euclidean(i,j) = qrt(sqr(xi(i) - xi(j) + sqr(x2(i) - x2(j)));
present(j)     = future(j)*exp(-interest*time(j));
```

The summation and product operators to be introduced later can also be used in direct assignments.

4.11.5 Variables

The decision variables (or endogenous variables) of a GAMS-expressed model must be declared with a `Variables` statement. Each variable is given a name, a domain if appropriate, and (optionally) text. The transportation model contains the following example of a `Variables` statement.

```
Variables
    x(i,j)  shipment quantities in cases
    z       total transportation costs in thousands of dollars ;
```

This statement results in the declaration of a shipment variable for each `(i,j)` pair. (You will see in chapter Equations, how GAMS can handle the typical real-world situation in which only a subset of the `(i,j)` pairs is allowable for shipment.)

The `z` variable is declared without a domain because it is a scalar quantity. Every GAMS optimization model must contain one such variable to serve as the quantity to be minimized or maximized.

Once declared, every variable must be assigned a type. A selection of permissible types are given in table Table 3. For a full list, see section Variable Types.

Table 3 : Permissible variable types

Variable Type	*Allowed Range of Variable*
`free(default)`	$-\infty$ to $+\infty$
`positive`	0 to $+\infty$
`negative`	$-\infty$ to 0
`binary`	0 or 1
`integer`	$0, 1, \ldots, 100$ (default)

The variable that serves as the quantity to be optimized must be a scalar and must be of the `free` type. In our transportation example, `z` is kept free by default, but `x(i,j)` is constrained to non-negativity by the following statement.

```
Positive variable x ;
```

Note that the domain of `x` should not be repeated in the type assignment. All entries in the domain automatically have the same variable type.

Section The .lo, .l, .up, .m Database describes how to assign lower bounds, upper bounds, and initial values to variables

4.11.6 Equations

The power of algebraic modeling languages like GAMS is most apparent in the creation of the equations and inequalities that comprise the model under construction. This is because whenever a group of equations or inequalities has the same algebraic structure, all the members of the group are created simultaneously, not individually.

4.11.6.1 Equation Declaration

Equations must be declared and defined in separate statements. The format of the declaration is the same as for other GAMS entities. First comes the keyword, `Equations` in this case, followed by the name, domain and text of one or more groups of equations or inequalities being declared. Our transportation model contains the following equation declaration:

```
Equations
     cost        define objective function
     supply(i)   observe supply limit at plant i
     demand(j)   satisfy demand at market j ;
```

Keep in mind that the word `Equation` has a broad meaning in GAMS. It encompasses both equality and inequality relationships, and a GAMS equation with a single name can refer to one or several of these relationships. For example, `cost` has no domain so it is a single equation, but `supply` refers to a set of inequalities defined over the domain `i`.

4.11.6.2 GAMS Summation (and Product) Notation

Before going into equation definition we describe the summation notation in GAMS. Remember that GAMS is designed for standard keyboards and line-by-line input readers, so it is not possible (nor would it be convenient for the user) to employ the standard mathematical notation for summations.

The summation notation in GAMS can be used for simple and complex expressions. The format is based on the idea of always thinking of a summation as an operator with two arguments: `Sum(index of summation, summand)` A comma separates the two arguments, and if the first argument requires a comma then it should be in parentheses. The second argument can be any mathematical expression including another summation.

As a simple example, the transportation problem contains the expression

```
Sum(j, x(i,j))
```

that is equivalent to $\sum_j x_{ij}$.

A slightly more complex summation is used in the following example:

```
Sum((i,j), c(i,j)*x(i,j))
```

that is equivalent to $\sum_i \sum_j c_{ij} x_{ij}$.

The last expression could also have been written as a nested summation as follows:

```
Sum(i, Sum(j, c(i,j)*x(i,j)))
```

In section Conditional Indexed Operations, we describe how to use the dollar operator to impose restrictions on the summation operator so that only the elements of `i` and `j` that satisfy specified conditions are included in the summation.

Products are defined in GAMS using exactly the same format as summations, replacing `Sum` by `Prod`. For example,

```
prod(j, x(i, j))
```

is equivalent to: $\Pi_j x_{ij}$.

Summation and product operators may be used in direct assignment statements for parameters. For example,

```
scalar totsupply   total supply over all plants;
totsupply = sum(i, a(i));
```

4.11.6.3 Equation Definition

Equation definitions are the most complex statements in GAMS in terms of their variety. The components of an equation definition are, in order:

1. The name of the equation being defined
2. The domain
3. Domain restriction condition (optional)
4. The symbol '..'
5. Left-hand side expression
6. Relational operator: =l=, =e=, =g= or others. For a complete list, see Table Equation Types.
7. Right-hand side expression

The transportation example contains three of these statements.

```
cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;
```

Here are some points to remember.

- The power to create multiple equations with a single GAMS statement is controlled by the domain. For example, the definition for the demand constraint will result in the creation of one constraint for each element of the domain j, as shown in the following excerpt from the GAMS output.

  ```
  demand(new-york)..x(seattle,new-york) + x(san-diego,new-york)=g=325 ;
  demand(chicago).. x(seattle,chicago) + x(san-diego,chicago)  =g=300 ;
  demand(topeka)..  x(seattle,topeka) + x(san-diego,topeka)    =g=275 ;
  ```

- The key idea here is that the definition of the demand constraints is exactly the same whether we are solving the toy-sized example above or a 20,000-node real-world problem. In either case, the user enters just one generic equation algebraically, and GAMS creates the specific equations that are appropriate for the model instance at hand. (Using some other optimization packages, something like the extract above would be part of the input, not the output.)
- In many real-world problems, some of the members of an equation domain need to be omitted or differentiated from the pattern of the others because of an exception of some kind. GAMS can readily accommodate this loss of structure using a powerful feature known as the *dollar* or '*such-that*' operator, which is not illustrated here. The domain restriction feature can be absolutely essential for keeping the size of a real-world model within the range of solvability.
- The relational operators have the following meanings:

=l=	less than or equal to
=g=	greater than or equal to
=e=	equal to

- It is important to understand the difference between the symbols '=' and '=e='. The '=' symbol is used only in direct assignments, and the '=e=' symbol is used only in equation definitions. These two contexts are very different. A direct assignment gives a desired value to a parameter before the solver is called. An equation definition also describes a desired relationship, but it cannot be satisfied until after the solver is called. It follows that equation definitions must contain variables and direct assignments must not.

- Variables can appear on the left or right-hand side of an equation or both. The same variable can appear in an equation more than once. The GAMS processor will automatically convert the equation to its equivalent standard form (variables on the left, no duplicate appearances) before calling the solver.
- An equation definition may appear anywhere in the GAMS input, provided the equation and all variables and parameters to which it refers were previously declared. (Note that it is permissible for a parameter appearing in the equation to be assigned or reassigned a value after the definition. This is useful when doing multiple model runs with one GAMS input.) The equations need not be defined in the same order in which they were declared.

4.11.7 Objective Function

This is just a reminder that GAMS has no explicit entity called the *objective function*. To specify the function to be optimized, you must create a variable, which is free (unconstrained in sign) and scalar-valued (has no domain) and which appears in an equation definition that equates it to the objective function.

4.11.8 Model and Solve Statements

The word `model` has a very precise meaning in GAMS. It is simply a collection of equations. Like other GAMS entities, it must be given a name in a declaration. The format of the declaration is the keyword `model` followed by the name of the model, followed by a list of equation names enclosed in slashes. If all previously defined equations are to be included, you can enter `/all/` in place of the explicit list. In our example, there is one `model` statement:

```
model transport  /all/  ;
```

This statement may seem superfluous, but it is useful to advanced users who may create several models in one GAMS run. If we were to use the explicit list rather than the shortcut `/all/`, the statement would be written as

```
model transport / cost, supply, demand / ;
```

The domains are omitted from the list since they are not part of the equation name. The list option is used when only a subset of the existing equations comprises a specific model (or sub-model) being generated.

Once a model has been declared and assigned equations, we are ready to call the solver. This is done with a `solve` statement, which in our example is written as

```
solve transport using lp minimizing z ;
```

The format of the solve statement is as follows:

1. The key word `solve`
2. The name of the model to be solved
3. The key word `using`
4. An available solution procedure. A complete list is given below. For further details, see section Classification of Models.

Solution	Description
lp	for linear programming
qcp	for quadratic constraint programming
nlp	for nonlinear programming
dnlp	for nonlinear programming with discontinuous derivatives
mip	for mixed integer programming
rmip	for relaxed mixed integer programming
miqcp	for mixed integer quadratic constraint programming
rmiqcp	for relaxed mixed integer quadratic constraint programming
minlp	for mixed integer nonlinear programming
rminlp	for relaxed mixed integer nonlinear programming
mcp	for mixed complementarity problems
mpec	for mathematical programs with equilibrium constraints
rmpec	for relaxed mathematical program with equilibrium constraints
cns	for constrained nonlinear systems
emp	for extended mathematical programming

5. The keyword '`minimizing`' or '`maximizing`'
6. The name of the variable to be optimized

4.11.9 Display Statements

The `solve` statement will cause several things to happen when executed. The specific instance of interest of the model will be generated, the appropriate data structures for inputting this problem to the solver will be created, the solver will be invoked, and the output from the solver will be printed to a file. To get the optimal values of the primal and/or dual variables, we can look at the solver output, or, if we wish, we can request a display of these results from GAMS. Our example contains the following statement:

```
display x.l, x.m ;
```

that calls for a printout of the final levels, `x.l`, and marginal (or reduced costs), `x.m`, of the shipment variables, `x(i,j)`. GAMS will automatically format this printout in to dimensional tables with appropriate headings.

4.11.10 The .lo, .l, .up, .m Database

GAMS was designed with a small database system in which records are maintained for the variables and equations. The most important fields in each record are:

`.lo` lower bound
`.l` level or primal value
`.up` upper bound
`.m` marginal or dual value

The format for referencing these quantities is the variable or equation's name followed by the field's name, followed (if necessary) by the domain (or an element of the domain).

GAMS allows the user complete read-and write-access to the database. This may not seem remarkable to you now, but it can become a greatly appreciated feature in advanced use. Some examples of use of the database follow.

4.11.10.1 Assignment of Variable Bounds and/or Initial Values

The lower and upper bounds of a variable are set automatically according to the variable's type (`free, positive, negative, binary`, or `integer`), but these bounds can be overwritten by the GAMS user. Some examples follow.

```
x.up(i,j)    =  capacity(i,j)  ;
x.lo(i,j)    =  10.0  ;
x.up('seattle','new-york') = 1.2*capacity('seattle','new-york') ;
```

It is assumed in the first and third examples that `capacity(i,j)` is a parameter that was previously declared and assigned values. These statements must appear after the variable declaration and before the `Solve` statement. All the mathematical expressions available for direct assignments are usable on the right-hand side.

In nonlinear programming it is very important for the modeler to help the solver by specifying as narrow a range as possible between lower and upper bound. It is also very helpful to specify an initial solution from which the solver can start searching for the optimum. For example, in a constrained inventory model, the variables are `quantity(i)`, and it is known that the optimal solution to the unconstrained version of the problem is a parameter called `eoq(i)`. As a guess for the optimum of the constrained problem we enter

```
quantity.l(i)  =  0.5*eoq(i)  ;
```

(The default initial level is zero unless zero is not within the bounded range, in which case it is the bound closest to zero.)

It is important to understand that the `.lo` and `.up` fields are entirely under the control of the GAMS user. The `.l` and `.m` fields, in contrast, can be initialized by the user but are then controlled by the solver.

4.11.10.2 Transformation and Display of Optimal Values

(This section may be skipped on first reading if desired.)

After the optimizer is called via the `solve` statement, the values it computes for the primal and dual variables are placed in the database in the `.l` and `.m` fields. We can then read these results and transform and display them with GAMS statements.

For example, in the transportation problem, suppose we wish to know the percentage of each market's demand that is filled by each plant. After the solve statement, we would enter

```
parameter pctx(i,j)  perc of market j's demand filled by plant i;
pctx(i,j) =  100.0*x.l(i,j)/b(j) ;
display pctx ;
```

Appending these commands to the original transportation problem input results in the following output:

```
pctx    percent of market j's demand filled by plant I
          new-york   chicago   topeka
seattle     15.385   100.000
san-diego   84.615             100.000
```

For an example involving marginal, we briefly consider the *ratio constraints* that commonly appear in blending and refining problems. These linear programming models are concerned with determining the optimal amount of each of several available raw materials to put into each of several desired finished products. Let `y(i,j)` be the variable for the number of tons of raw material i put into finished product `j`. Suppose the *ratio constraint* is that no product can consist of more than 25 percent of one ingredient, that is,

```
y(i,j)/q(j) =l=   .25 ;
```

for all `i`, `j`. To keep the model linear, the constraint is written as

```
ratio(i,j).. y(i,j) - .25*q(j)  =l=  0.0  ;
```

rather than explicitly as a ratio.

The problem here is that `ratio.m(i,j)`, the marginal value associated with the linear form of the constraint, has no intrinsic meaning. At optimality, it tells us by at most how much we can benefit from relaxing the linear constraint to

```
y(i,j) - .25*q(j)  =l=  1.0 ;
```

Unfortunately, this relaxed constraint has no realistic significance. The constraint we are interested in relaxing (or tightening) is the nonlinear form of the ration constraint. For example, we would like to know the marginal benefit arising from changing the ratio constraint to

```
y(i,j)/q(j)  =l=  .26  ;
```

We can in fact obtain the desired marginals by entering the following transformation on the undesired marginals:

```
parameter  amr(i,j)  appropriate marginal for ratio constraint ;
amr(i,j)  =  ratio.m(i,j)*0.01*q.l(j) ;
display amr ;
```

Notice that the assignment statement for `amr` accesses both `.m` and `.l` records from the database. The idea behind the transformation is to notice that

```
y(i,j)/q(j)  =l=  .26  ;
```

is equivalent to

```
y(i,j) - .25*q(j)  =l=  0.01*q(j)  ;
```

4.11.11 GAMS Output

The default output of a GAMS run is extensive and informative. For a complete discussion, see chapter GAMS Output. This tutorial discusses output partially as follows:

- Echo Print
- Reference Maps
- Status Reports
- Error Messages
- Model Statistics
- Solution Reports

A great deal of unnecessary anxiety has been caused by textbooks and users' manuals that give the reader the false impression that flawless use of advanced software should be easy for anyone with a positive pulse rate. GAMS is designed with the understanding that even the most experienced users will make errors. GAMS attempts to catch the errors as soon as possible and to minimize their consequences.

4.11.11.1 Echo Prints

Whether or not errors prevent your optimization problem from being solved, the first section of output from a GAMS run is an echo, or copy, of your input file. For the sake of future reference, GAMS puts line numbers on the left-hand side of the echo. For our transportation example, which luckily contained no errors, the echo print is as follows:

```
 2  Sets
 3       i   canning plants   / seattle, san-diego /
 4       j   markets          / new-york, chicago, topeka / ;
 5
 6  Parameters
 7
 8       a(i)  capacity of plant i in cases
 9         /    seattle     350
10              san-diego   600  /
11
12       b(j)  demand at market j in cases
13         /    new-york    325
14              chicago     300
15              topeka      275  / ;
16
17  Table d(i,j)  distance in thousands of miles
18                     new-york       chicago      topeka
19      seattle           2.5           1.7          1.8
20      san-diego         2.5           1.8          1.4  ;
21
22  Scalar f  freight in dollars per case per thousand miles  /90/ ;
23
24  Parameter c(i,j)  transport cost in thousands of dollars per case ;
25
26            c(i,j) = f * d(i,j) / 1000 ;
27
28  Variables
```

```
29       x(i,j)  shipment quantities in cases
30       z       total transportation costs in thousands of dollars ;
31
32  Positive Variable x ;
33
34  Equations
35       cost        define objective function
36       supply(i)   observe supply limit at plant i
37       demand(j)   satisfy demand at market j ;
38
39  cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
40
41  supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
42
43  demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;
44
45  Model transport /all/ ;
46
47  Solve transport using lp minimizing z ;
48
49  Display x.l, x.m ;
50
```

The reason this echo print starts with line number 2 rather than line number 1 is because the input file contains a *dollar-print-control* statement. This type of instruction controls the output printing, but since it has nothing to do with defining the optimization model, it is omitted from the echo. The dollar print controls must start in column 1.

```
$title a transportation model
```

The `$title` statement causes the subsequent text to be printed at the top of each page of output. Other available instructions are given in chapter Dollar Control Options.

4.11.11.2 Error Messages

When the GAMS compiler encounters an error in the input file, it inserts a coded error message inside the echo print on the line immediately following the scene of the offense. These messages always start with **** and contain a '$' directly below the point at which the compiler thinks the error occurred. The $ is followed by a numerical error code, which is explained after the echo print. Several examples follow.

Example 1 : Entering the statement

```
set  q  quarterly time periods  / spring, sos1, fall, wtr / ;
```

results in the echo

```
      1  set  q  quarterly time periods  / spring, sos1, fall, wtr / ;
****                                                 $160
```

In this case, the GAMS compiler indicates that something is wrong with the set element sum. At the bottom of the echo print, we see the interpretation of error code 160:

```
Error Message
160  Unique element expected....
```

The problem is that `sos1` is a reserved word, which can not be used as identifiers in general. The complete list of reserved words is shown in this chapter. Thus our set element must have a unique name like `'summer'`. This is a common beginner's error.

Example 2 : Another common error is the omission of a semicolon preceding a direct assignment or equation definition. In our transportation example, suppose we omit the semicolon prior to the assignment of `c(i,j)`, as follows.

```
Parameter c(i,j)  transport cost in 1000s of dollars per case
c(i,j)  = f * d(i,j)  /  1000  ;
```

Here is the resulting output.

```
  16   Parameter c(i,j) transport cost in 1000s of dollars per case
  17             c(i,j)  = f * d(i,j)/1000
****                     $97       $195,96,409
Error Message
 96  Blank needed between identifier and text
       (-or- illegal character in identifier)
       (-or- check for missing ';' on previous line)
 97  Explanatory text can not start with '$', '=', or '..'
       (-or- check for missing ';' on previous line)
195  Symbol redefined with a different type
409  Unrecognizable item - skip to find a new statement
       looking for a ';' or a key word to get started again
```

It is not uncommon for one little offense like our missing semicolon to generate five intimidating error messages. The lesson here is: concentrate on fixing the first error and ignore the other! The first error detected (in line 17), code 97, indicate that GAMS thinks the symbols in line 17 are a continuation of the documentary text at the end of line 16 rather than a direct assignment as we intended. The error message also appropriately advises us to check the preceding line for a missing semicolon.

Unfortunately, you cannot always expect error messages to be so accurate in their advice. The compiler cannot read your mind. It will at times fail to comprehend your intentions, so learn to detect the causes of errors by picking up the clues that abound in the GAMS output. For example, the missing semicolon could have been detected by looking up the c entry in the cross-reference list (to be explained in the next section) and noticing that it was never assigned.

```
SYMBOL      TYPE      REFERENCES
c           PARAM     declared     15   ref   17
```

Example 3 : Many errors are caused merely by spelling mistakes and are caught before they can be damaging. For example, with `'Seattle'` spelled in the table differently from the way it was introduced in the set declaration, we get the following error message.

```
   4   sets
   5          i   canning plants  /seattle, san-diego /
   6          j   markets  /new-york, chicago, topeka /  ;
   7
   8   table d(i,j)  distance in thousand of miles
   9            new-york   chicago   topeka
  10      seattle       2.5      1.7        1.8
****         $170
  11      san-diego    2.5      1.8            1.4   ;
Error Message
170  Domain violation for element
```

Example 4 : Similarly, if we mistakenly enter `dem(j)` instead of `b(j)` as the right-hand side of the demand constraint, the result is

```
   45   demand(j)  ..   sum(i,  x(i,j)  )   =g=  dem(j)  ;
****                                          $140
    Error Message
    140  Unknown symbol
```

Example 5 : The next example is a mathematical error, which is sometimes committed by novice modelers and which GAMS is adept at catching. The following is mathematically inconsistent and, hence, is not an interpretable statement.

$$\text{For all } i, \quad \sum_i x_{ij} = 100$$

There are two errors in this equation, both having to do with the co ntrol of indices. Index i is over-controlled and index j is under-controlled.

You should see that index i is getting conflicting orders. By appearing in the quantifier '*for all i*', it is supposed to remain fixed for each instance of the equation. Yet, by appearing as an index of summation, it is supposed to vary. It can't do both. On the other hand, index j is not controlled in any way, so we have no way of knowing which of its possible values to use.

If we enter this meaningless equation into GAMS, both errors are correctly diagnosed.

```
   meaninglss(i)  ..    sum(i, x(i,j))   =e=    100 ;
****                        $125   $149
Error Messages
125  Set is under control already         [This refers to set i.]
149  Uncontrolled set entered as constant [This refers to set j.]
```

More information about error reporting is given in section Error Reporting. Comprehensive error detection and well-designed error messages are a big help in getting models implemented quickly and correctly.

4.11.11.3 Reference Maps

The next section of output, which is the last if errors have been detected, is a pair of *reference maps* that contain summaries and analyses of the input file for the purposes of debugging and documentation.

The first reference map is a *cross-reference map* such as one finds in most modern compilers. It is an alphabetical, cross-referenced list of all the entities (sets, parameters, variables, and equations) of the model. The list shows the type of each entity and a coded reference for each appearance of the entity in the input. The cross-reference map for our transportation example is as follows (we do not display all tables). To turn the *cross-reference map* on, please add the line

```
$onSymXRef
```

at the beginning of your model right after the `$title` statement.

```
SYMBOL      TYPE   REFERENCES

a            PARAM declared        9  defined       10      ref       42
b            PARAM declared       13  defined       14      ref       44
c            PARAM declared       25 assigned       27      ref       40
cost         EQU   declared       36  defined       40 impl-asn       48      ref       46
d            PARAM declared       18  defined       18      ref       27
demand       EQU   declared       38  defined       44 impl-asn       48      ref       46
f            PARAM declared       23  defined       23      ref       27
i            SET   declared        4  defined        4      ref        9       18       25       27
j            SET   declared        5  defined        5      ref       13       18       25       27
supply       EQU   declared       37  defined       42 impl-asn       48      ref       46
transport    MODEL declared       46  defined       46 impl-asn       48      ref       48
x            VAR   declared       30 impl-asn       48      ref       33       40       42       44
z            VAR   declared       31 impl-asn       48      ref       40       48
```

For example, the cross-reference list tells us that the symbol `a` is a parameter that was declared in line 9, defined (assigned value) in line 10, and referenced in line 42. The symbol `i` has a more complicated entry in the cross-reference list. It is shown to be a set that was declared and defined in line 4. It is referenced once in lines 9, 18, 25, 27, 30, 37, 44 and referenced twice in lines 40 and 44. Set `i` is also used as a controlling index in a summation, equation definition or direct parameter assignment in lines 27, 40, 42 and 44.

For the GAMS novice, the detailed analysis of the cross-reference list may not be important. Perhaps the most likely benefit he or she will get from the reference maps will be the discovery of an unwanted entity that mistakenly entered the model owing to a punctuation or syntax error.

The second part of the reference map is a list of model entities grouped by type and listed with their associated documentary text. For example, this list is as follows.

```
SETS

i             canning plants
j             markets

PARAMETERS

a             capacity of plant i in cases
b             demand at market j in cases
c             transport cost in thousands of dollars per case
d             distance in thousands of miles
f             freight in dollars per case per thousand miles

VARIABLES

x             shipment quantities in cases
z             total transportation costs in thousands of dollars

EQUATIONS

cost          define objective function
demand        satisfy demand at market j
supply        observe supply limit at plant i

MODELS

transport
```

4.11.11.4 Equation Listings

Once you succeed in building an input file devoid of compilation errors, GAMS is able to generate a model. The question remains, and only you can answer it, does GAMS generate the model you intended?

The equation listing is probably the best device for studying this extremely important question. A product of the solve command, the equation listing shows the specific instance of the model that is created when the current values of the sets and parameters are plugged into the general algebraic form of the model. For example, the generic demand constraint given in the input file for the transportation model is

```
demand(j)  ..   sum(i, x(i,j))  =g=  b(j)  ;
```

while the equation listing of specific constraints is

```
--------demand   =g=   satisfy demand at market  j
demand(new-york)..  x(seattle, new-york) +x(san-diego, new-york)  =g=  325  ;
demand(chicago)..  x(seattle, chicago) +x(san-diego, chicago )  =g=  300  ;
demand(topeka)..  x(seattle, topeka) +x(san-diego, topeka)  =g=  275 ;
```

The default output is a maximum of three specific equations for each generic equation. To change the default, insert an input statement prior to the solve statement:

```
option limrow  =  r ;
```

where `r` is the desired number.

The default output also contains a section called the column listing, analogous to the equation listing, which shows the coefficients of three specific variables for each generic variable. This listing would be particularly useful for verifying a GAMS model that was previously implemented in MPS format. To change the default number of specific column printouts per generic variable, the above command can be extended:

```
option limrow  =  r,  limcol  = c ;
```

where `c` is the desired number of columns. (Setting `limrow` and `limcol` to 0 is a good way to reduce the size of your lst file after your model has been debugged.) In nonlinear models, the GAMS equation listing shows first-order Taylor approximations of the nonlinear equations. The approximations are taken at the starting values of the variables.

4.11.11.5 Model Statistics

The last section of output that GAMS produces before invoking the solver is a group of statistics about the model's size, as shown below for the transportation example.

```
MODEL STATISTICS

BLOCKS OF EQUATIONS       3     SINGLE EQUATIONS        6
BLOCKS OF VARIABLES       2     SINGLE VARIABLES        7
NON ZERO ELEMENTS        19
```

The `BLOCK` counts refer to the number of generic equations and variables. The `SINGLE` counts refer to individual rows and columns in the specific model instance being generated. For nonlinear models, some other statistics are given to describe the degree of non-linearity in the problem.

4.11.11.6 Status Reports

After the solver executes, GAMS prints out a brief *Solve Summary* whose two most important entries are `SOLVER STATUS` and the `MODEL STATUS`. For our transportation problem the solve summary is as follows:

```
               S O L V E      S U M M A R Y

     MODEL   TRANSPORT           OBJECTIVE  z
     TYPE    LP                  DIRECTION  MINIMIZE
     SOLVER  CPLEX               FROM LINE  49

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      1 Optimal
**** OBJECTIVE VALUE               153.6750

 RESOURCE USAGE, LIMIT          0.031      1000.000
 ITERATION COUNT, LIMIT         4    2000000000
```

The status reports are preceded by the same **** string as an error message, so you should probably develop the habit of searching for all occurrences of this string whenever you look at an output file for the first time. The desired solver status is `1 NORMAL COMPLETION`, but there are other possibilities, documented in section The Solve Summary, which relate to various types of errors and mishaps.

There are eleven possible model statuses, including the usual linear programming termination states (`1 Optimal, 3 Unbounded, 4 Infeasible`), and others relating to nonlinear and integer programming. In nonlinear programming, the status to look for is `2 Locally Optimal`. The most the software can guarantee for nonlinear programming is a local optimum. The user is responsible for analyzing the convexity of the problem to determine whether local optimality is sufficient for global optimality. In integer programming, the status to look for is `8 Integer Solution`. This means that a feasible integer solution has been found. More detail follows as to whether the solution meets the relative and absolute optimality tolerances that the user specifies.

4.11.11.7 Solution Reports

If the solver status and model status are acceptable, then you will be interested in examining the results of the optimization. The results are first presented in as standard mathematical programming output format, with the added feature that rows and columns are grouped and labeled according to names that are appropriate for the specific model just solved. In this format, there is a line of printout for each row and column giving the lower limit, level, upper limit, and marginal. Generic equation block and the column output group the row output by generic variable block. Set element names are embedded in the output for easy reading. In the transportation example, the solver outputs for `supply(i)`, `demand(j)`, and `x(i,j)` are as follows:

```
---- EQU supply      observe supply limit at plant i

             LOWER     LEVEL     UPPER    MARGINAL

seattle       -INF    350.000   350.000      EPS
san-diego     -INF    550.000   600.000      .

---- EQU demand      satisfy demand at market j

             LOWER     LEVEL     UPPER    MARGINAL
```

```
new-york   325.000   325.000     +INF      0.225
chicago    300.000   300.000     +INF      0.153
topeka     275.000   275.000     +INF      0.126

---- VAR x           shipment quantities in cases

                       LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york       .       50.000     +INF       .
seattle  .chicago        .      300.000     +INF       .
seattle  .topeka         .         .        +INF      0.036
san-diego.new-york       .      275.000     +INF       .
san-diego.chicago        .         .        +INF      0.009
san-diego.topeka         .      275.000     +INF       .
```

The single dots '.' in the output represent zeroes. The entry EPS, which stands for *epsilon*, mean very small but nonzero. In this case, EPS indicates degeneracy. (The slack variable for the Seattle supply constraint is in the basis at zero level. The marginal is marked with EPS rather than zero to facilitate restarting the optimizer from the old basis.)

If the solvers results contain either infeasibilities or marginal costs of the wrong sign, then the offending entries are marked with INFES or NOPT, respectively. If the problem terminates unbounded, then the rows and columns corresponding to extreme rays are marked UNBND.

At the end of the solvers solution report is a very important *report summary*, which gives a tally of the total number of non-optimal, infeasible, and unbounded rows and columns. For our example, the report summary shows all zero tallies as desired.

```
**** REPORT SUMMARY :        0     NONOPT
                             0 INFEASIBLE
                             0  UNBOUNDED
```

After the solver's report is written, control is returned from the solver back to GAMS. All the levels and marginals obtained by the solver are entered into the GAMS database in the .l and .m fields. These values can then be transformed and displayed in any desired report. As noted earlier, the user merely lists the quantities to be displayed, and GAMS automatically formats and labels an appropriate array. For example, the input statement.

```
display x.l, x.m  ;
```

results in the following output.

```
----     50 VARIABLE  x.L           shipment quantities in cases

             new-york     chicago      topeka

seattle        50.000     300.000
san-diego     275.000                 275.000

----     50 VARIABLE  x.M           shipment quantities in cases

              chicago      topeka

seattle                     0.036
san-diego       0.009
```

As seen in reference maps, equation listings, solution reports, and optional displays, GAMS saves the documentary text and '*parrots*' it back throughout the output to help keep the model well documented.

4.11.12 Summary

This tutorial has demonstrated several of the design features of GAMS that enable you to build practical optimization models quickly and effectively. The following discussion summarizes the advantages of using an algebraic modeling language such as GAMS versus a matrix generator or conversational solver.

- By using an algebra-based notation, you can describe an optimization model to a computer nearly as easily as you can describe it to another mathematically trained person.
- Because an algebraic description of a problem has generality, most of the statements in a GAMS model are reusable when new instances of the same or related problems arise. This is especially important in environments where models are constantly changing.
- You save time and reduce generation errors by creating whole sets of closely related constraints in one statement.
- You can save time and reduce input errors by providing formulae for calculating the data rather than entering them explicitly.
- The model is self-documenting. Since the tasks of model development and model documentation can be done simultaneously, the modeler is much more likely to be conscientious about keeping the documentation accurate and up to date.
- The output of GAMS is easy to read and use. The solution report from the solver is automatically reformatted so that related equations and variables are grouped together and appropriately labeled. Also, the `display` command allows you to modify and tabulate results very easily.
- If you are teaching or learning modeling, you can benefit from the insistence of the GAMS compiler that every equation be mathematically consistent. Even if you are an experienced modeler, the hundreds of ways in which errors are detected should greatly reduce development time.
- By using the *dollar* operator and other advanced features not covered in this tutorial, one can efficiently implement large-scale models. Specific applications of the dollar operator include:

  1. It can enforce logical restrictions on the allowable combinations of indices for the variables and equations to be included in the model. You can thereby screen out unnecessary rows and columns and keep the size of the problem within the range of solvability.
  2. It can be used to build complex summations and products, which can then be used in equations or customized reports.
  3. It can be used for issuing warning messages or for terminating prematurely conditioned upon context-specific data edits.

4.12 Quick Start Tutorial

GAMS (the General Algebraic Modeling System) is a high-level modeling system for mathematical programming problems. This tutorial is aimed at modelers who are new to GAMS and are looking for a quick introduction to the core features of GAMS. Note that the text contains many links to the GAMS User's Guide for further exploration. Observe that this tutorial is adapted from the Quick Start Tutorial from Bruce McCarl. Another introductory text is the tutorial A GAMS Tutorial by Richard E. Rosenthal.

4.12.1 Three Basic Models

Most modelers who are interested in GAMS have one (or more) of the following backgrounds:

- They wish to solve constrained optimization problems with objective functions.
- They are looking for a system to solve general equilibrium problems that are arise in various areas of economics.
- They wish to solve nonlinear systems of equations that arise in engineering.

GAMS is well equipped to model and solve all three types of problems. We will start by introducing examples for all three areas and the discussions that will follow will refer to these three basic examples.

4.12.1.1 First Example: Solving a Linear Programming Problem (LP)

The simplest constrained optimization problem is an LP. Assume we wish to solve the following LP that represents a simple farm model where profit is maximized:

$$
\begin{array}{llrcrcrcll}
\max & & 109X_{corn} & + & 90X_{wheat} & + & 115X_{cotton} & & & \\
s.t. & & X_{corn} & + & X_{wheat} & + & X_{cotton} & \leq & 100 & \text{(land)} \\
 & & 6X_{corn} & + & 4X_{wheat} & + & 8X_{cotton} & \leq & 500 & \text{(labor)} \\
 & & & & & & X_{corn} & \geq & 0 & \text{(nonnegativity)} \\
 & & & & & & X_{wheat} & \geq & 0 & \text{(nonnegativity)} \\
 & & & & & & X_{cotton} & \geq & 0 & \text{(nonnegativity)}
\end{array}
$$

Note that there are three decision variables: X_{corn} denotes the land assigned to growing corn, X_{wheat} is the land assigned to growing wheat and X_{cotton} represents the land where cotton is grown. In the first line the total profit is expressed as a function of the land allotted to the three crops, the multipliers represent the expected profit per acre depending of the type of crop. The first inequality imposes a limit on the available land, the second inequality imposes a limit on the available labor and the final three lines restrict the decision variables to nonnegative values.

This problem may be expressed in GAMS as follows:

```
Positive Variables    Xcorn, Xwheat, Xcotton;
Variables             Z;

Equations     obj, land, labor;

obj..  Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
land..             Xcorn +      Xwheat +       Xcotton =l= 100;
labor..        6 * Xcorn +  4 * Xwheat +   8 * Xcotton =l= 500;

Model farmproblem / obj, land, labor /;

solve farmproblem using LP maximizing Z;
```

We will analyze this formulation and discuss every part of it after introducing the other two basic examples.

4.12.1.2 Second Example: Solving for an Economic Equilibrium

The simplest general equilibrium model is the single good, single market problem. Assume we wish to solve the following equilibrium problem:

$$\begin{array}{lllll} \text{Demand Price:} & P & \geq & P_d & = & 6 - 0.3Q_d \\ \text{Supply Price:} & P & \leq & P_s & = & 1 + 0.2Q_s \\ \text{Quantity Equilibrium:} & Q_s & \geq & Q_d & & \\ \text{Nonnegativity:} & P & \geq & 0 & & \\ & Q_s & \geq & 0 & & \\ & Q_d & \geq & 0 & & \end{array}$$

Here P is the market clearing price, P_d is the demand curve, Q_d denotes the quantity demanded, P_s the supply curve and Q_s the quantity supplied. Note that this is a problem in 3 equations and 3 variables. The variables are P, Q_d, and Q_s. Note that P_d and P_s are not variables, since they can be computed from the equality relations.

Usually all equality constraints are used for such a set up. However, we will use a more general setup, because it relaxes some assumptions and more accurately depicts a model ready to be implemented in GAMS. In particular, we permit the case where the supply curve price intercept may be above the demand curve price intercept and thus the market may clear with a nonzero price but a zero quantity. In addition, we allow the market price to be above the demand curve price and below the supply curve price. We also impose some additional conditions based on Walras' Law to ensure a proper solution in such cases.

$$\begin{array}{lll} Q_d(P - P_d) = 0 & \text{or} & Q_d(P_d - (6 - 0.3Q_d)) = 0 \\ Q_s(P - P_s) = 0 & \text{or} & Q_s(P_s - (1 + 0.2Q_s)) = 0 \\ P(Q_s - Q_d) = 0 & & \end{array}$$

Here the quantity demanded is nonzero only if the market clearing price equals the demand curve price, the quantity supplied is nonzero only if the market clearing price equals the supply curve price and the market clearing price is only nonzero if $Q_s = Q_d$.

The simplest GAMS formulation is given below. Note that we needed to rearrange the equation `Psupply` in order to achieve a greater than inequality, thus we accomodated the requirements of the solver PATH.

```
Positive Variables  P, Qd , Qs;

Equations     Pdemand, Psupply, Equilibrium;
Pdemand..     P            =g= 6 - 0.3*Qd;
Psupply..    ( 1 + 0.2*Qs) =g= P;
Equilibrium.. Qs           =g= Qd;

Model problem / Pdemand.Qd, Psupply.Qs, Equilibrium.P /;

solve problem using MCP;
```

We will analyze this formulation and discuss every component of it after introducing the third example.

4.12.1.3 Third Example: Solving a Nonlinear Equation System

Engineers often wish to solve a nonlinear system of equations. Examples include chemical equilibria or problems in the context of oil refining. Many other such problems of this type exist. The problem that follows is adapted from the paper Wall, T W, Greening, D, and Woolsey , R E D, "Solving Complex Chemical Equilibria Using a Geometric-Programming Based Technique" Operations Research 34, 3 (1987).

```
ba * so4 = 1

baoh / ba / oh = 4.8

hso4 / so4 / h = 0 .98

h * oh = 1

ba + 1e-7*baoh = so4 + 1e-5*hso4

2 * ba + 1e-7*baoh + 1e-2*h = 2 * so4 + 1e-5*hso4 + 1e-2*oh
```

Note that this is a nonlinear system of equations with the variables `ba`, `so4`, `baoh`, `oh`, `hso4` and `h`. The following formulation in GAMS is from the model [**WALL**] in the GAMS model library.

```
Variables ba, so4, baoh, oh, hso4, h ;

Equations r1, r2, r3, r4, b1, b2 ;

r1..  ba * so4 =e= 1 ;
r2..  baoh / ba / oh =e= 4.8 ;
r3..  hso4 / so4 / h =e= .98 ;
r4..  h * oh =e= 1 ;
b1..  ba + 1e-7*baoh =e= so4 + 1e-5*hso4 ;
b2..  2 * ba + 1e-7*baoh + 1e-2*h =e= 2 * so4 + 1e-5*hso4 + 1e-2*oh ;

Model wall / all / ;

ba.l=1; so4.l=1; baoh.l=1; oh.l=1; hso4.l=1; h.l=1;

solve wall using nlp minimizing ba;
```

4.12.2 Components of the Example Models

Now that we have our three basic models in place, we will analyze them and identify and discuss their components, including variables, equations, model definitions, solve statements and starting points.

4.12.2.1 Variables in the Example Models

Variables in GAMS have to be declared as variables with a variable statement. The variable statements in the examples above are repeated below:

- First example:

```
Positive Variables    Xcorn, Xwheat, Xcotton;
Variables             Z;
```

- Second example:

```
Positive Variables     P, Qd , Qs;
```

- Third example:

```
Variables ba, so4, baoh, oh, hso4, h ;
```

Note that `variable(s)` is a keyword in GAMS. `Positive` is another keyword, it serves as a modifier to the keyword `variable` and has the effect that the variables that follow may take only nonnegative values. Variables that are declared with the keyword `variable` without a modifier are unresticted in sign. For more details on variable declarations and variable types, see sections Variable Declarations and Variable Types respectively.

Observe that GAMS is not case sensitive and allows a line feed to be used instead of a comma. Thus, the following three variable declarations are all valid and have the same effect:

```
POSITIVE VARIABLES  Xcorn, Xwheat, Xcotton;
Positive Variables  xcorn,
                    xwheat,
                    xcotton;
positive variables  Xcorn
                    Xwheat , Xcotton;
```

In the GAMS formulation of the first example, we have introduced the variable `Z` in addition to the three variables that featured in the mathematical formulation. Note that GAMS requires the optimization model

```
Maximize cx
```

to have the following form:

```
Maximize z
         z = cx
```

Here `z` is a variable, also called *objective variable*. Observe that it is the objective variable that is maximized, not the function `cx`. The name of the objective variable may be chosen freely by the user, like any other variable name. The objective variable is a free variable, which means that it has no bounds. Hence, it each optimization problem there must always be at least one free variable.

Given the requirement of an objective variable in an optimization problem, we need to declare a new free variable and introduce an equation for it. In our first example we declared `Z` as a free variable, then we declared and specified the equation `obj` setting `Z` equal to the objective function expression and we instructed the solver to maximize `Z`. The relevant lines of code from the first example follow:

```
Variables     Z;

Equation     obj,  land ,  labor;

obj..   Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;

solve farmproblem using LP maximizing Z;
```

Note

> In some optimization problems users do not need to introduce a new variable to serve as an objective variable, but a free variable that already features in the model may be used. For example, in the second example above, `ba` is a free variable that is used as objective variable, since the model type NLP requires an objective variable. We will discuss this further below.

4.12.2.2 Equations in the Example Models

Each equation in a model must first be *declared* with a equation declaration statement and then *defined* with a equation definition statement. We repeat the respective statement from the three examples below:

- First example:

```
Equations     obj, land, labor;

obj..  Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
land..             Xcorn +      Xwheat +       Xcotton =l= 100;
labor..        6 * Xcorn +  4 * Xwheat +   8 * Xcotton =l= 500;
```

- Second example:

```
Equations     Pdemand, Psupply, Equilibrium;

Pdemand..     P            =g= 6 - 0.3*Qd;
Psupply..    ( 1 + 0.2*Qs) =g= P;
Equilibrium.. Qs           =g= Qd;
```

- Third example:

```
Equations r1, r2, r3, r4, b1, b2 ;

r1..  ba * so4 =e= 1 ;
r2..  baoh / ba / oh =e= 4.8 ;
r3..  hso4 / so4 / h =e= .98 ;
r4..  h * oh =e= 1 ;
b1..  ba + 1e-7*baoh =e= so4 + 1e-5*hso4 ;
b2..  2 * ba + 1e-7*baoh + 1e-2*h =e= 2 * so4 + 1e-5*hso4 + 1e-2*oh ;
```

Note that the equation declaration statement begins with the keyword `Equation(s)`. Its main purpose is to name the equation(s). For further details on equation declarations, see section Declaring Equations.

The algebraic structure of the equation is specified in the equation definition statement, where the name of the equation is followed by two dots `..` and the algebra. The relation between the left-hand side and right-hand side is specified by special symbols indicating the equation type. The most common symbols are `=e=` for equality, `=l=` for a less than or equal to relation and `=g=` for a greater than or equal to relation. An overview of all GAMS equation types is given in Table Equation Types. For more information on equation definitions, see sections Defining Equations and Conditional Equations.

4.12.2.3 Model Definitions in the Example Models

Once all structural elements of a model have been defined and specified, a model statement is used to define the model that will be solved. The model statement form the three examples follow.

- First example:

```
Model farmproblem / obj, land, labor /;
```

- Second example:

```
Model problem / Pdemand.Qd,  Psupply.Qs, Equilibrium.P /;
```

- Third example:

```
Model wall / all / ;
```

Note that a model statement always starts with the keyword `model`. It is follwed by the name of the model which users may choose freely and a list of the equations that are part of the model. Observe that the list of equations in the first example contains all equations that were declared and defined. However, this need not be the case. It is possible to declare and define an equation and then not use it in a model. For example, users could define several models that contain different subsets of equations. If all equations that were previously declared and defined enter the model, the keyword `all` may be used as a shortcut, like in the third example.

Attention

Only equations that have been declared may be listed in a model statement. All equations in a model must have been defined before the model can be solved.

Note that equilibrium problems like the second example are solved as Mixed Complementarity Problems (MCPs). MCPs require a special variant of the model statement where not only the equations are listed , but also their complementarity relationships. The complementarity relationship between an equation and its associated variable is marked with a dot ".". Hence, in the second example, the equation `Pdemand` is perpendicular to the the variable`Qd`, the equation `Psupply` is perpendicular to the variable `Qs` and the equation `Equilibrium` is perpendicular to the variable `P`.

In the three model statements above the name of the model was immediately followed by the list of the equations included in the model. Observe that an optional explanatory text may be inserted after the name of the model.

4.12.2.4 Solve Statements in the Example Models

After the model has been defined it remains to be solved, i.e. to find a solution for the variables. The solve statement directs GAMS to use a solver to optimize the model or solve the system of equations. The solve statements of the three examples follow:

- First example:

```
solve farmproblem using LP maximizing Z;
```

- Second example:

```
solve problem using MCP;
```

- Third example:

```
solve wall using nlp minimizing ba;
```

Note that a solve statement always begins with the keyword `solve` follwed by the name of the model, as previously specified with a model statement, the keyword `using` and the *type* of the model. We have an LP (linear programming problem) in the first example, an MCP (mixed compementarity problem) in the second example and an NLP (nonlinear programming program) in the third example. A complete list of GAMS model types is given in section Classification of Models. After the model type the direction of the optimization is given: either `maximizing` as in the first example or`minimizing` as in the third example. In the final position is the name of the objective variablethat is to be optimized.

Note

MCPs and systems of equations of the model type Constrained Nonlinear System (CNS) do not have objective variables. Therefore their solve statements end with the model type.

Observe that the system of nonlinear equations in the second example was expressed as an NLP that requires an objective function and a related objective variable. Actually this is an older practice in GAMS as the GAMS model type CNS was added after this example was initially formulated. Hence, we could alternatively use the following solve statement:

```
solve wall using cns;
```

However, note that the model type CNS can only be solved by select solvers and cannot incorporate integer variables. A formulation as an optimization problem relaxes these restrictions, thus allowing the use of the model type MINLP and all NLP solvers. If the model type NLP or one of its variants is chosen, one of the variables must be selected as objective variable to be optimzed. Observe that which variable is chosen may not really have any effect since a feasible solution requires all of the simultaneous equations to be solved. Thus, while in the third example the variable `ba` is maximized, there is no inherent interest in attaining its maximum - it is just convenient.

4.12.2.5 Starting Points

In the third example we have the following line *before* the solve statement:

```
ba.l=1; so4.l=1; baoh.l=1; oh.l=1; hso4.l=1; h.l=1;
```

Note that this line provides starting points for the variables in the model. The suffix `.l` appended to a variable name denotes the level value of that variable. If a level value of a variable is specified before the solve statement, this value will act as a starting point for the search algorithms. The level value of a variable is one example of a *variable attribute*. Other examples include lower and upper bounds which may be set with the suffixes `.lo` and `.up`. For more information on variable attributes, see section Variable Attributes. For guidance on how to choose good starting points, see the tutorial Good NLP Formulations. Specifying starting points may be important for avoiding numerical problems in the model solution. For more on this topic, see the tutorial Finding and Fixing Execution Errors and Performance Problems.

Observe that the statetements above are the first examples of assignment statements in this tutorial. Assignment statements play a crucial role in GAMS, they are introduced in section The Assignment Statement.

4.12.3 Running a GAMS Job

The model formulation of a GAMS model is saved in a text file with the extension `.gms`, say `myfile.gms`. Then the file is submitted to GAMS. GAMS will execute the instructions in the `.gms` file, with the result that calculations are done, solvers are used and an output file with the solution results is created. The output file is also called the *listing file*. By default, the name listing file of the input file `myfile.gms` will be `myfile.lst`.

There are two ways to submit a job to GAMS: via the command line and via the IDE GAMS Studio.

4.12.3.1 Running GAMS on the Command Line

The `myfile.gms` file may be run with GAMS using the following call:

```
> gams myfile
```

Note the extension `.gms` may be omitted. This basic GAMS call may be extended with arguments that are called *command line parameters*. The following example serves as illustration:

```
> gams trnsport pw=80 ps=9999 s=mysave
```

Note that there are three command line parameters: the first will set the page width to 80, the second will set the page length to 9999 and the third will have the effect that a work file named `mysave` is saved. GAMS offers many command line parameters, they are introduced and discussed in detail in chapter The GAMS Call and Command Line Parameters.

4.12.3.2 Running GAMS with GAMS Studio

GAMS Studio is an integrated development environment for Windows, macOS, and Linux that facilitates editing, development, debugging GAMS models and running GAMS jobs.

4.12.4 Examining The Output

The output of a GAMS run is saved in the output or listing file. The listing file may contain many parts. Some parts may be suppressed by the modeler, other parts are suppressed by default and need to be activated. For an introduction and detailed discussion, see chapter GAMS Output.

In this section we will review the output that is generated by running the three example models.

4.12.4.1 The Echo Print

The listing file always begins with the echo print. The echo print is a copy of the input file with added line numbers. For example, the echo print of the first example is given below:

```
 1  Positive Variables    Xcorn, Xwheat, Xcotton;
 2  Variables             Z;
 3
 4  Equations     obj, land, labor;
 5
 6  obj..  Z =e= 109 * Xcorn + 90 * Xwheat + 115 * Xcotton;
 7  land..             Xcorn +      Xwheat +       Xcotton =l= 100;
 8  labor..         6 * Xcorn +  4 * Xwheat +   8 * Xcotton =l= 500;
 9
10  Model farmproblem / obj, land, labor /;
11
12  solve farmproblem using LP maximizing Z;
```

Note that the echo print of the other two examples follows the same principles. Note further, that the echo print serves as an important reference guide since GAMS reports the line numbers of any errors that were detected and of solve and display statements.

Observe that usually the code of even experienced GAMS modelers will contain compilation errors. They are marked in the echo print. For more information on compilation error and how to resolve them, see section Compilation Errors and the tutorial Fixing Compilation Errors.

4.12.4.2 Model Generation Output

Once GAMS has successfully compiled the input file and any numerical computations in assignments have been performed, the solve statements will be executed. The first step is generating a computer readable version of the equations in the problem that will be passed on to an appropriate solver system. During the model generation phase GAMS creates the following output:

- A listing of the equations of the model.
- A listing of the variables of the model.
- A summary of the model structure.
- If errors were detected during model generation they will be reported as well.

The Equation Listing

The equation listing is the first part of the output generated by a solve statement. By default, the first three equations in every block are listed. The equation listing of the first two equations of each of the three examples are given below:

- First example:

```
Equation Listing    SOLVE farmproblem Using LP From line 12


---- obj  =E=

obj..  - 109*Xcorn - 90*Xwheat - 115*Xcotton + Z =E= 0 ; (LHS = 0)


---- land  =L=

land..  Xcorn + Xwheat + Xcotton =L= 100 ; (LHS = 0)
```

- Second example:

```
Equation Listing    SOLVE problem Using MCP From line 10


---- Pdemand  =G=

Pdemand..  P + 0.3*Qd =G= 6 ; (LHS = 0, INFES = 6 ****)


---- Psupply  =G=

Psupply..  - P + 0.2*Qs =G= -1 ; (LHS = 0)
```

- Third example:

```
Equation Listing    SOLVE wall Using NLP From line 16


---- r1  =E=

r1..  (1)*ba + (1)*so4 =E= 1 ; (LHS = 1)


---- r2  =E=

r2..  - (1)*ba + (1)*baoh - (1)*oh =E= 4.8 ; (LHS = 1, INFES = 3.8 ****)
```

Note that the equation listing is a representation of the algebraic structure of the linear terms in the equation and a local representation containing the first derivatives of the nonlinear terms. The nonlinear terms are automatically placed in parentheses to indicate a local approximation. For further details, see section The Equation Listing in chapter GAMS Output.

The Column Listing

The column or variable listing contains a list of the individual coefficients sorted by column rather than by row (like in the equation listing). By default, the first three entries for each variable are shown, along with their lower bound `.lo`, upper bound `.up` and current level values `.l`. The column listing of the first two variables for each of the three examples are given below.

- First example:

```
Column Listing      SOLVE farmproblem Using LP From line 12


---- Xcorn

Xcorn
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
     -109       obj
        1       land
        6       labor


---- Xwheat

Xwheat
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
      -90       obj
        1       land
        4       labor
```

- Second example:

```
Column Listing      SOLVE problem Using MCP From line 10


---- P

P
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
```

```
          1       Pdemand
         -1       Psupply


---- Qd

Qd
                 (.LO, .L, .UP, .M = 0, 0, +INF, 0)
        0.3      Pdemand
       -1        Equilibrium
```

- Third example:

```
Column Listing      SOLVE wall Using NLP From line 16


---- ba

ba
                 (.LO, .L, .UP, .M = -INF, 1, +INF, 0)
       (1)       r1
      (-1)       r2
        1        b1
        2        b2


---- so4

so4
                 (.LO, .L, .UP, .M = -INF, 1, +INF, 0)
       (1)       r1
      (-1)       r3
       -1        b1
       -2        b2
```

Model Statistics

The final information generated while a model is being prepared for solution, is the statistics block. Its most obvious use is to provide details on the size and nonlinearity of the model. The model statistics of the third example follow.

```
MODEL STATISTICS

BLOCKS OF EQUATIONS           6     SINGLE EQUATIONS            6
BLOCKS OF VARIABLES           6     SINGLE VARIABLES            6
NON ZERO ELEMENTS            20     NON LINEAR N-Z             10
DERIVATIVE POOL              20     CONSTANT POOL              16
CODE LENGTH                  22
```

For more information on model statistics, see section The Model Statistics in chapter GAMS Output.

4.12.4.3 The Solution Report

The final major component of the listing file is the solution output. It consists of a summary, some solver-specific output and a report of the solutions for equations and variables.

The Solve Summary

The solve summary is very important since it contains an overview of key information of the solution. The solve summary of the third example is given below.

```
               S O L V E      S U M M A R Y

     MODEL   wall                OBJECTIVE  ba
     TYPE    NLP                 DIRECTION  MINIMIZE
     SOLVER  CONOPT              FROM LINE  16

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      2 Locally Optimal
**** OBJECTIVE VALUE                1.0000

 RESOURCE USAGE, LIMIT          0.034      1000.000
 ITERATION COUNT, LIMIT         7    2000000000
 EVALUATION ERRORS              0             0
```

For details on the solve summary including possible values for `SOLVER STATUS` and `MODEL STATUS`, see section The Solve Summary in chapter GAMS Output.

The Solver Report

The next part of the listing file is the solver report. It contains solver specific output. The respective output of the third example follows:

```
CONOPT 3         24.7.3 r58181 Released Jul 11, 2016 DEG x86 64bit/MacOS X


    C O N O P T 3   version 3.17A
    Copyright (C)   ARKI Consulting and Development A/S
                    Bagsvaerdvej 246 A
                    DK-2880 Bagsvaerd, Denmark

                  Pre-triangular equations:   0
                  Post-triangular equations:  0


 ** Optimal solution. There are no superbasic variables.
```

For more on solver reports, see section Solver Report in chapter GAMS Output.

The Solution Listing

The solution listing is a row-by-row then column-by-column listing of the solutions returned to GAMS by the solver program. Each individual equation and variable is listed including their level and marginal values and their lower and upper bounds. The solution listing of the first example follows:

```
                            LOWER          LEVEL          UPPER         MARGINAL

---- EQU obj                  .              .              .             1.0000
---- EQU land               -INF          100.0000       100.0000       52.0000
---- EQU labor              -INF          500.0000       500.0000        9.5000

                            LOWER          LEVEL          UPPER         MARGINAL

---- VAR Xcorn                .             50.0000        +INF            .
---- VAR Xwheat               .             50.0000        +INF            .
---- VAR Xcotton              .                .           +INF         -13.0000
---- VAR Z                  -INF          9950.0000        +INF            .
```

Note that single dots '.' represent zeros. The extended range arithmetic symbols `-INF` and `+INF` denote minus and plus infinity respectively. `EPS` is another extended range arithmetic symbol that often appears in solution listings. In this context, it is used to indicate basis status in the degenerate case. For example, a basic variable is indicated by a zero (single dot) marginal, while a non-basic variable whose marginal is (nearly) zero is indicated by a marginal of EPS. For further details, see section The Solution Listing in chapter GAMS Output.

4.12.5 Exploiting the Algebraic Structure

GAMS was deliberately designed to make it easy to express algebraic relationships. Thus it provides notation for indexed operations like sums. Assume that z is defined as the sum over x_i where $i = \{1, 2, 3\}$:

$$z = \sum_{i=1}^{3} x_i = x_1 + x_2 + x_3$$

This can be expressed in GAMS in the following way:

```
z = sum(i, x(i));
```

Here `i` is a set with three elements, `z` is a scalar or variable and `x(i)` is a parameter or variable defined over the set `i`. Note that the keyword `sum` automatically cycles through all elements of the set `i`.

Indexed operations like `sum` may be used in the algebraic specification of equations or in the context of assignments where the value of identifiers are computed. In the following two sections we will introduce revised versions of the first two examples, where we illustrate how using `sum` simplifies the code and enables more general formulations.

4.12.5.1 Revised First Example

Recall that the first example is the following LP:

$$\begin{array}{lrcrcrcll}
\max & 109X_{corn} & + & 90X_{wheat} & + & 115X_{corn} & & & \\
\text{s.t.} & X_{corn} & + & X_{wheat} & + & X_{cotton} & \leq & 100 & \text{(land)} \\
 & 6X_{corn} & + & 4X_{wheat} & + & 8X_{cotton} & \leq & 500 & \text{(labor)} \\
 & & & & & X_{corn} & \geq & 0 & \text{(nonnegativity)} \\
 & & & & & X_{wheat} & \geq & 0 & \text{(nonnegativity)} \\
 & & & & & X_{cotton} & \geq & 0 & \text{(nonnegativity)}
\end{array}$$

Note that this is a special case of the general resource allocation problem that can be written as follows:

$$\begin{array}{lrcll} \max & \sum_j c_j X_j & & & \\ \text{s.t.} & \sum_j a_{ij} X_j & \leq & b_i & \text{for all } i \\ & X_j & \geq & 0 & \text{for all } j, \end{array}$$

where

- $j = \{\text{corn, wheat, cotton}\}$
- $i = \{\text{land, labor}\}$
- $X_j = \{X_{\text{corn}}, X_{\text{wheat}}, X_{\text{cotton}}\}$
- $c_j = \{109, 90, 115\}$
- $a_{ij} = \begin{pmatrix} 1 & 1 & 1 \\ 6 & 4 & 8 \end{pmatrix}$
- $b_i = \{100, 500\}$.

This model may be formulated in GAMS in the following way:

```
Set       j        / corn, wheat, cotton /
          i        / land, labor /;

Parameter c(j)     / corn 109, wheat 90, cotton 115 /
          b(i)     / land 100 , labor 500 /;

Table     a(i,j)
            corn    wheat    cotton
land          1        1        1
labor         6        4        8      ;

Positive Variables x(j);
Variables          profit;

Equations          objective
                   constraint(i) ;

objective..      profit =e= sum(j, (c(j))*x(j));
constraint(i)..  sum(j, a(i,j) *x(j))  =l=   b(i);

Model resalloc /all/;
solve resalloc using LP maximizing profit;
```

We will discuss the components of this model in section Components of the Revised Example Models below.

4.12.5.2 Revised Second Example

Recall that the second example is an economic equilibrium model with a single commodity:

$$
\begin{array}{lllll}
\text{Demand Price:} & P & \geq & P_d & = & 6 - 0.3Q_d \\
\text{Supply Price:} & P & \leq & P_s & = & 1 + 0.2Q_s \\
\text{Quantity Equilibrium:} & Q_s & \geq & Q_d & & \\
\text{Nonnegativity:} & P & \geq & 0 & & \\
 & Q_s & \geq & 0 & & \\
 & Q_d & \geq & 0 & &
\end{array}
$$

A more general formulation of this model accomodates multiple commodities. Consider the following formulation where c denotes the commodities:

$$
\begin{array}{lllll}
\text{Demand Price for c:} & P_c & \geq & Pd_c & = & Id_c - \sum_{cc} Sd_{c,cc} Qd_{cc} \quad \text{for all } c \\
\text{Supply Price for c:} & P & \leq & Ps_c & = & Is_c + \sum_{cc} Ss_{c,cc} Qs_{cc} \quad \text{for all } c \\
\text{Quantity Equil. for c:} & Qs_c & \geq & Qd_c & & \quad \text{for all } c \\
\text{Nonnegativity:} & P_c & \geq & 0 & & \quad \text{for all } c \\
 & Qs_c & \geq & 0 & & \quad \text{for all } c \\
 & Qd_c & \geq & 0 & & \quad \text{for all } c,
\end{array}
$$

where

- P_c is the price of commodity c,
- Qd_c is the quantity demanded of commodity c,
- Pd_c is the price from the inverse demand curve for commodity c,
- Qs_c is the quantity supplied of commodity c,
- Ps_c is the price from the inverse supply curve for commodity c,
- cc is an alternative notation for the commodities and is equivalent to c,
- Id_c is the inverse demand curve intercept for c,
- $Sd_{c,cc}$ is the slope of the inverse demand curve. It is used to model the effect of buying one unit of commodity cc on the demand price of commodity c. When $c = cc$ it is an own commodity effect and when $c \neq cc$ it is a cross commodity effect.
- Is_c is the inverse supply curve intercept for c,
- Ssc, cc is theslope of the inverse supply curve. It is used to model the effect of supplying one unit of commodity cc on the supply price of commodity c. When $c = cc$ it is an own commodity effect and when $c \neq cc$ it is a cross commodity effect.

This model may be formulated in GAMS in the following way:

```
Set commodities / corn, wheat /;
Set curvetype   / supply, demand/;

Table intercepts(curvetype,commodities)
                  corn   wheat
         demand    4       8
         supply    1       2;
```

```
Table slopes(curvetype,commodities,commodities)
                    corn  wheat
     demand.corn    -.3    -.1
     demand.wheat   -.07   -.4
     supply.corn     .5     .1
     supply.wheat    .1     .3  ;

Positive Variables  P(commodities)
                    Qd(commodities)
                    Qs(commodities);

Equations      PDemand(commodities)
               PSupply(commodities)
               Equilibrium(commodities);

Alias (cc,commodities);

Pdemand(commodities)..
    P(commodities) =g= intercepts("demand",commodities)
                       + sum(cc,slopes("demand",commodities,cc)*Qd(cc));
Psupply(commodities)..
    intercepts("supply",commodities)
    + sum(cc,slopes("supply",commodities,cc)* Qs(cc)) =g= P(commodities);

Equilibrium(commodities)..
     Qs(commodities) =g=  Qd(commodities);

Model problem / Pdemand.Qd, Psupply.Qs,Equilibrium.P /;
solve problem using MCP;
```

4.12.6 Components of the Revised Example Models

The revised example models have several new features including sets, specific data entry, and variables and equations that are defined over sets. In addition, if the models are run, modelers will notice some differences in the output. In this section we will discuss these new language features and the differences in the output they entail.

4.12.6.1 Sets in the Revised Examples

In the revised examples we used the subscripts `i`, `j`, `commodities` and `cc`. In GAMS, subscripts are sets. They have to be defined before they may be used as subscripts. Sets are defind with set statements. Consider the set statement from the first revised example:

```
Set  j   / corn, wheat, cotton /
     i   / land, labor /;
```

Note that the statement begins with the keyword `set` followed by the name of the set and a list of the elements of the set. Note further, that more than one set may be defined with one set statement. In addition, an optional explanatory text may be inserted after the name of the set and also after each set element. For more details on set definitions, see chapter Set Definition.

Observe that the following line appears in the revised second example:

```
Alias (cc,commodities);
```

This is an alias statement that introduces a new alternative name for a set that was defined earlier. In our example, the set `commodities` was defined in the first line of the code and `cc` is the alias, the alternative name for the set `commodities`. Note that more than one alias may be defined in an alias statement. For further information on aliases, see section The Alias Statement: Multiple Names for a Set.

Note that in our example the alias facilitates to consider both the effects of own and cross commodity quantity on the demand and supply price for an item.

4.12.6.2 Data Entry in the Revised Examples

GAMS provides three formats for data entry: scalars, parameters and tables. Usually scalars are defined with a scalar statement, data vectors are defined with a parameter statement and matrices are defined with a table statement.

Note that we used parameter and table statements in the revised examples. In this section we will discuss parameter and table statements. For details on scalar statements, see section Scalars.

Parameters

The parameter format is used to enter items defined over sets. Generally, the parameter format is used for data items that are one-dimensional (vectors), but multi-dimensional data may be entered with a parameter statement as well. Consider the parameter statement from the first revised example:

```
Parameter c(j)     / corn 109, wheat 90, cotton 115 /
          b(i)     / land 100, labor 500 /;
```

Note that the statement begins with the keyword `parameter` followed by the name of the parameter and the set over which the parameter is defined, the *index* or *domain*. Then a list follows where a numerical value is assigned to each member of the index set. Note that the referenced elements must have been defined to be members of the respective set. By default, elements of the domain that are not listed in the parameter statement are assigned the value of zero. Note that more than one parameter may be defined with one parameter statement. For further details on parameter statements, see section Parameters.

Tables

The table format is used to enter data that are dependent on two or more sets. Consider the following two table statements from the revised examples:

```
Table a(i,j) crop data
             corn  wheat cotton
    land      1     1      1
    labor     6     4      8      ;

Table intercepts(curvetype,commodities)
                   corn   wheat
         demand     4       8
         supply     1       2;
```

Note that the statement begins with the keyword `table` followed by the name of the table and the sets over which the table is defined. The next line serves as header for the columns of the table, the elements of the set in the *second* index position are listed here. The elements of the set in the *first* index position are the headers of the rows. Thus the elements of the two index positions span a grid where numerical values may be entered. Like in the parameter format, the referenced set elements must have been defined to be members of the respective set.

Note

Alignment is crucial in table statements. The numerical entries must be placed in one and only one column of the table.

By default, elements of the domain that are not listed in the table statement are assigned the value of zero. Note that only one table may be defined with a table statement. For further details on table statements, see section Tables. Observe that data may also be entered with assignment statements. For more information, see section Data Entry by Assignment.

4.12.6.3 Indexed Variables and Equations in the Revised Examples

When the algebraic structure of a problem is exploited in modeling, variables and equations are often defined over one or more sets, they are *indexed*. For example, in the first revised example we have the following lines:

```
Positive Variable x(j);
Equations         constraint(i);
```

Note that here the variable `x` is defined over the set `j` and the equation `constraint` is indexed over the set `i`. Similarly, in the second revised example we have the following variable and equation statements:

```
Positive Variables P(commodities)
                   Qd(commodities)
                   Qs(commodities) ;
Equations  PDemand(commodities)
           PSupply(commodities)
           Equilibrium(commodities)  ;
```

Observe that here all positive variables and all equations are indexed over the set `commodities`. Such definitions indicate that the variables and equations are potentially defined for every element of the defining set. Thus, for example, a variable `P` could exist for each and every element of the set `commodities`. However, how many of these potential cases are activated is determined by the respective equation definion statement(s) where the variable `P` is used. For further details on indexed variables and equations,see chapter Variables and section Indexed Equations respectively.

Next, we will discuss the equation definition statements. The respectives lines from the first revised example follow:

```
objective..      profit =e= sum(j, (c(j))*x(j));
constraint(i)..  sum(j, a(i,j) *x(j)) =l= b(i);
```

Note that the equation `constraint` is indexed over the set `i` and there are no restrictions specified in the equation definition statement. Thus, GAMS will generate a separate equation for every element of the set `i` in the model generation phase.

The same logic applies to the indexed equations of the second revised example whose definition statements are repeated below:

```
Pdemand(commodities)..
    P(commodities) =g= intercepts("demand",commodities)
                       + sum(cc,slopes("demand",commodities,cc)*Qd(cc));

Psupply(commodities)..
    intercepts("supply",commodities) + sum(cc,slopes("supply",commodities,cc)* Qs(cc))
                     =g= P(commodities);

Equilibrium(commodities)..
    Qs(commodities) =g=  Qd(commodities);
```

Observe that equations may be defined over only a part of their domain. This restriction is usually achieved with dollar conditions. For details see section Conditional Equations.

4.12.6.4 Differences in the Output

If variables and equations are defined over sets, some parts of the listing file will look different. In particular, there are some changes in the equation listing, the variable listing and the solution listing.

Revised Models: The Equation Listing

Note that indexed variables are given with their indices in the equation listing. In addition, the specific equations generated for each element of the domain are listed under the name of an indexed equation. To illustrate, we present the equation listing of the first revised example below:

```
---- objective  =E=

objective..  - 109*x(corn) - 90*x(wheat) - 115*x(cotton) + profit =E= 0 ; (LHS = 0)


---- constraint  =L=

constraint(land)..  x(corn) + x(wheat) + x(cotton) =L= 100 ; (LHS = 0)

constraint(labor)..  6*x(corn) + 4*x(wheat) + 8*x(cotton) =L= 500 ; (LHS = 0)
```

Revised Models: The Column Listing

Similar to indexed equations in the equation listing, each instance of a variable is listed under the name of an indexed variable in the variable listing. The respective output of the first revised model follows.

```
---- x

x(corn)
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
     -109       objective
        1       constraint(land)
        6       constraint(labor)

x(wheat)
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
      -90       objective
        1       constraint(land)
        4       constraint(labor)

x(cotton)
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
     -115       objective
        1       constraint(land)
        8       constraint(labor)
```

Revised Models: The Solution Listing

In the solution listing of the revised models there is a separate line for each element of the index set. The respective output of the first revised model is given below:

```
---- EQU constraint

             LOWER          LEVEL          UPPER         MARGINAL

land         -INF         100.0000       100.0000        52.0000
labor        -INF         500.0000       500.0000         9.5000

---- VAR x

             LOWER          LEVEL          UPPER         MARGINAL

corn            .           50.0000        +INF             .
wheat           .           50.0000        +INF             .
cotton          .              .           +INF          -13.0000
```

4.12.7 Documenting the GAMS Code

We have now covered the essential GAMS features. However, any good code includes documentation, otherwise it might be useless if it is revisited after a longer time. GAMS offers three ways to document the code: comments, explanatory texts and naming.

Comments

There are several ways to add comments in GAMS. The most common are *single line comments* and *block comments*. Single line comments may be inserted on any line by placing an asterisk $*$ in the first column position. The text that follows the asterisk will be completely ignored by the GAMS compiler and may contain any content including GAMS keywords. Note that several successive lines may be single line comments. Block comments are marked with the dollar control option $ontext at the beginning of the comment block and the dollar control option $offtext at the end of the comment. Block comments usually span several lines, but they may also contain just one line on the one hand and whole sections of the code on the other hand.

In addition, users may freely enter blank lines to set off certain sections of the code and generally enhance readability. For further details on comments, see section Comments.

Explanatory Text

Declarations of sets, parameters, variables and equations may include an optional descriptive text that follows the name of the identifier. In addition, the elements of sets may be accompanied by a text. This text is more than a comment: it is retained by GAMS and is displayed whenever results are written for the respective identifier in the output file. Note that the explanatory textmay be quoted or unquoted and single or double quotes may be used, but they must match. An example is given below. For further details, see section Text.

Naming

Apart from avoiding reserved words, names for identifiers in GAMS may be freely chosen. Some modelers, particularly if they have a background in mathematics, prefer short names like `x(i)`. Other modelers strongly prefer longer descriptive names that makes it easier for them to recall what quantities are denoted. In this case naming is regarded as part of the documentation. GAMS accomodates both styles and modelers may choose which style works best for them. If short names are used, we recommend to generously use descriptive texts and comments for documentation.

To illustrate the virtues of comments, blank lines, explanatory text and long names we will repeat the code of the revised LP from above and then offer an alternative, documented formulation.

```
Set       j        / corn, wheat, cotton /
          i        / land , labor /;

Parameter c(j)     / corn 109, wheat 90, cotton 115 /
          b(i)     / land 100 , labor 500 /;

Table     a(i,j)
            corn    wheat    cotton
land          1        1        1
labor         6        4        8      ;

Positive Variables x(j);
Variables          profit;

Equations          objective
                   constraint(i) ;

objective..      profit =e= sum(j, (c(j))*x(j));
constraint(i)..  sum(j, a(i,j) *x(j))  =l=   b(i);

Model resalloc /all/;
solve resalloc using LP maximizing profit;
```

The documented version with longer names follows:

```
$ontext
  well formatted algebraic version of the first example model
$offtext
Set       products  'Items produced by farm'
              / corn   'in acres'
                wheat  'in acres'
                cotton 'in acres' /
          resources  'Resources limiting farm production'
              / land   'in acres'
                labor  'in hours' /;

Parameter netreturns(products)  'Net returns per unit produced'
              / corn 109 , wheat 90, cotton 115 /
          endowments(resources) 'Amount of each resource available'
              / land 100, labor 500 /;

Table     resourceusage(resources,products) 'Resource usage per unit produced'
                          corn    wheat  cotton
               land          1       1       1
               labor         6       4       8      ;

Positive Variables   production(products) 'Number of units produced';
Variables            profit               'Total sum of net returns';

Equations            profitAcct           'Profit accounting equation'
                     available(resources) 'Resource availability limit';

$ontext
      Specify definition of profit
$offtext

profitAcct..
```

```
      profit
      =e= sum(products, netreturns(products)*production(products));

$ontext
      Limit available resources
      Fix at exogenous levels
$offtext

available(resources)..
      sum(products,
          resourceusage(resources,products) *production(products))
      =l= endowments(resources);

Model resalloc /all/;

solve resalloc using LP maximizing profit;
```

4.12.8 Guidelines on Ordering GAMS Statements and Formatting GAMS Programs

A GAMS program is a collection of GAMS statements. In this section we will offer some general guidelines on ordering GAMS statements and formatting GAMS programs.

- GAMS is case insensitive. This applies to GAMS keywords as well as to user-defined names. For example, the keyword `VARIABLE` is identical to `Variable` and `variable` and the user-defined name `mincost` is identical to `minCost` and `minCOST`. However, note that the spelling in the output file is determined by the spelling in the first occurence of an identifier or label.
- Individual GAMS statements may be formatted in almost any style. Multiple lines may be used for a statement, blank lines may be embedded and any number of spaces or tabs may be inserted. In addition, several statements may be placed on one line. Note that they have to be separated by semicolons `;`.
- We recommend that every GAMS statement is terminated with a semicolon `;`. However, note that semicolons are not manadatory if the next word is a GAMS keyword.
- Identifiers like sets, parameters, scalars, tables, acronyms, variables, equations, models and files must first be *declared* and *defined* before they may be used in the code. An identifier is named in a declaration statement and specific values are assigned to it when it is defined. Often identifiers are defined at the same time they are declared. Note that for equations the declaration and definition statements are always distinct.
- The names for identifiers and labels and the explanatory text must follow certain rules. See chapter GAMS Programs and the tutorial Good Coding Practices for more information.
- Statements must be ordered such that identifiers are declared before they are used. If identifiers are used on the right-hand side of an assignment statement, they must also have been defined. If they are used in an equation and the equation is included in a model, then they must be assigned data before a solve statement for the respective model. Note that compilation errors will be triggered if this order is not followed.

4.12.9 Adding Complexity

There are several GAMS features that are widely used and serve to add subtlety and compexity to models. They include exception handling through conditionals, displaying data in the output file and report writing where the information from the optimal solution is used to create reports that meet the needs of modelers and their clients.

4.12.9.1 Conditionals

Assignments are often valid only for certain cases and sometimes equations should reasonably be restricted to a subset of the domain over which they were defined. It is easy to model such assignments and equations in GAMS with conditionals. Conditional expressions in GAMS are introduced and discussed in detail in chapter Conditional Expressions, Assignments and Equations. In this section we will only give a quick overview to demonstrate the capabilities of conditionals in GAMS.

Note that the dollar condition is at the heart of conditionals in GAMS. The general form of a conditional expression is as follows:

```
term $ logical_condition
```

This translates to: "Do something with 'term' only if the logical condition is true." Observe that `term` may be a number, a set, a parameter, a variable or an equation. The dollar operator `$` is a GAMS speciality and one of the most powerful features of GAMS. The logical condition may take many different forms, see section Logical Conditions for details.

The following examples illustrate how conditionals in GAMS may be used.

Conditional Assignments

Consider the following example:

```
x $ (y > 0) = 10;
```

Note that `x` is assigned the value of ten only if the the scalar `y` is greater than zero, otherwise no assignment is made and `x` keeps its previous value. For more information, see section Conditional Assignments.

Conditional Indexed Operations

Consider the following example:

```
z = sum[i $ (y(i) <> 0), x(i)];
```

Note that the term `x(i)` will only be included in the sum if `y(i)` is nonzero. For further information, see section Conditional Indexed Operations.

Conditionals in the Domain of Definition of Equations

Consider the following equation definition statements:

```
Eq1 $ (qq > 0)..            xvar =e= 3;
Eq2 $ (sum(i, q(i)) > 0)..  yvar =l= 4;
Eq3(i) $ (a(i) > 0)..       ivar(i) =g= -a(i);
```

Note that in each equation the domain of definition is restricted to those cases where the logical condition evaluates to `TRUE`. For further details, see section Dollar Control over the Domain of Definition.

Conditionals in the Algebraic Formulation of Equations

Our last example illustrates how a dollar condition may be used in the body of an equation:

```
Eq4 . .    xvar + yvar $ (qq > 0) =e= 3;
```

Note that the term `yvar` is included in the equation only if `qq` is greater than zero, otherwise `yvar` is treated as if it were zero. For more information, see section Dollar Operators within the Algebra of Equations.

4.12.9.2 Displaying Data

In GAMS, the display statement is a quick way to write data into the output file. For example, assume we add the following statement after the solve statement in the revised and documented version of the farm linear programming model above:

```
display profit.l, production.l;
```

Recall that `profit` and `production` are variables. The suffix `.l` indicates that we wish to display the variable attribute `level`. The following output will be generated in the listing file:

```
----     47 VARIABLE profit.L              =     9950.000  Total sum of net returns

----     47 VARIABLE production.L  Number of units produced

corn  50.000,    wheat 50.000
```

Observe that the name of the variable, the explanatory text and the respective numerical values are given. In addition to data like parameters, sets, variable attributes, equation attributes and model attributes, quoted text may be displayed. Note that numerical entries that equal zero will not be displayed. For a more detailed introduction, see chapter The Display Statement.

4.12.9.3 Report Writing

In many cases modelers wish to summarize the most important results of the solution in a table for a quick overview. GAMS allows post-solution computations where information form the solution may be used to assign values to new parameters that are then displayed. The information from the solution most often used for report writing includes the level values of variables and equations and the marginal values of equations (also called *dual values* or *shadow prices*). Note that variable and equation attributes are accessed as follows:

```
var_name.sfx
eqn_name.sfx
```

Here `var_name` and `eqn_name` is the name of the variable and equation in GAMS respectively. The *attribute* is denoted by `.sfx`, where `sfx` may be `l` for level and `m` for marginal. Note that the suffix `sfx` may take other values as well. For details see sections Variable Attributes and Equation Attributes.

Observe that the numerical values of the levels and marginals of variables and equations are generally undefined until a solve statement is executed. After GAMS has retrieved the solution from the solver, the respective values from the solution are assigned to the attributes. These values remain unchanged until the next solve, where they are replaced with the values from the most recent solution.

In the remainder of this section we will present two examples to illustrate report writing. Assume we add the following report writing sequence after the solve statement to the revised and documented version of the farm linear programming model above:

```
Set item   / Total, "Use by", Marginal /;
Set qitem  / Available, Corn, Wheat, Cotton, Value /;

Parameter Thisreport(resources,item,qitem) 'Report on resources';
Thisreport(resources,"Total","Available") = endowments(resources);
Thisreport(resources,"Use by",qitem) =
   sum(products$sameas(products,qitem),
       resourceusage(resources,products) * production.l(products));
Thisreport(resources,"Marginal","Value") =
        available.m(resources);

option thisreport:2:1:2;
display thisreport;
```

Note that both, equation marginals (`available.m(resources)`) and variable levels (`production.l(products)`) are included in the calculations. The predefined symbol sameas in the logical condition above returns the value `TRUE` if the element of the set `products` is equivalent to the element of the set `qitem` and `FALSE` otherwise. Thus this condition ensures that the third index of the parameter `Thisreport` is identical to the element of the set `products` in the sum. Observe that with the option statement in the penultimate line the appearance of the display is customized. For details see section Local Display Control. The following report will be generated by the display statement:

```
----     61 PARAMETER Thisreport  Report on resources

            Total      Use by      Use by    Marginal
        Available        corn       wheat       Value

land       100.00       50.00       50.00       52.00
labor      500.00      300.00      200.00        9.50
```

Similarly, we could add the following report writing sequence to the revised version of the equilibrium model above:

```
Set         qitem / Demand, Supply, "Market Clearing" /;
Set         item  / Quantity, Price /;

Parameter myreport(qitem,item,commodities);
myreport("Demand","Quantity",commodities) = Qd.l(commodities);
myreport("Supply","Quantity",commodities) = Qs.l(commodities);
myreport("Market Clearing","Price",commodities) = P.l(commodities);

display myreport;
```

Note that in the new parameter the level values for supply and demand as well as the market clearing price are saved. The resulting report follows:

```
----     39 PARAMETER myreport

                              Corn       Wheat
Supply          .Quantity     1.711       8.156
Demand          .Quantity     1.711       8.156
Market Clearing.Price         2.671       4.618
```

For more on report writing, see chapters The Display Statement and The Put Writing Facility.

4.12.10 Advantages of Algebraic Modeling in General

We will conclude this tutorial with a discussion of the advantages of using algebraic modeling in general.

Algebraic modeling languages like GAMS facilitate model formulations in general algebaric terms, that are very concise and readable. Language elements that are essential include sets that may serve as indices, algebraic expressions, indexed operations, powerful sparse index and data handling variables and constraints with user-defined names. Model formulations are largely independent of the data and exact application contexts. Such formulations may be easily transferred to different contexts, data may be added without the need to reformulate the model and the model may be extended to reflect additional complexity.

However, GAMS algebraic requirements and the summation notation are difficult for some users. Some modelers will always prefer the exact problem context, not an abstract general formulation. This may lead to a strategy most modelers use: Start with a small concrete formulations that capture the essence of the problem and support the development of more general GAMS models.

4.12.10.1 One Model - Different Contexts

In the linear programming problem above we modeled profit maximizing in a farm. This model may easily be transferred to another context as follows:

```
Set       products  'Items produced'
            / Chairs, Tables, Dressers /
          resources  'Resources limiting production'
            / RawWood, Labor, WarehouseSpace /;

Parameter Netreturns(products)  'Net returns per unit produced'
            / Chairs 19, Tables 50, Dressers 75 /
          Endowments(resources) 'Amount of each resource available'
            / RawWood 700, Labor 1000,  WarehouseSpace 240 /;

Table     Resourceusage(resources,products) 'Resource usage per unit produced'
                          Chairs   Tables  Dressers
          RawWood            8        20      32
          Labor             12        32      45
          WarehouseSpace     4        12      10   ;

Positive Variables   Production(products) 'Number of units produced';
Variables            Profit               'Total sum of net returns' ;

Equations            ProfitAcct           'Profit accounting equation '
                     Available(resources) 'Resource availability limit';

 ProfitAcct..
      Profit
      =e= sum(products, netreturns(products) * production(products)) ;
 Available(resources)..
      sum(products,
          resourceusage(resources,products) * production(products))
      =l= endowments(resources);

Model resalloc /all/;

solve reasalloc using LP maximizing Profit;
```

Note that in this model we have chairs, tables and dressers instead of corn, wheat and cotton and raw wood, labor and warehouse space instead of land and labor, but the *algebraic structure* of the model is the same. Thus we still have sets for products and resources, parameters for net returns per unit produced and available resources, a table for resource usage per unit produced and exactly the same variables and equations, the same model and solve statement. Hence, if the algebraic structure for a type of problem is built, it may be used in another context of the same problem type with just minor modifications in the data.

4.12.10.2 Adding More Data

It is easy to add more data to a model. For example, we could add two new products and two new resources to the carpenter model above in the following way:

```
Set       products  'Items produced'
              / Chairs, Tables, Dressers, HeadBoards, Cabinets /
          resources  'Resources limiting production'
              / RawWood, Labor, WarehouseSpace, Hardware, ShopTime /;

Parameter Netreturns(products)  'Net returns per unit produced'
            / Chairs 19, Tables 50, Dressers 75, HeadBoards 28, Cabinets 25 /
          Endowments(resources) 'Amount of each resource available'
            / RawWood 700, Labor 1000, WarehouseSpace 240, Hardware 100, Shoptime 600 /;

Table     Resourceusage(resources,products) 'Resource usage per unit produced'
                          Chairs   Tables  Dressers HeadBoards Cabinets
          RawWood            8        20      32         22      15
          Labor             12        32      45         12      18
          WarehouseSpace     4        12      10          3       7
          Hardware           1         1       3          0       2
          Shoptime           6         8      30          5      12;

Positive Variables   Production(products) 'Number of units produced';
Variables            Profit               'Total sum of net returns' ;

Equations            ProfitAcct           'Profit accounting equation'
                     Available(resources) 'Resource availability limit';

 ProfitAcct..
      Pofit
      =e= sum(products, netreturns(products) * production(products)) ;

 Available(resources)..
      sum(products,
          resourceusage(resources,products) * production(products))
      =l= endowments(resources);

Model resalloc /all/;

solve reasalloc using LP maximizing Profit;
```

Observe that the elements `HeadBoards` and `Cabinets` were added to the set `Products` and the elements `Hardware` and `ShopTime` were added to the set `Resources`. In addition, the data in the two parameters and the table was updated to reflect these new labels. However, the model structure remained unchanged. Thus, GAMS models may easily be extended from smaller to larger data sets. Note that this feature may be exploited for model development. Users may develop a model with a small data set and test and debug it. Afterwards, they may move to the full problem data set without having to alter the algebraic structure of the model.For more details on this strategy, see section Small to Large: Aid in Development and Debugging.

4.12.10.3 Extending the Model

Assume we wish to make a model more complex by adding new features. For example, we could extend the carpenter model above to reflect the possibility of renting or hiring additional resources subject to a maximum limit. Consider the following code:

```
Set       products  'Items produced'
              / Chairs, Tables, Dressers /
          resources  'Resources limiting production'
              / RawWood, Labor, WarehouseSpace/
          hireterms  'Resource hiring terms'
              / Cost, Maxavailable /;

Parameter Netreturns(products)  'Net returns per unit produced'
              / Chairs 19, Tables 50, Dressers 75 /
          Endowments(resources) 'Amount of each resource available'
              / RawWood 700, Labor 1000, WarehouseSpace 240 /;

Table     Resourceusage(resources,products) 'Resource usage per unit produced'
                          Chairs   Tables  Dressers
          RawWood            8        20      32
          Labor             12        32      45
          WarehouseSpace     4        12      10   ;

Table     Hiredata(resources,hireterms)  'Resource hiring data'
                          Cost   Maxavailable
          RawWood            3        200
          Labor             12        120
          WarehouseSpace     4        112;

Positive Variables   Production(products)    'Number of units produced'
                     HireResource(resources) 'Resources hired';
Variables            Profit                  'Total sum of net returns' ;

Equations            ProfitAcct              'Profit accounting equation'
                     Available(resources)    'Resource availability limit'
                     Hirelimit(resources)    'Resource hiring limit';

 ProfitAcct..
      Profit
      =e= sum(products, Netreturns(products) * Production(products))
         - sum(resources, Hiredata(resources,"cost") * HireResource(resources))   ;

 Available(resources)..
      sum(products,
          Resourceusage(resources,products) * Production(products))
      =l= Endowments(resources) +  HireResource(resources);

 Hirelimit(resources)..
          HireResource(resources) =l= Hiredata(resources,"Maxavailable");

Model resalloc /all/;

solve reasalloc using LP maximizing Profit;
```

Observe that we introduced the set `hireterms`, the table `Hiredata`, the positive variable `HireResource` and the equation `Hirelimit`. In addition, we included new terms in the equations `Profit` and `Available`

to reflect that through hiring the resources are increased, but hiring comes with a cost diminishing the profit. Thus the algebraic structure of the earlier model could be used as the core for this model that has additional features.

Note that this method may also be exploited for model development. Users may adapt models from other studies customizing them for the problem at hand and thus speeding up the development process. In addition to adapting models from related earlier studies that were done by the modeler or his group, model development may be jumpstarted by adapting models from the extensive GAMS Model Library.

4.13 Good Coding Practices

The GAMS language is quite flexible regarding the syntax and format of the code it accepts, offering users considerable latitude in how they organize and format their GAMS code. Most modelers develop their own style as they gain experience with the GAMS system. This tutorial reflects the coding preferences of Bruce A. McCarl (currently professor of Agricultural Economics at Texas A&M University). Note that Bruce has extensive experience with GAMS, both as a modeler and an educator, and many GAMS users know, use, and benefit from his work. The goal of this tutorial is *not* to present a rigid set of rules to follow arbitrarily, but rather to help users develop their own coding preferences and style. The larger goal is to build self-documenting models that are easy to read and understand, to edit, and to debug: both for the developer working in the present, and for a larger group of colleagues and consultants working with the model over a span of months or years.

We will cover the following topics:

- Using Longer Names and Descriptive Text
- Including Comments
- Choosing Raw Data Instead Of Computed Data
- Avoiding the Universal Set in the Context of Data Input
- Defining Sets and Subsets Wisely
- Structuring and Formatting Files to Improve Readability
- Other Suggestions

4.13.1 Using Longer Names and Descriptive Text

The readability of GAMS code may be significantly improved by using longer self-explanatory names for identifiers (e.g. names of sets, parameters, variables, etc). Consider the following lines of code from the production and inventory model [**ROBERT**]:

```
Sets  p     'products'       / low, medium, high /
      r     'raw materials'  / scrap, new /
      tt    'long horizon'   / 1*4 /
      t(tt) 'short horizon'  / 1*3 /;

Table  a(r,p)  input 'coefficients'

          low  medium  high
 scrap      5     3      1
 new        1     2      3;
```

```
Table  c(p,t)  'expected profits'

           1    2    3
low       25   20   10
medium    50   50   50
high      75   80  100;

Variables  x(p,tt)  'production and sales'
           s(r,tt)  'opening stocks'
           profit;

 Positive variables x, s;

 Equations  cc(t)    'capacity constraint'
            sb(r,tt) 'stock balance'
            pd       'profit definition' ;

 cc(t)..        sum(p, x(p,t)) =l= m;

 sb(r,tt+1)..   s(r,tt+1) =e= s(r,tt) - sum(p, a(r,p)*x(p,tt));

 pd..           profit =e= sum(t, sum(p, c(p,t)*x(p,t))
                          - sum(r, misc("storage-c",r)*s(r,t)))
                          + sum(r, misc("res-value",r)*s(r,"4"));

 s.up(r,"1") = misc("max-stock",r);
```

These lines may be reformatted in the following way (see (good.gms)):

```
Sets process          'available production process'
       / low    'uses a low amount of new materials',
         medium 'uses a medium amount of new materials',
         high   'uses a high amount of new materials' /
     rawmateral        'source of raw materials' / scrap, new /
     Quarters          'long horizon'            / spring, summer, fall, winter /
     quarter(Quarters) 'short horizon'           / spring, summer, fall /

 Table  usage(rawmateral,process)  'input coefficients'
          low  medium  high
 scrap      5     3      1
 new        1     2      3

 Table  expectprof(process,quarters)  'expected profits'
         spring summer fall
 low        25    20    10
 medium     50    50    50
 high       75    80   100;

 Variables  production(process,Quarters)    'production and sales'
            openstock(rawmateral,Quarters)  'opening stocks'
            profit ;

 Positive variables production, openstock;

 Equations  capacity(quarter)               'capacity constarint'
            stockbalan(rawmateral,Quarters) 'stock balance'
            profitacct                      'profit definition' ;
```

```
capacity(quarter)..
       sum(process, production(process,quarter)) =l= mxcapacity;

stockbalan(rawmateral,Quarters+1)..
     openstock(rawmateral,Quarters+1) =e=
     openstock(rawmateral,Quarters)
     - sum(process, usage(rawmateral,process)
                  *production(process,Quarters));

profitacct.. profit =e=
  sum(quarter,
      sum(process, expectprof(process,quarter)
                   *production(process,quarter))
    - sum(rawmateral, miscdata("store-cost",rawmateral)*
                      openstock(rawmateral,quarter)))
    + sum(rawmateral, miscdata("endinv-value",rawmateral)
                     *openstock(rawmateral,"winter"));

openstock.up(rawmateral,"spring") = miscdata("max-stock",rawmateral);
```

Note that the two formulations are equivalent in their effect, but in the second formulation longer, more descriptive names were used for the sets, tables, variables and equations. In addition, longer names were used for the set elements and in the definition of the set `process` the set elements have additional explanatory text. Observe that the second formulation is easier to understand. This will be particularly useful if and when the code is revisited in 5 years' time.

Note

- Recall that GAMS allows long names for identifiers and labels (set elements). Users may exploit this feature to introduce long descriptive names. However, note that names for labels that are longer than 10 characters do not work well in multi-column displays. See the paragraph on customizing display width for details.
- Use explanatory text for identifiers to indicate units, sources, descriptions, etc. It's not that hard to do and it pays dividends later.
- Similarly, use explanatory text for set elements as appropriate.

For example, the descriptive text in the in the second line in the following code snippet is much more informative than the text in the first line:

```
Parameter vehsales(r)   'regional vehicle sales';
Parameter vehsales(r)   'regional vehicle sales ($ millions/yr)';
```

Note that the descriptive text will be displayed whenever the respective identifier is displayed. Hence, including units in the text will save time if results will have to be interpreted later.

4.13.2 Including Comments on Procedures and the Nature and Sources of Data

We recommend that the documentation of the code offers answers to the following questions:

- What are the units of the variables and parameters?

- Where did the data come from?
- What are the characteristics of the data such as units and year of applicability?
- Why was a constraint set up in the way it is implemented?

In addition, it is often helpful to add comments that describe assumptions, the intent of equation terms, data sources, including document name, page number, table number, year of applicability, units, URL etc.

Consider the following example where various forms of comments are illustrated:

```
* this is a one line comment that could describe data
$ontext
My data could be described in this multi-line comment
This is the second line
$offtext

* The following dollar control option activates end-of-line comments
* and redefines the symbol for end-of-line comments
$eolCom #
x = sum(i, z(i)) ; # this is an end-of-line comment

* The following dollar control option activates in-line comments
* and redefines the symbols for in-line comments
$inLineCom (*  *)
x = sum(i, z(i)) ; (* this is an in-line comment *) r = sum(i, z(i)) ;
```

For more information on comments in GAMS, see section Comments.

4.13.3 Choosing Raw Data Instead Of Computed Data

Modelers often have a choice how they enter data: they could either use raw data and transform it to the extent needed inside GAMS or process data externally and enter the final results into GAMS.

The second choice may be attractive if the raw data is available in a spreadsheet where it can be manipulated before it is introduced to GAMS. However, over time spreadsheets and other data manipulation programs change or get lost and often these programs are not documented well. Therefore, we recommend to enter data into GAMS in a form that is as close as possible to the actual collected data and then manipulate the data with GAMS to obtain the desired form. This will make it much easier to update models later or to work out implicit assumptions.

4.13.4 Avoiding the Universal Set in the Context of Data Input

While GAMS permits using the universal set $*$ as an index in a parameter or table statement, in most cases it is *not* advisable to do so. Consider the following example from the production and inventory model [**ROBERT**]:

```
Sets   r          'raw materials'  / scrap, new / ;

Table  misc(*,r)  'other data'
            scrap  new
 max-stock    400  275
 storage-c     .5    2
 res-value     15   25   ;
...

pd.. profit =e= sum(t, sum(p, c(p,t)*x(p,t))
              - sum(r, misc("storage-c",r)*s(r,t)))
              + sum(r, misc("res-value",r)*s(r,"4"));
```

Note that the definition of the table `misc` indicates that any entry in the first index position is allowed. There is no domain checking. Consequently, if the label `res-value` is misspelled as `res-val` in the equation `pd`, GAMS will compile and execute the program without signaling an error, but instead of the expected values (i.e. `misc(r,"res-value")`), the values of `misc(r,"res-val")` will be used in the equation. These zero values will lead to faulty results, and the modeler will not be alerted to this fact. To ensure that the results of a GAMS run are reliable and trustworthy, we strongly recommend to use domain checking by introducing a new set for the labels in the first index position of the table `misc`:

```
Sets   r         'raw materials'
         / scrap, new /
       miscitem  'misc input items'
         / max-stock, storage-c, res-value /;

Table  misc(miscitem,r)  'other data'
            scrap  new
 max-stock    400  275
 storage-c     .5    2
 res-value     15   25   ;
```

Observe that the new set `miscitem` contains exactly the labels that appear in the rows of the table `misc`. Hence the set `miscitem` may be used in the first index position of the definition of `misc` without loss of generality, but with the benefit of domain checking.

4.13.5 Defining Sets and Subsets Wisely

Generally, the elements of a set have a feature in common or they are similar in some way. In this section we will give some guidance on how to partition the labels in the data into sets. In addition, we will discuss in which contexts it is useful to introduce subsets. For an introduction to sets in GAMS, see chapter Set Definition.

For example, suppose we have three grades of oil and three processes to crack it. The question arises whether we should introduce one set with nine elements or two sets with three elements and a two-dimensional set. We recommend the second alternative.

In another example, we consider a budget for farm spending: we have annual (i.e. cumulative yearly) spending for fertilizer and for seed and also monthly spending for labor and for water. There are 26 decisions or items in the budget. We could introduce a set with 26 elements or we could use the following formulation:

```
Sets resources  / fertilizer, seed, labor, water /
     periods    / jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec, annual /
     use(resources,periods)
                / (fertilizer,seed).annual
                      (labor,water)    .(jan,feb,mar,apr,may,jun,jul,aug,sep,oct,nov,dec) /;
```

We recommend the formulation above and to err on the side of being more extensive or exact with set definitions.

Occasionally it is necessary to group some labels into one set for a certain purpose and then single out some of them for another purpose. Subsets facilitate modeling such a case. For example, a set of all cities in a model may be needed to enter distances and compute related transportation costs. In addition, a subset can be used to specify the cities that are hubs for some activity, since some equations should be restricted to these hubs.

4.13.6 Structuring and Formatting Files to Improve Readability

In this section we will offer some guidlines on structuring and formatting the GAMS code to make it easy to read.

There are several ways to structure the GAMS code. Two styles are outlined in section Organization of GAMS Programs. The following recommendation to enter the sections of the code in a fixed order is an extended version of the first style:

1. Set definitions for sets that are data related
2. Parameter, scalar and table definitions, possibly intermixed with calculations
3. Variable definitions
4. Equation declarations
5. Equation definitions (algebraic specification of equations)
6. Model and solve statement(s)
7. Definitions of sets and parameters for report writing
8. Calculations for report writing
9. Display statement(s) for reports

Note that the code will be easiest to navigate if each section of the code contains only one type of statements. For example, interspersing set definitions with parameter definitions will make the code unnecessarily difficult to read.

In addition to following a fixed structure, it is also essential to properly format the code. Of course, formatting is in many respects a matter of taste. The following list offers some ideas:

- Align the names of identifiers and descriptive text, as demonstrarted in the examples in this tutorial and in the GAMS User's Guide in general.
- Use spacing and indents.
- Use blank lines to highlight something and to mark sections of the code.
- Ensure that variables and all their index positions are on one line in equation definitions.

- Indent in indexed operations like sums and programming flow control structures like loops and if statements to delineate terms. The structure of a long and complex statement may be revealed through careful indentation and thoughtful placement of closing parentheses.

We will demonstrate the effect of proper formatting with the following two examples. The first example contains valid GAMS code, but is deliberately poorly formatted:

```
Sets  products  'available production process' / low 'uses low new materials'
medium 'uses medium new materials', high 'uses high new materials'/
rawmateral    'source of raw materials'  / scrap, new /
Quarters    'long horizon'   / spring, summer, fall ,winter /
quarter(Quarters) 'short horizon'  / spring, summer, fall / ;
Variables  production(products,Quarters)  'production and sales' ;
openstock(rawmateral,Quarters)  'opening stocks', profit ;
Positive variables production, openstock;
Equations  capacity(quarter)  'capacity constraint',
stockbalan(rawmateral,Quarters) 'stock balance',
profitacct  profit definition ;
profitacct.. profit =e= sum(quarter, sum(products, expectprof(
products,quarter) *production(products,quarter))-sum(
rawmateral,miscdata("store-cost",rawmateral)*openstock(rawmateral
  ,quarter)))+ sum(rawmateral, miscdata("endinv-value",rawmateral)  *openstock(rawmateral,"winter"))
```

The second example contains the same code as the first example, but is carefully formatted:

```
Sets products          'available production process'
       / low    'uses a low amount of new materials',
         medium 'uses a medium amount of new materials',
         high   'uses a high amount of new materials' /
     rawmateral        'source of raw materials' / scrap, new /
     Quarters          'long horizon'            / spring, summer, fall, winter /
     quarter(Quarters) 'short horizon'           / spring, summer, fall / ;

Variables  production(products,Quarters)   'production and sales'
           openstock(rawmateral,Quarters)  'opening stocks'
           profit ;
Positive Variables production, openstock ;

Equations  capacity(quarter)               'capacity constraint'
           stockbalan(rawmateral,Quarters) 'stock balance'
           profitacct                      'profit definition' ;

profitacct..
    profit =e=
    sum(quarter,
       sum(products, expectprof(products,quarter)
                   *production(products,quarter)
           )
      - sum(rawmateral, miscdata("store-cost",rawmateral)*
                        openstock(rawmateral,quarter)
           )
       )
   + sum(rawmateral, miscdata("endinv-value",rawmateral)
                  *openstock(rawmateral,"winter")
       )
  ;
```

Observe that inserting blank lines, aligning the names of identifiers and descriptive text, and indenting and formatting closing parentheses in the sums makes the code much easier to read and understand (both now and in the future) and is well worth adopting as standard practice when writing GAMS code.

4.13.7 Other Suggestions

We will complete this tutorial by offering some other useful suggestions that may help modelers develop their own conventions.

Even though GAMS is case insensitive, it is advisable to establish some convention on the use of upper and lower case letters. For example, Paul N. Leiby (currently at Oak Ridge National Laboratory) uses lower case for texts and comments, and upper case for GAMS reserved words and variable and parameter names. The casing used when an identifier or label is first encountered in a GAMS program is the casing stored by GAMS and used in subsequent outputs like the listing file or a GDX file. Any casing can be used (so `noWhere` is equivalent to `nowHere`) but the casing stored is determined by first use.

A similar situation holds for label quoting: the type of quotes stored (if any) are determined by first use.

Note that the dollar control option $onSymList will cause a list of all identifier names to be displayed in the compilation output of the output file. This list may be used to review the spelling and casing of the identifiers as they will appear in output files. Similarly, the dollar control option $onUELList will cause an ordered list of all labels to be displayed in the compilation output of the output file. This is useful for checking both the case and order of the labels used in the GAMS program. For more on issues related to label ordering, see section Ordered and Unordered Sets.

To keep track of the data types of identifier names, some modelers always start set names with `s_`, names of parameters with `d_` (for "data"), variables names with `v_` and equation names with `e_`.

Some experienced GAMS users always surround explanatory text with quotes: this makes the text stand out, prevents it from being interpreted as a label or identifier, and allows special characters like `$`, `-` and `&` to be used.

If a file is used by several modelers and is updated occasionally, a file modification log at the top of the file will be in order. It should contain the following information: the modification date, version number, modification(s) made and who made the modification. For example, a set called `version` may be used to keep track of the dates the input files were modified:

```
Set version(*,*,*,*);
. . .
version("my_file","May","19","2016") = yes;
version("my_include_file","Sep","30","2016") = yes;
. . .
display version;
```

Note that the display statement will generate a display of all elements of the set `version`, each indicating on which day a component of the model was modified.

4.14 Fixing Compilation Errors

As detailed in chapter GAMS Output, the execution of a GAMS program passes through several stages, where the *compilation* is the first stage. Often when a program is run for the first time, it cannot be solved successfully because of compilation errors. This can be very frustrating, especially for new users. In this tutorial we will explain the causes of the most common compilation errors and offer some advice on how to fix them. For an introduction to compilation errors, see section Compilation Errors. In addition, the tutorial A GAMS Tutorial by Richard E. Rosenthal offers some detailed material on this topic.

Note

> Frequently, many compilation errors in the latter part of the code are actually consequential errors that will disappear as soon as the compilation errors in the beginning have been resolved. Therefore we recommend to start with fixing the first few errors and to run the program again. More often than not, many subsequent compilation errors will have vanished.

4.14.1 Preliminary Remarks

Before we will turn to an overview of the most common errors, examples that illustrate them and advice how to resolve them, we will provide some basic information on compilation errors and error messages.

4.14.1.1 Finding Compilation Errors

Compilation errors are marked with four asterisks (****) in the compilation output of the GAMS listing file (also called output file or lst file, since it has the extension `.lst`), hence it is easy to detect lines where compilation errors occured by inspection.

Note

Compilation error messages also appear in the LOG file. And, if GAMS Studio is used, a click at an error message in the LOG will navigate directly to the problematic code in the source file.

Consider the following simple example:

```
Set c "crops"   / barley, wheat, soy, wheat, rice /;
```

If this set statement appears in a program, the resulting compilation output will contain the following lines:

```
   1  Set c "crops"   / barley, wheat, soy, wheat, rice /;
****                                        $172

...
Error Messages

172  Element is redefined
```

Note that in the echo print of the input file a line starting with **** is inserted and the dollar sign `$` followed by a number appears on this line. This indicates that a compilation error - in this case error `172` - was discovered in the line above. In addition, a list of all errors with explanatory error messages is given at the end of the echo print.

In our example, the error refers to `wheat` and the error message addresses the cause of the error: the respective element is redefined. Thus we check the other elements of the set and quickly realize that `wheat` appears a second time. If we delete the second instance of `wheat` and run the code again, this line will not cause any errors anymore. Note that many compilation errors are as easy to fix as demonstrated here.

4.14.1.2 Repositioning Compilation Error Messages

By default, the error messages are shown directly beneath the line where the respective error is marked. GAMS allows to customize this position with the command line parameter ErrMsg: the value of 0 for `ErrMsg` will list all error messages at the end of the echo print and the value of 2 will suppress the error messages in the lst file completely. Note that the default is 1.

Consider the following simple example.

```
Set       a    / a1 * a7 /;
Set       b(a) / a7 * a9 /;
Parameter p(k) / i3 47 /;
```

The resulting compilation output will contain the following lines:

```
   1  Set       a    / a1 * a7 /;
   2  Set       b(a) / a7 * a9 /;
****                          $170
**** 170  Domain violation for element
   3  Parameter p(k) / i3 47 /;
****              $120
**** 120  Unknown identifier entered as set
```

Observe that the error messages are placed on the line after the error is marked. This can be especially useful if the program contains many lines of code. Next, we run the same code with the following call:

```
  > gams test  ErrMsg=0
```

The resulting compilaton output follows.

```
   1  Set       a    / a1 * a7 /;
   2  Set       b(a) / a7 * a9 /;
****                          $170
   3  Parameter p(k) / i3 47 /;
****              $120
...
Error Messages

120  Unknown identifier entered as set
170  Domain violation for element
```

Now the error messages are displayed at the end of the compiler listing.

Note

Default values for command line parameters and environment variables can be specified using a configuration file gamsconfig.yaml. The GAMS Configuration Editor can be used to view and edit the GAMS configuration file but any other text editor can be used as well. Alternatively, users may change the system level defaults by entering the line `ErrMsg=0` in the file `gmsprmnt.txt` on Windows, or `gmsprmun.txt` on Unix machines as in the following example:

```
************************************************
* GAMS 2.50 Default Parameterfile for Windows NT                *
* Gams Development Corp.                                        *
* Date : 20 Mar, 1998                                           *
************************************************
* entries required by CMEX, put in by gams.exe:
* SYSDIR
* SCRDIR
* SCRIPTNEXT
* INPUT
errmsg=0
```

4.14.2 Resolving Common Compilation Errors

There are hundreds of compilation errors in GAMS, but some of them are particularly frequent. In the table below we present these common errors, with a brief description of the possible cause and a link to a subsection below where examples and more details are given. We recommend that users also read the error messages, since they often contain additional hints.

GAMS Error	Common Cause of Error	Subsections with Examples and More Details
8	Closing parentheses, square brackets or braces are missing.	H
36	The two dots `..` are missing in the equation definition.	I
37	The equation type (eg. `=L=`) is missing in the body of the equation.	I
51-60	There are prohibited nonlinear expressions.	J
66	A data item which has not been given numerical values appears in an equation.	K
71	The equation has been declared, but not defined.	I
96, 97	A statement followed by another statement is not terminated with `;`.	B
120	GAMS cannot find a set with this name. Check for typos in the set name and set elements that are referenced without quotes.	C, L
125	The set is controlled more than once, e.g., by an indexed operation like `sum` and by an equation definition.	F
140	GAMS is looking for a keyword or declared item and cannot find it. Check spelling and declarations.	A, C, K, M

GAMS Error	Common Cause of Error	Subsections with Examples and More Details
141	The parameter without data is used or problems with `solve` and therefore attributes `.l` and `.m` are empty.	K
148	The identifier is referenced with more or less indexed sets than in the declaration.	E
149	The set is not controlled, neither by an indexed operation like `sum`, nor by an equation definition, nor by a loop or similar.	G, L
170	The referenced set element cannot be found in the set defined for this index position. Check for typos, omissions in the set declaration, missing quotes and references to the wrong set.	C, D
171	A domain error. The wrong set is referenced for the respective index position.	E, L
195	The name used here was already used for another identifier.	N
198	Using the operation `ord` or a lag/lead operation with a set that is not ordered.	O
256	Something is wrong with the model specification. This is often a consequential error of another error. Look for other error messages immediately after the solve statement.	I, J
257	The solver is not checked. This may be a consequential error of any GAMS error.	
340	Probably the quotes in a set element reference are missing.	L
408	Surplus closing parentheses, square brackets or braces.	H

4.14.2.1 Error A: Misplaced Semicolons

Maybe the most common error for new users is related to semicolons. Consider the following example adapted from the well-known transportation model [**TRNSPORT**]:

```
Sets  i   "canning plants"   / Seattle, San-Diego / ;
      j   "markets"          / New-York, Chicago, Topeka / ;
```

The resulting compilation output will contain the following lines:

```
   20    Sets i   canning plants   / Seattle, San-Diego / ;
   21         j   markets          / New-York, Chicago, Topeka / ;
****         $140      $36
...
Error Messages

 36  '=' or '..' or ':=' or '$=' operator expected
     rest of statement ignored
140  Unknown symbol
```

What went wrong? GAMS statements like the set statement in our example have to terminate with a semicolon, unless the next line of code begins with a reserved word. Now, set statements may extend over several lines and define several sets. In our example, the set statement extends over two lines and two sets are declared. However, there is a semicolon at the end of the first line, therefore GAMS assumes that the set statement ends there. The symbol `j` at the beginning of the next line has not been declared yet and it is not a GAMS keyword, thus it is marked as `Unknown symbol`. Note that error `36` is a consequential error that will disappear as soon as the first error has been resolved.

How do we fix this? There are two ways: either we could drop the semicolon at the end of the first line and thus indicate that the set statement continues to the second line or we could insert the keyword `Set` at the start of the second line and thus introduce a new set statement. These two alternatives are illustrated below:

```
Sets  i   "canning plants"   / Seattle, San-Diego /
      j   "markets"          / New-York, Chicago, Topeka / ;
```

or

```
Sets  i   "canning plants"   / Seattle, San-Diego / ;
Set   j   "markets"          / New-York, Chicago, Topeka / ;
```

Note

- In general, GAMS statements have to be terminated with a semicolon. The semicolon may be omitted if the next line starts with a GAMS keyword.
- Even if it is not required, it is good practice to always end a statement with a semicolon.

4.14.2.2 Error B: Missing Semicolons

Consider the following example adapted from the well-known transportation model [**TRNSPORT**]:

```
Equations
       cost        "define objective function"
       supply(i)   "observe supply limit at plant i"
       demand(j)   "satisfy demand at market j"

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
```

The resulting compilation output will contain the following lines:

```
   52    Equations
   53         cost        define objective function
   54         supply(i)   observe supply limit at plant i
   55         demand(j)   satisfy demand at market j
   56
   57    cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
****         $97                     $195,96,195,96 $195,96
...
Error Messages

 96  Blank needed between identifier and text
       (-or- illegal character in identifier)
       (-or- check for missing ';' on previous line)
 97  Explanatory text can not start with '$', '=', or '..'
       (-or- check for missing ';' on previous line)
```

What went wrong? GAMS statements like the equation declaration in our example have to terminate with a semicolon, unless the next line of code begins with a GAMS keyword. In our example, we omitted the semicolon at the end of the equation declaration statement and started a new statement after a blank line.

How do we fix this? We simply add a semicolon after the last explanatory text at the end of the equation declaration statement.

Note

A missing semicolon is often associated with error 96 or 97.

4.14.2.3 Error C: Spelling Mistakes

This error occurs if sets, set elements, parameters, etc. are referenced with a different name than the name they were declared with. Note that differences in capitalization are not considered spelling mistakes since GAMS is case insensitive.

Consider the following example adapted from the well-known transportation model [**TRNSPORT**]:

```
Sets i   "canning plants"   / Seattle, San-Diego /
     j   "markets"          / New-York, Chicago, Topeka / ;

Parameters a(i)  "capacity of plant i in cases"
          /   Seatle      350
              san-diego   600  / ;
...
Equations
       cost        "define objective function"
       supply(i)   "observe supply limit at plant i"
       demand(j)   "satisfy demand at market j" ;

cst ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
```

The resulting compilation output will contain the following lines:

```
   20    Sets i   canning plants   / Seattle, San-Diego /
   21         j   markets          / New-York, Chicago, Topeka / ;
   22
   23    Parameters a(i)  capacity of plant i in cases
   24             /   Seatle      350
****                     $170
   25                san-diego   600  / ;
...
   49    Equations
   50         cost        define objective function
   51         supply(i)   observe supply limit at plant i
   52         demand(j)   satisfy demand at market j ;
   53
   54    cst ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
****      $140
...
Error Messages

140  Unknown symbol
...
170  Domain violation for element
```

What went wrong? Note that in line 24 the set element `Seattle` was referenced as `Seatle`. GAMS does not recognize this symbol as an element of the set `i` and thus marks this with error `170`: `Domain violation`. GAMS catches this error since it automatically performs domain checking. The typo in line 54, where the equation `cost` is referenced as `cst`, is marked with error `140` (`Unknown symbol`), since `cst` was *not* declared before.

Note

Set name misspellings are associated with error `120`, set element misspellings with `170` and other misspellings with `140`.

Fixing errors like these is as easy as fixing typos.

4.14.2.4 Error D: Missing Set Elements

Sometimes users forget to include an element in a set statement, but reference it later. Consider the following simple example:

```
Set       c    "crops"                 / barley, wheat, soy /;
Parameter d(c) "demand in metric tons" / rice 3, barley 1, wheat 4, soy 2 /;
```

The resulting compilation output will contain the following lines:

```
   1  Set            c    "crops"                 / barley, wheat, soy /;
   2  Parameter      d(c) "demand in metric tons" / rice 3, barley 1, wheat 4, soy 2 /;
****                                                $170
...
Error Messages

170  Domain violation for element
```

What went wrong? Note that the symbol `rice` was not defined as an element of the set `c`, thus GAMS does not recognize it and marks it with error `170`: `Domain violation`. GAMS catches this error since it automatically performs domain checking.

Note

If symbols are not defined as set elements, but are referenced as if they belong to the set later, error `170` will occur.

This is easy to fix: we just add the missing element(s) to the elements of the respective set.

4.14.2.5 Error E: Problems with Indices

Recall that variables, sets, parameters and equation may be defined over one or more indices. If the identifiers are referenced later in the program, the indices must appear exactly in the order that was specified in the respective definition statement. Consider the following example adapted from the well-known transportation model [**TRNSPORT**]:

```
Sets i   "canning plants"   / seattle, san-diego /
     j   "markets"          / new-york, chicago, topeka / ;

Table d(i,j)  "distance in thousands of miles"
                    new-york       chicago      topeka
      seattle          2.5           1.7          1.8
      san-diego        2.5           1.8          1.4  ;

Scalar    f       "freight in dollars per case per thousand miles"  /90/ ;
Parameter c(i,j)  "transport cost in thousands of dollars per case" ;
c(i,j,j) = f * d(j,i) / 1000 ;
```

The resulting compilation output will contain the following lines:

```
   40    Parameter c(i,j)  transport cost in thousands of dollars per case ;
   41    c(i,j,j) = f * d(j,i) / 1000 ;
****           $148       $171,171
...
Error Messages

148  Dimension different - The symbol is referenced with more/less
        indices as declared
171  Domain violation for set
```

What went wrong? Note that the parameter `c` was declared over two indices, but in the assignment statement in line 41 it is referenced with three indices. Such a mistake is marked with error `148`. Note further, that the indices of `d` are referenced in the wrong order, which is marked with error `171`.

Note

If an identifier is referenced with more or less indices as declared, then the error `148` will be triggered. If the indices are in the wrong order, the error `171` will be triggered.

This is easy to fix: we just check the declaration or definition statement(s) and adjust the reference.

Observe that the domain error `171` is also triggered if an identifier is referenced with index say `i`, but was defined over say `j`. However, there will be no domain error if `i` is a subset of `j` or `i` and `j` reference the same set since they are aliases.

4.14.2.6 Error F: Summing over Sets that are Already Indexed

Consider the following equation definition that is adapted from the well-known transportation model [**TRNSPORT**]:

```
supply(i) ..   sum((i,j), x(i,j))  =l=  a(i) ;
```

The resulting compilation output will contain the following lines:

```
   59    supply(i) ..   sum((i,j), x(i,j))  =l=  a(i) ;
****                          $125
...
Error Messages

125  Set is under control already
```

What went wrong? Note that the equation is indexed over the set `i`, therefore the indexed operation `sum` in the body of the equation may not be controlled by the index `i` again.

Note

Summing over sets that are already indexed will trigger error `125`.

How do we fix this? We need to carefully check the indexed operation and drop the surplus index. In other cases the controlling index may have to be dropped from the equation name. Note that an error like this is often indicative of a lack of clarity in thinking.

4.14.2.7 Error G: Uncontrolled Sets

Consider the following equation definition that is adapted from the well-known transportation model [**TRNSPORT**]:

```
demand ..   sum(i, x(i,j))  =g=  b(j) ;
```

The resulting compilation output will contain the following lines:

```
   61    demand ..   sum(i, x(i,j))  =g=  b(j) ;
****                             $149        $149
...
Error Messages

149  Uncontrolled set entered as constant
```

What went wrong? Note that the variable `x` is indexed over the sets `i` and `j`, but, while `i` is the index of the indexed operation `sum`, `j` is "free": it does neither appear as index of the equation nor as controlling index in `sum`. Therefore it was entered as if it were a constant. Note further, that `j` is also used as an index on the right-hand side of the equation without being controlled.

Note

Error `149` marks instances when an index is not controlled in the context of an equation or an indexed operation like `sum` or `prod`.

How do we fix this? We have to think what we actually want to model. In this case we need to enter `j` as a controlling index for the equation. But it would also be possible to add it as a controlling index for `sum` (but not both!). In the latter case there would still be the error on the right-hand side which needed to be taken care of with another indexed operation, e.g., `sum` again. Both approaches would remove the compilation error, but one has to think about what actually should be modeled to decide which is the right fix here.

4.14.2.8 Error H: Mismatched Parentheses

Consider the following two equations adapted from the model [**CHENERY**]:

```
mb(i).. x(i) =g= y(i) + sum(j, aio(i,j)*x(j)) + ( e(i) - m(i) ))$t(i) ;
...
dvv(i)$(sig(i) <> 0)..  vv(i) =e= (pi*(1-del(i))/del(i))**(-rho(i)/(1+rho(i)) ;
```

The resulting compilation output will contain the following lines:

```
   137   mb(i).. x(i) =g= y(i) + sum(j, aio(i,j)*x(j)) + ( e(i) - m(i) ))$t(i) ;
****                                                                    $408,409
...
   155   dvv(i)$(sig(i) <> 0)..  vv(i) =e= (pi*(1-del(i))/del(i))**(-rho(i)/(1+rho(i)) ;
****                                                                           $8
...
Error Messages

  8  ')' expected
...
408  Too many ),] or }
409  Unrecognizable item - skip to find a new statement
       looking for a ';' or a key word to get started again
```

What went wrong? Note that in the first equation we have one surplus closing parenthesis, this is marked with error `408`. The error marked with `409` is a consequential error, it will disappear once the first error has been fixed. In the second equation, one closing parenthesis is missing resulting in error `8`.

Attention

Opening and closing parentheses `( )`, square brackets `[ ]` and braces (curly brackets) { } must match.

Note

Surplus closing parentheses, bracketes and braces are marked with error `408` and missing closing parenthesis, brackets and braces are marked with error `8`. *Missing opening* parentheses, brackets and braces may result in GAMS marking *surplus closing* parentheses, brackets and braces; *surplus opening* parentheses, brackets and braces may result in GAMS marking *missing closing* parentheses, brackets and braces.

While fixing errors like these entails carefully counting opening and closing parentheses, there are strategies that help to prevent mismatching parentheses errors. Many text editors (like GAMS Studio) offer a feature that identifies matching parentheses and will issue a warning if there is a mismatch. We recommend to use this feature. Further, we recommend to also use the alternatives to parentheses: square brackets and braces. They are especially useful if there are several opening parentheses since it is easier to determine by inspection if each has a matching closing symbol.

4.14.2.9 Error I: Mistakes Relating to Equations

Recall that each declared equation must be defined if it is to be used in a model statement. If the equation definition is missing, GAMS will mark an error beneath the solve statement that refers to a model which references the respective undefined equation. Consider the following example adapted from the well-known transportation model [**TRNSPORT**]:

```
Equations
       cost        "define objective function"
       supply(i)   "observe supply limit at plant i"
       demand(j)   "satisfy demand at market j" ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;

Model transport /all/ ;
solve transport using lp minimizing z ;
```

The resulting compilation output will contain the following lines:

```
   60    Model transport /all/ ;
   61    solve transport using lp minimizing z ;
****                                          $71,256
**** The following LP errors were detected in model transport:
****  71 demand is an undefined equation
...
Error Messages

 71  The symbol shown has been declared as an equation, but no
       Symbolic equation (..) was found. hint - look for commas in the
       Documentation text for the equations. use quotes around the
       Text or eliminate the commas.
256  Error(s) in analyzing solve statement. More detail appears
     Below the solve statement above
```

What went wrong? Note that the equation `demand` was declared and included in the model `transport`, but it was not defined. GAMS detects that the equation definition is missing when it compiles the solve statement that relates to the model `transport`. Therefore the error `71` appears only there. The error `256` is a consequential error, it will disappear once the missing equation definition has been added (*before* the solve statement).

Note

Error `71` marks a missing equation definition. The message associated with this error is a nice example of an error message that is very descriptive and offers advice on how to fix the error.

Apart from forgetting to define a declared equation and forgetting to terminate the equation definition statement with a semicolon (see above), two other mistakes relating to equations are frequent: omitting the syntax element `..` after the name of the equation and not properly defining the equation type in the equation definition statement. The following simple example adapted from the portfolio analysis model [**ALAN**] illustrates these mistakes:

```
fsum       sum(i, x(i))               =e=  1.0;
dmean..    sum(i, mean(i)*x(i))        =   target;
```

The resulting compilation output will contain the following lines:

```
   45  fsum     sum(i, x(i))                 =e= 1.0;
****             $36
   46  dmean..    sum(i, mean(i)*x(i))        =  target;
****                                        $37
...
Error Messages

36  '=' or '..' or ':=' or '$=' operator expected
     rest of statement ignored
37  '=l=' or '=e=' or '=g=' operator expected
```

What went wrong? Note that in the first equation the two dots `..` are missing, which is marked with error `36`, and in the second equation the equation type is not properly specified, which leads to error `37`.

4.14.2.10 Error J: Issues with Nonlinear Expressions

The compilation error messages `51` to `60` refer to issues with nonlinear expressions involving variables in equations. For example, nonlinear expressions in an LP model will cause errors of this sort. So will discontinuous functions with endogenous arguments in an NLP model. For information on endogenous arguments in functions, see section Functions in Equation Definitions. For an overview of GAMS model types, see section Classification of Models.

Consider the following simple artificial example:

```
Variables x, y, z ;
Equations eq1, eq2 ;

eq1.. x**2 - y =e= z ;
eq2.. min(x,y) =l= 20 ;

Model silly / all / ;
solve silly using lp maximizing z ;
```

The resulting compilation output will contain the following lines:

```
   7  Model silly / all / ;
   8  solve silly using lp maximizing z ;
****                                   $54,51,256
**** The following LP errors were detected in model silly:
****  54 equation eq1.. VAR operands for **
****  51 equation eq2.. the function MIN is called with non-constant arguments

Error Messages

51  Endogenous function argument(s) not allowed in linear models
54  Endogenous operands for ** not allowed in linear models
256  Error(s) in analyzing solve statement. More detail appears
     Below the solve statement above
```

Note that error `54` marks the nonlinear expression x^2 in a linear model and error `51` refers to the discontinuous function `min` that may be used with variables only in models of the type DNLP. The error `256` is a consequence of the two previous errors.

4.14.2.11 Error K: Using Undefined Data

Referencing data items that were not declared with a scalar, parameter or table statement will cause error 140: GAMS will indicate that the respective symbol is unknown. More often data items are declared, but not defined, i.e. they are not initialized with values.

Consider the following example adapted from the well-known transportation model [**TRNSPORT**]. Assume that the scalar f is declared, but has no numerical value. The resulting compilation output will contain the following lines:

```
   40    Scalar f  freight in dollars per case per thousand miles  ;
   41
   42    Parameter c(i,j)  transport cost in thousands of dollars per case ;
   43
   44              c(i,j) = f * d(i,j) / 1000 ;
****                      $141
...
   61    Model transport /all/ ;
   62    solve transport using lp minimizing z ;
****                                          $257
   63
   64    display x.l, x.m ;
****              $141
...
Error Messages

141  Symbol declared but no values have been assigned. Check for missing
        data definition, assignment, data loading or implicit assignment
        via a solve statement.
        A wild shot: You may have spurious commas in the explanatory
        text of a declaration. Check symbol reference list.
257  Solve statement not checked because of previous errors
```

What went wrong? Note that the scalar f was declared, but there was no value assigned to it. Thus, when it is referenced in the calculation in line 44, the "empty" scalar causes error 141. Error 257 is a consequential error and leads to the second error 141. This error occurs in the context of variable attributes in the display statement. Recall that variable attributes are data associated with variables. As the solve statement could not be checked, the level and marginal values of the variable x are not defined and hence error 141 follows.

Note

> Error 141 marks places where data items, that have not been defined, are referenced in computations in the context of assignment statements. It also marks places where undefined variable (or equation) attributes are referenced.

In addition, undefined data items may be referenced in the context of equation definitions. Suppose we declare the parameter c in model [**TRNSPORT**], but forget the corresponding assignment statement. Then the compilation output will contain the following lines:

```
   57    cost ..        z  =e=  sum((i,j), c(i,j)* x(i,j)) ;
 ...
   62    solve transport using lp minimizing z ;
****                                          $66,256
**** The following LP errors were detected in model transport:
```

```
****  66 equation cost.. symbol "c" has no values assigned
...
Error Messages

 66  The symbol shown has not been defined or assigned
        A wild shot: You may have spurious commas in the explanatory
        text of a declaration. Check symbol reference list.
256  Error(s) in analyzing solve statement. More detail appears
     Below the solve statement above
```

Note that the missing data in the body of the equation `cost` is marked with error `66` and the following error is a consequential error which will disappear once `66` has been fixed.

Note

Referencing undefined data items in the body of an equation causes error `66`.

4.14.2.12 Error L: Referencing Set Elements Without Quotes

Recall that if set elements are referenced, they need to appear in single or double quotes. Omitting the quotes may cause different errors depending on the context. The following example is adapted from the well-known transportation model [**TRNSPORT**]. Suppose we start by defining the two sets and the two parameters and then add the following statements:

```
Scalar s;
s = a(seattle);
```

The resulting compilation output will contain the following lines:

```
   37  Scalar s;
   38  s = a(seattle);
****              $120,340,149,171
...
Error Messages

120  Unknown identifier entered as set
149  Uncontrolled set entered as constant
171  Domain violation for set
340  A label/element with the same name exist. You may have forgotten
        to quote a label/element reference. For example,
        set i / a,b,c /; parameter x(i); x('a') = 10;
```

What went wrong? On the right-hand side of the assignment statement we have a reference with missing quotes. Note that this creates four (!) errors:

- `120` - GAMS expects a controlling set and `Seattle` is not recognized and therefore considered an unknown identifier.
- `340` - Then GAMS realizes that actually there is a label called `Seattle` in the program and helpfully offers a hint: `You may have forgotten to quote a label/element reference.`
- `149` - See section Error G: Uncontrolled Sets above.
- `171` - See section Error E: Problems with Indices above.

This is a nice example for one mistake triggering several errors that may look daunting. However, fixing them is as easy as adding single or double quotes.

4.14.2.13 Error M: Missing Declaration Statements

Recall that sets, parameters, variables and equations have to be declared before they may be used in definition or assignment statements. Suppose the equation `demand` in the well-known transportation model [**TRNSPORT**] was defined but not declared. Then the compilation output would contain the following lines:

```
   59    demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;
****         $140
...
Error Messages

140  Unknown symbol
```

Note that as `demand` was not declared, GAMS does not recognize it and therefore marks it with error `140`: `Unknown symbol`.

Note

If error `140` is reported, the respective declaration statement is probably missing. However, the underlying cause may also be a simple typo.

4.14.2.14 Error N: Using the Same Name for Different Identifiers

Recall that each GAMS identifier must have a unique name. If the same name is used more than once, an error will be triggered. Consider the following modification of the transportation model [**TRNSPORT**]:

```
Parameters  capacity(i)  "capacity of plant i in cases"
               / Seattle 350, San-Diego 600 /

            demand(j)    "demand at market j in cases"
               / New-York 325, Chicago 300, Topeka 275 /;
...
Equations
       cost        "define objective function"
       supply(i)   "observe supply limit at plant i"
       demand(j)   "satisfy demand at market j" ;
```

The resulting compilation output will contain the following lines:

```
   47    Equations
   48         cost        define objective function
   49         supply(i)   observe supply limit at plant i
   50         demand(j)   satisfy demand at market j ;
****             $195
...
Error Messages

195  Symbol redefined with a different type
```

What went wrong? Note that `demand` was first declared as a parameter and later as an equation. GAMS will mark this repeated use of the same name with error `195`.

4.14.2.15 Error O: Using ORD with an Unordered Set

Recall that the operation ord is only defined for ordered sets. If `ord` is used with an unordered set, error `198` will be triggered.

Consider the following example:

```
Set    a   "a couple of the elements" / r2, r3 /;
Set    b   "more elements"            / r1*r4 /;
Scalar c   "counter"                  / 0 /;

loop( b $ (ord(b) > 3),
    c = c + 1 );
display c;
```

The resulting compilation output will contain the following lines:

```
   5  loop( b $ (ord(b) > 3),
****                $198
   6      c = c + 1 );
...
Error Messages

198  Set used in 'ord' or lag is not ordered.
        Hint: Some of the elements of the set were used before this
        was initialized and the order was different from the order used
        in this set. Try to initialize the set earlier.
        $offOrder allows lag operations on dynamic sets, reset with
        $onOrder
```

What went wrong? Two elements of the set `b` featured already in set `a` that was defind before set `b`. Therefore the order of the elements in set `b` is as follows:

```
r2 r3 r1 r4
```

Obviously, this set is not orderd and hence GAMS marks error `198` when the operator `ord` is applied to it.

How do we fix this? In this case it is easy: we just define the set `b` before set `a` and `b` will be an ordered set.

Consider one last example:

```
Set    a   "all elements"                / r1*r10 /;
Set    b   "elements in different order" / r3, r1, r4, r7 /;
Scalar c   "counter"                     / 0 /;

loop( b $ (ord(b) > 3),
    c = c +1 );
display c;
```

The resulting compilation output will contain the following lines:

```
   5  loop( b $ (ord(b) > 3),
****                 $198
   6      c = c +1 );
```

What went wrong? Note that even though all elements in set `b` are also elements of set `a`, they are not specified in the same order. Therefore the set `b` is unordered. For more details, see section Ordered and Unordered Sets.

4.15 Finding and Fixing Execution Errors and Performance Problems

This tutorial is structured in four main parts: we will discuss how to find and resolve errors that are detected during execution of a GAMS model, we will give some guidance for model development and debugging and we will present techniques to increase efficiency by reducing GAMS execution time and memory use.

4.15.1 Resolving Execution Errors

Recall that GAMS passes through a program file several times in the process of generating and solving a model. Errors may occur in each phase. In this section we will give some guidance on how to resolve errors that occur during execution, so after compilation. For advice on resolving compilation errors, see the tutorial Fixing Compilation Errors. For more information on the process of generating and solving a model in GAMS, see the introduction to chapter GAMS Output.

At execution, several things could go wrong and cause an error. We will look at these potential error sources separately in this section. First we look at arithmetic errors and exceeded internal limits during data manipulation, we will continue with problems during model generation and model solution. At the end, we will briefly discuss how execution errors may be managed with the function execError.

4.15.1.1 Arithmetic Errors

GAMS execution errors may be caused by illegal arithmetic operations like a negative argument for `log`, division by zero and exponentiation where the base is a negative number. The following simple example serves as illustration:

```
Set       s     / s1*s5 / ;
Parameter p(s)  "data to be exponentiated"
          d(s)  "divisors"
          r(s)  "result";

p(s)    = 1;
p("s2") = -1;
d(s)    = 1;
d("s3") = 0;
r(s)    = p(s)**2.1 / d(s)
display r;
```

The first sign that something in the execution went wrong is the following flag in the log output:

```
*** Status: Execution error(s)
```

The resulting execution output will contain the following lines:

```
E x e c u t i o n

**** Exec Error at line 10: rPower: FUNC DOMAIN: x**y, x < 0
**** Exec Error at line 10: division by zero (0)

----     11 PARAMETER r  result

s1 1.000,    s2  UNDF,    s3  UNDF,    s4 1.000,    s5 1.000
```

Observe that the execution output begins with two error messages that can be easily found since they are marked with four asterisks ****. The error messages are very informative: they indicate the line where the errors occurred and provide details about the nature of the errors. Further, the output generated by the display statement shows that the errors occurred when the values for `r("s2")` and `r("s3")` were computed. Inspecting the assignment statement for these two values, we realize that in the first instance the base for the exponentiation is -1, which obviously is a negative number and hence is not allowed in this operation. In the second instance, the problem is that we divide by `d("s3")` which equals zero.

In this example, the errors are easily resolved with data revisions. In general, we recommend to use conditional assignments to prevent errors like these.

Note that in the example above the error messages indicated exactly where the problem was and it was easy to find the cause of the error. However, this is not always the case. In particular, if the problem is within a multi-dimensional item the user will need more patience. Usually it helps to display the results of the problematic operation and look for faulty entries. In addition, displaying the input data to the respective operation will help to investigate the numerical properties of the data that was entered in the computation. Often more displays will be needed to trace faulty input data through the program. Eventually this will lead the user to understand why the data has taken on the specific numerical values it has.

4.15.1.2 Exceeding GAMS Limits

By default, GAMS stops the solve of a model after 1e10 seconds (wall clock time) or 2e9 iterations. These limits may be adjusted with the options reslim and iterlim respectively. Note that both options are also available as command line parameters and model attributes. In addition, the workspace may be limited with the command line parameters WorkFactor and WorkSpace. Note that these options are also available as model attributes. If any of these limits are exceeded, the execution of the solve statement will be interrupted.

For example, we could add the following option statement somewhere *before* the solve statement in the production model [**CHENERY**]:

```
option iterlim = 20;
```

Note that this statement reduces the iteration limit to just 20. The log output will contain the following lines:

```
 ** Feasible solution. Value of objective =    1033.34069261

 ** The iteration limit has been reached.

--- Restarting execution
--- chenery.gms(228) 2 Mb
--- Reading solution for model chenrad
*** Status: Normal completion
--- Job chenery.gms Stop 11/21/16 16:52:43 elapsed 0:00:00.106
```

Also the solve summary in the listing file notes the interrupt:

```
               S O L V E      S U M M A R Y

     MODEL   chenrad             OBJECTIVE  td
     TYPE    NLP                 DIRECTION  MAXIMIZE
     SOLVER  CONOPT              FROM LINE  228

**** SOLVER STATUS     2 Iteration Interrupt
**** MODEL STATUS      7 Feasible Solution
**** OBJECTIVE VALUE             1058.9199

 RESOURCE USAGE, LIMIT          0.078      1000.000
 ITERATION COUNT, LIMIT        20            20
 EVALUATION ERRORS              0             0
```

Observe that the solver status `Iteration Interrupt` indicates that the execution terminated because the iteration limit has been reached resulting in a feasible solution, but not the optimal solution. The line `ITERATION COUNT, LIMIT ...` reports that 20 iterations were performed and that 20 was also the limit for the number of iterations. Setting `iterlim` to a larger value will resolve this issue.

Similarly, allocating too little work space will cause the solver to terminate with no solution. For example, restricting the work space for the nonlinear test model [**MHW4D**] to just 0.1 MB and running it with the solver MINOS will produce the following lines in the log output:

```
 Work space requested by solver --     0.77 Mb
 Work space requested by user   --     0.10 Mb
 Work space allocated           --     0.10 Mb

  Reading Rows...
  Reading Columns...
  Reading Instructions...


 EXIT - Not enough storage to solve the model.
        Request at least    0.19 Mbytes.
```

The solve summary in the listing file will contain the following information:

```
               S O L V E      S U M M A R Y

     MODEL   wright              OBJECTIVE  m
     TYPE    NLP                 DIRECTION  MINIMIZE
     SOLVER  MINOS               FROM LINE  32

**** SOLVER STATUS     9 Setup Failure
**** MODEL STATUS      13 Error No Solution
```

Note that increasing the work space to at least the minimum amount requested by the solver will resolve this issue.

When dealing with large nonlinear expressions defined over a very large domain, one can face the following error.

```
*** Status: Terminated due to limits in NLCodeAdd
***         Cannot handle more than 2147483647 instruction in NL code
***         Inspect listing file for more information
```

The first thing to check in this case is correctness of the model i.e., if you are not generating anything more than necessary. If this is not the case and the model is indeed large, a generic advise is to introduce intermediate variables to get a smaller code size per block of equations. One can also consider partitioning the domains so that one can have more number of smaller blocks.

4.15.1.3 Resolving Model Generation Errors

Further execution errors may be detected when GAMS is generating the model before passing it to the solver. These errors may be arithmetic errors in the *body* of equations or errors in the structure of the model that cause the model to be inherently infeasible.

Consider the following simple example with arithmetic errors in the body of the equations. They are similar to the errors in the assignment in the example in section Arithmetic Errors above.

```
Set       s     / s1*s5 / ;
Parameter p(s)  "data to be exponentiated"
          d(s)  "divisors"
          m(s)  "multipliers";

p(s)    = 1;
p("s2") = -1;
d(s)    = 1;
d("s3") = 0;
m(s)    = 1;
m("s4") = 0;

Positive variable x(s);
Variable          z;

Equations obj  "objective function"
          xlim;

obj..      z =e= sum(s,p(s)**2.2*x(s));
xlim(s)..  m(s) / d(s)*x(s) =e= 1;

Model mymodel / all /;
solve mymodel using lp maximizing z;
```

If we run this model, the log output will contain the followng lines:

```
*** SOLVE aborted
--- Executing CPLEX: elapsed 0:00:00.006
--- test.gms(23) 4 Mb 3 Errors
*** Status: Execution error(s)
--- Job test.gms Stop 11/21/16 19:10:12 elapsed 0:00:00.006
```

Observe that the solve was aborted since there are 3 execution errors. The equation listing in the listing file will contain further details about these execution errors:

```
Equation Listing    SOLVE mymodel Using LP From line 23

**** Exec Error at line 19: rPower: FUNC DOMAIN: x**y, x < 0

---- obj  =E=  objective function

obj..  - x(s1) + UNDF*x(s2) - x(s3) - x(s4) - x(s5) + z =E= UNDF ; (LHS = UNDF)

**** Exec Error at line 20: division by zero (0)
**** Exec Error at line 20: Equation infeasible due to rhs value
```

```
**** INFEASIBLE EQUATIONS ...

---- xlim  =E=

xlim(s4)..  0 =E= 1 ; (LHS = 0, INFES = 1 ****)

REMAINING 4 ENTRIES SKIPPED
```

Note that there is an arithmetic error relating to exponentiation in the first equation and an arithmetic error and an infeasibility in the second equation.

In our example, it was easy to detect the execution errors and their cause. However, an error in a multi-dimensional equation block may be much more difficult to find. Note that by default, only the first three entries in each equation block are shown in the equation listing. We recommend to use the option limrow to get a full listing, as this is the easiest way to inspect execution errors in the body of equations.

4.15.1.4 Resolving Solve Errors

In the solution phase, an external solver program processes the model and creates output with details about the solution process. Solve errors may be either function evaluation errors or presolve errors.

Resolving Function Evaluation Errors

Some solve statements require the evaluation of nonlinear functions and the computation of derivatives. Since these calculations are not carried out by GAMS but by other subsystems not under the direct control of GAMS, errors associated with these calculations are reported in the solution report.

Function evaluation errors are numerical errors like those discussed in section Arithmetic Errors above. Other examples include square roots of negative variables and squaring a negative term, say `x`, using the syntax `x**2`.

Attention

> Squaring a negative term, say `x`, using the syntax `x**2` will cause an error. However, the alternatives `sqr(x)` and `x*x` will work (see here for an explanation).

Note that by default the solver subsystems will interrupt the solution process if arithmetic errors are encountered. Users may wish to permit a certain number of arithmetic errors and have reported *error warnings* instead. The option domlim facilitates this modification. Note that the default value for `domlim` is zero.

The best way to avoid evaluating functions outside their domain of definition is to specify reasonable variable bounds. However, there are cases when bounds are not enough. Consider the following simple example:

```
Set       i  / i1*i5 /;
Variables x(i), z;
Equations r1, r2(i);

r1..     z =e= log(sum(i, x(i)));
r2(i)..  x(i) =l= 10;
x.lo(i) = 0;
x.l(i)  = 5;

Model takelog / all /;
solve takelog using nlp minimizing z;
```

If we try to solve this little program with the solver MINOS, the log output will contain the following line:

```
EXIT - Function evaluation error limit exceeded.
```

The solution report in the listing file will have more detailed information:

```
               S O L V E      S U M M A R Y

     MODEL   takelog             OBJECTIVE  z
     TYPE    NLP                 DIRECTION  MINIMIZE
     SOLVER  MINOS               FROM LINE  12

**** SOLVER STATUS     5 Evaluation Interrupt
**** MODEL STATUS      7 Feasible Solution
**** OBJECTIVE VALUE                0.0000

 RESOURCE USAGE, LIMIT          0.183      1000.000
 ITERATION COUNT, LIMIT         0    2000000000
 EVALUATION ERRORS              2             0

...

 EXIT - Function evaluation error limit exceeded.

**** ERRORS/WARNINGS IN EQUATION r1
     2 error(s): log: FUNC DOMAIN: x < 0 (RETURNED 0)

...

**** REPORT SUMMARY :         1     NONOPT ( NOPT)
                             0 INFEASIBLE
                             0  UNBOUNDED
                             1     ERRORS ( ****)
```

Note that the solver status has a value of 5 (`Evaluation Interrupt`), which means that the solver has been interrupted as more evaluation errors have been encountered than specified with the option domlim. In our case `domlim` equals its default value zero, thus one error is enough to cause the interruption. The equation in which the evaluation error occurred and the type of error is reported a few lines later. In our example, the equation `r1` is problematic, since we take the logarithm of the expression `sum(i, x(i))`, an expression which may become zero.

Note that in models such as this each individual variable `x(i)` should be allowed to become zero, but the sum should not. This may be achieved by introducing an intermediate variable, say `xsum`, adding a lower bound greater than zero for it and using this variable as the argument for the function `log`:

```
Variable xsum;
xsum.lo = 0.0001;

Equations defxsum, r1;

defxsum ..    xsum =e= sum(i, x(i));
r1..          z =e= log(xsum);
```

For more information on intermediate variables, see section Avoiding Expressions in Nonlinear Functions in the tutorial Good NLP Formulations.

Observe that solvers report the type of arithmetic problem encountered and the problematic equation, but do not identify the particular offending variable or the labels in the index of an equation that cause the error. If the cause is not obvious, users will have to investigate the numerical properties of the variables, labels and parameters in the body of the respective equation. This may involve the following:

- Displaying the input data items to the nonlinear terms in the respective equation.
- Searching the solution for equations that are infeasible (`INFES`) and variables that are nonoptimal (`NOPT`) in order to see where problems are present and which variables were being manipulated at the end of the run.
- Investigating variables and equations whose level values are zero, negative or very large at the end of the run.
- Deactivating part of the code to narrow down the problem as discussed in section Isolating Terms in Slow Statements below.

Resolving function evaluation errors will usually entail the following techniques:

- Adding lower bounds to variables to keep them above zero.
- Adding upper bounds to variables to prevent them from getting too large.
- Reformulating the model, for example, introducing intermediate variables.
- Providing better starting points that direct the solver search to a more relevant region. See section Specifying Initial Values in tutorial Good NLP Formulations for details.
- Fixing faulty input data.

Presolve Errors

Some solvers use a pre-processing step where the program is presolved to make the main solution process faster. During this step model errors could already be discovered, as in the following example:

```
Variables         z;
Integer Variables y1,y2;

Equations         r1,r2,r3,r4;

   r1..  z=e=y1+y2;
   r2..  y1=g=0.10;
   r3..  y2=g=0.10;
   r4..  y1+y2=l=1;

Model badpresol /all/;
solve badpresol using mip maximizing z;
```

For this problem, Cplex detects in the presolve already, that there is no feasible integer solution. This is reported in the log:

```
Row 'r4' infeasible, all entries at implied bounds.
Presolve time = 0.00 sec. (0.00 ticks)

...

CPLEX Error  1217: No solution exists.
Problem is integer infeasible.
```

Here, Cplex makes it clear, where we have a problem: Row `r4` is infeasible, because all entries are at their "implied bounds". Lets look at `r2` and `r3` to see what this means: These equations set a lower bound of 0.1 for `y1` and `y2`. Since both variables are defined as `Integer Variables`, they get an implicit lower bound of 1. Given that, equation r4 must be infeasible.

Solver Specific Limits

Many solvers have internal limits that may be exceeded and may cause the listing file to report an execution error. These errors may be resolved by using either GAMS options or solver specific options to increase the respective limits. Usually, the listing file will contain information about which options to use. Note that the solver manuals distributed with GAMS list the options that may be specified for each solver. For example, to relax the MINOS major iteration limit, the user may create a file named `minos.opt` with the following line:

```
Major iterations 1000
```

More about solver option files can be found in section The Solver Options File.

4.15.1.5 Managing Execution Errors with the Function execError

The function execError facilitates implementing procedures that manage execution errors. Consider the following example, which is an extension of the example in section Arithmetic Errors above.

```
Set       s     / s1*s5 / ;
Parameter p(s)  "data to be exponentiated"
          d(s)  "divisors"
          r(s)  "result";

p(s)    = 1;
p("s2") = -1;
d(s)    = 1;
d("s3") = 0;
r(s)    = p(s)**2.1 / d(s)
display r;

*cause z to be undefined
Scalar z;
z = 1/0;

if(execError > 0,
   r(s)$(r(s) = z) = 0;);
display r;
```

Observe that we introduced a new scalar `z` that is deliberately undefined. In the if statement that follows, we use the function `execError` in the logical condition and the undefined scalar in the conditional assignment. The if statement has the effect that undefined entries are removed from the array of the parameter `r`, as illustrated in the following lines of the execution output:

```
E x e c u t i o n

**** Exec Error at line 10: rPower: FUNC DOMAIN: x**y, x < 0
**** Exec Error at line 10: division by zero (0)

----     11 PARAMETER r  result

s1 1.000,    s2  UNDF,    s3  UNDF,    s4 1.000,    s5 1.000

**** Exec Error at line 16: division by zero (0)

----     20 PARAMETER r  result

s1 1.000,    s4 1.000,    s5 1.000
```

In addition, the function execError may be used to reset the count of the number of execution errors. Typically, it is reset to zero so that GAMS will terminate with the status message `Normal completion`. For example, we could add the following line at the end of the code in the example above:

```
execError = 0;
```

Note

Setting `execError = 0;` will not only result in a `normal completion` in the example above. A solve statement will not be executed if there were execution errors before by default. Setting `execError = 0;` before the solve statement, will allow to execute it again.

Setting `execError = 0;` results also in a notification in the log:

```
********************************
*** Errors have been cleared ***
********************************
*** Status: Normal completion
```

4.15.2 Small to Large: Aid in Development and Debugging

Many GAMS users are overly impressed with how easily GAMS handles large models. Modelers often feel such a facility means they should always work on the full model. The result is often a large, sometimes extremely large, model in the early stages of model development. Debugging such large formulations is not easy.

The algebraic modeling style employed in GAMS is inherently expandable. This offers interesting possibilities in terms of the strategy that may be employed for model development and debugging which are discussed herein.

4.15.2.1 An Illustrative Example

The set based algebraic modeling style implemented in GAMS is by its very nature easy to expand. It is easy to use the same model formulation on differently sized data sets. We will illustrate this based on the transportation model [**TRNSPORT**]. Note that we included some post-solution calculations at the end.

```
* Data section
Sets  i   "canning plants"   / Seattle, San-Diego /
      j   "markets"          / New-York, Chicago, Topeka / ;

Parameters  a(i)  "capacity of plant i in cases"
               / Seattle   350, San-Diego  600/

            b(j)  "demand at market j in cases"
               / New-York 325, Chicago 300, Topeka 275 /;

Table       d(i,j)  "distance in thousands of miles"

                          New-York       Chicago      Topeka
           Seattle          2.5            1.7         1.8
           San-Diego        2.5            1.8         1.4  ;

Scalar f  "freight in dollars per case per thousand miles" /90/ ;

Parameter   c(i,j)  "transport cost in thousands of dollars per case" ;
c(i,j) = f * d(i,j) / 1000 ;

* Model Section
Positive Variable  x(i,j)  "shipment quantities in cases";
Variable           z       "total transportation costs in thousands of dollars";

Equations cost        "define objective function"
          supply(i)   "observe supply limit at plant i"
          demand(j)   "satisfy demand at market j";

cost ..        z  =e=  sum((i,j), c(i,j) * x(i,j)) ;
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;
solve transport using lp minimizing z ;

Parameter   m(*,*)  "commodity movement";
m(i,j) = x.l(i,j);
m("total",j) = sum(i, x.l(i,j));
m(i,"total") = sum(j, x.l(i,j));
m("total","total") = sum(j, m("total",j));
option decimals = 0;
display m;
```

This model may be easily extended by adding more data:

```
* Data section
Sets  i   "canning plants"   / Seattle, San-Diego, Baltimore, Dallas    /
      j   "markets"          / New-York, Chicago, Topeka, Boston, Miami /;
```

```
Parameters  a(i)  "capacity of plant i in cases"
              / Seattle   350, San-Diego  600, Baltimore 450, Dallas  750 /

            b(j)  "demand at market j in cases"
              / New-York 325, Chicago 300, Topeka 275, Boston 330, Miami 290 /;

Table       d(i,j)  "distance in thousands of miles"

                          New-York    Chicago   Topeka   Boston   Miami
           Seattle          2.5          1.7      1.8     3.1      3.3
           San-Diego        2.5          1.8      1.4     3.0      2.7
           Baltimore        0.2          0.7      1.8     0.4      1.1
           Dallas           1.5          0.9      0.5     1.8      1.3 ;

Scalar f  "freight in dollars per case per thousand miles" /90/ ;

Parameter   c(i,j)  "transport cost in thousands of dollars per case" ;
c(i,j) = f * d(i,j) / 1000 ;

* Model Section
Positive Variable  x(i,j)  "shipment quantities in cases";
Variable           z       "total transportation costs in thousands of dollars";

Equations cost        "define objective function"
          supply(i)   "observe supply limit at plant i"
          demand(j)   "satisfy demand at market j";

cost ..        z  =e=  sum((i,j), c(i,j) * x(i,j)) ;
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;
solve transport using lp minimizing z ;

Parameter   m(*,*)  "commodity movement";
m(i,j) = x.l(i,j);
m("total",j) = sum(i, x.l(i,j));
m(i,"total") = sum(j, x.l(i,j));
m("total","total")=sum(j, m("total",j));
option decimals = 0;
display m;
```

Observe that the two sets (`i` and `j`) were enlarged, the capacity (`a`) and demand (`b`) data were expanded to cover the new plant and market entries and the distance table (`d`) was adjusted accordingly. However, the data calculation, equations, model definition, model solution and report writing sections are identical in the two models.

4.15.2.2 Motivation and Step by Step Guide

As we have demonstrated in the example above, GAMS allows the model structure, calculations and report writing to be developed and implemented using a small data set, that may be easily expanded to larger data sets. Thus, we strongly recommend to start with a representative purposefully small data set and enlarge it to its full size once the work of model development, testing and debugging has been done. In short: work from small to large.

The larger the model the longer it takes to compile it, generate the model, execute and solve it. Generally, time expands exponentially. Working with a large model from the start will often lead to frustration even when the user is trying to find some relatively small data problems.

If a model that has already been completed needs some modification, it will be tempting to use the large data set instead of developing the modifications on a small data set. We strongly advise to use a small data set in this case, as experience shows that this way a considerable amount of time may be saved.

We recommend to follow these steps in model development:

1. Set up a small data set representing the full model with all structural features, set names, parameters etc.
2. Implement all data calculations, model features and report writing calculations.
3. Test the results of step 2 thoroughly.
4. Save the small model. Then implement a larger version with the full data set. Create separate files for data, calculation, model definition and report writing to maintain size independence. Use include files or the save and restart feature.
5. Test the larger model. Use the modeling techniques discussed below to facilitate your work.
6. Keep the small model current. As additional structural features are added to the large model, use it to test them. See section Introducing Strategical Subsets below for an easy way to maintain a small model.

4.15.2.3 Modeling Techniques

If users follow the steps for model development outlined in section Motivation and Step by Step Guide above, they will notice that it will not always be possible to model every needed feature with the small model. It is important to carefully choose the small data set so that it has all features of the larger data set. However, occasionally the peculiarities and interrelationships of the full data set cannot be reproduced in the small data set. In this section we will introduce some modeling techniques for finding problems that arise *only* when the full data set is used. They include saving and restarting to isolate the problem area, strategically introducing subsets and data reduction.

Isolating Problem Areas through Saving and Restarting

Suppose we have a model with a large data set that takes several hours to run and we wish to add some lines of code in a relatively small segment. The best way to do this is by isolating the relevant part. Isolating the part we wish to modify makes it possible to do tests and repairs without having to input data, do initial calculations and solve the whole model with each run. We recommend to use save and restart files.

For example, in chapter The Save and Restart Feature we demonstrate how to split the transportation model [**TRNSPORT**] in three parts: the file `tranmodel.gms` contains the data and the model, the file `transolve.gms` contains the solve statement and the file `tranreport.gms` contains a display statement. To run the whole model we use the following sequence, saving and restarting from the saved file:

```
> gams tranmodel s=s1
> gams transolve  r=s1  s=s2
> gams tranreport r=s2
```

Assume we want a more elaborate report than just the display of some level values. As the file `tranreport.gms` contains the code relevant for reporting, we will modify only this file. Then we will test the result by running only this file, restarting from `s2`, without having to solve the whole model repeatedly.

Introducing Strategical Subsets

When full data sets are used in debugging or development, it is often helpful to narrow the focus on a few items in a set by introducing subsets. The following example is a modified version of the extended transportation model from section An Illustrative Example above.

```
* Data section
Sets  i   "canning plants"   / Seattle, San-Diego, Baltimore, Dallas /
      j   "markets"          / New-York, Chicago, Topeka, Boston, Miami / ;

Sets  plants(i)   "a reduced set of canning plants"
                            / Seattle, San-Diego /
      markets(j)  "a reduced set of demand markets"
                            / New-York, Chicago, Topeka / ;

*plants(i) = yes; markets(j) = yes;

Parameters  a(i)  "capacity of plant i in cases"
               / Seattle   350, San-Diego  600, Baltimore 450, Dallas  750 /

            b(j)  "demand at market j in cases"
               / New-York 325, Chicago 300, Topeka 275, Boston 330, Miami 290 /;

Table       d(i,j)  "distance in thousands of miles"

                            New-York    Chicago   Topeka   Boston   Miami
           Seattle            2.5          1.7      1.8      3.1      3.3
           San-Diego          2.5          1.8      1.4      3.0      2.7
           Baltimore          0.2          0.7      1.8      0.4      1.1
           Dallas             1.5          0.9      0.5      1.8      1.3;

Scalar f  "freight in dollars per case per thousand miles" /90/ ;

Parameter   c(i,j)  "transport cost in thousands of dollars per case" ;
c(plants,markets) = f * d(plants,markets) / 1000 ;

* Model section
Positive Variable  x(i,j)  "shipment quantities in cases";
Variable           z       "total transportation costs in thousands of dollars";

Equations cost        "define objective function"
          supply(i)   "observe supply limit at plant i"
          demand(j)   "satisfy demand at market j";

cost ..              z  =e=  sum((plants,markets), c(plants,markets) * x(plants,markets)) ;
supply(plants) ..    sum(markets, x(plants,markets))  =l=  a(plants) ;
demand(markets) ..   sum(plants, x(plants,markets))   =g=  b(markets) ;

Model transport /all/ ;
solve transport using lp minimizing z ;
```

Observe that we introduced the subsets `plants` and `markets` that contain only some of the elements of their supersets `i` and `j`. Note that all tables, parameters and variables are defined with the supersets, the equations are *declared* over the supersets, but *defined* over the subsets and the calculation of the parameter `c` is also restricted to the subsets. Hence the model is restricted to the elements of the subsets. However, it is easy to change the restricted model back to the full model by removing the asterisks indicating a comment line:

```
plants(i) = yes; markets(j) = yes;
```

Observe that the sets `plants` and `markets` are now dynamic sets. Note that this assignment could be inserted anywhere in the code. Thus, introducing strategic subsets may be combined with isolating problem areas, as detailed in section Isolating Problem Areas through Saving and Restarting above.

Introducing strategic subsets has proven to be an effective way of maintaining a small data set with little effort. Users only have to choose elements that are representative for model development and debugging from the full sets.

Reducing Data

Recall that GAMS skips cases where data items equal zero. Thus a large model may be reduced by temporarily removing data from data sets by simply setting items to zero. Consider the following example:

```
Sets      o         'origin'                          / o1*o100 /
          d         'destination'                     / d1*d100 /;
Parameter dist(o,d) 'distance';
dist(o,d) = 120 + 50*ord(d) - 0.5*ord(o);

Sets      so(o)     'small set of origins for testing' / o4, o47, o91 /
          sd(d)     'small set of destinations'        / d3, d44, d99 /;

dist(o,d) $ (not (so(o) and sd(d))) = 0;

Parameter cost(o,d) 'transportation cost';
cost(o,d) $ dist(o,d) = 3 + 2*dist(o,d);
display cost, dist;
```

Note that we introduced strategic subsets and used them in the logical condition of a conditional assignment to set almost all entries of the parameter `dist` to zero. Note further, that the assignment for the parameter `cost` is conditioned on nonzero entries for the distance. Now, if the model were conditioned on nonzero transportation costs, the size of the whole model would be greatly reduced.

4.15.3 Increasing Efficiency: Reducing GAMS Execution Time

GAMS can take a long time for computations and model generation. There are some signs which indicate that it may be possible to reduce the execution time, e.g., an execution time that is unexpectedly long in general or a long execution of a single line, which could be seen, if the log shows the same line number for a long time.

In this section we will discuss how to find the causes for slow program execution and how to eliminate the main causes for slow execution.

4.15.3.1 Finding the Causes for Slow Program Execution

The best strategy for discovering the causes for slow execution is a combination of the techniques discussed in section Small to Large: Aid in Development and Debugging above and the techniques that we will introduce in this section, including generating an execution profile and isolating terms in slow statements. We will also touch briefly on observing the log file and we will point out why this is not the first choice.

Generating an Execution Profile

The quickest way to find GAMS statements that take particularly long to execute, is generating an execution profile in the output file. The execution profile contains the individual and cumulative time required to execute the sections of the GAMS model as well as information on memory use. An execution profile is generated when the option profile is assigned a value larger than zero (zero is the default). This can be done either by setting a command line parameter or by using the option statement. We will show an example of an execution profile below. For more information on execution profiles, further examples and details on the values the option `profile` may take, see the detailed description here.

Consider the following example:

```
option profile = 1;
option limrow = 0; option limcol = 0;
option solprint = off;

Sets    a / 1*22 /, b / 1*22 /, c / 1*20 /,
        d / 1*20 /, e / 1*22 /;

Parameters x(e,d,c,b,a), y, z(a,b,c,d,e);
x(e,d,c,b,a) = 10;
z(a,b,c,d,e) = x(e,d,c,b,a);
y            = sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a));

Variable obj;
Positive Variable var(e,b,a);

Equations objeq, r(b,c,d), q(a,b,c);

objeq..      obj =e= sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a) * var(e,b,a));
r(b,c,d)..   sum((a,e), var(e,b,a)) =l= sum((a,e), x(e,d,c,b,a)*z(a,b,c,d,e));
q(a,b,c)..   sum((d,e), var(e,b,a)/x(e,d,c,b,a)*z(a,b,c,d,e)) =l= 20;

Model slow /all/;
solve slow maximizing obj using lp;

Parameter sumofvar;
sumofvar = sum((a,b,c,d,e), z(a,b,c,d,e)*x(e,d,c,b,a)*var.l(e,b,a));
display sumofvar;
```

The listing file will contain an execution profile like this (spread over the file):

```
----      9 Assignment x              0.374     0.374 SECS    109 MB  4259200
----     10 Assignment z              2.231     2.605 SECS    286 MB  4259200
----     11 Assignment y              2.324     4.929 SECS    286 MB        0
----     23 Solve Init slow           0.000     4.961 SECS    286 MB
----     18 Equation   objeq          3.510     8.471 SECS    287 MB        1
----     19 Equation   r              3.088    11.559 SECS    464 MB     8800
----     20 Equation   q              5.741    17.300 SECS    470 MB     9680
----     23 Solve Fini slow           0.780    18.080 SECS    470 MB  4482809
----     23 GAMS Fini                 0.359     0.359 SECS    470 MB
----      1 InitE                     0.032     0.032 SECS    213 MB
----      1 ExecInit                  0.000     0.032 SECS    213 MB
----     23 Solve Alg  slow           0.000     0.032 SECS    213 MB
----     23 Solve Read slow           0.000     0.032 SECS    215 MB
----     26 Assignment sumofvar       2.620     2.652 SECS    287 MB        0
----     27 Display                   0.032     2.684 SECS    287 MB
----     27 GAMS Fini                 0.000     0.000 SECS    287 MB
```

The first column shows the line number in the input file of the GAMS statement that is executed. The second column reports the type of statement. For an overview of all GAMS statements, see section Classification of GAMS Statements. The next two columns give the individual time needed to execute the respective statement and the cumulative time spent so far. The memory use follows and finally, the number of assignments generated in the respective line is shown.

In addition, there is a `Profile Summary` at the end of the lst file showing the most expensive statements:

```
---- Profile Summary (19 records processed)
     5.741   0.470GB        20 Equation   q (9680)
     3.510   0.287GB        18 Equation   objeq (1)
     3.088   0.464GB        19 Equation   r (8800)
     2.620   0.287GB        26 Assignment sumofvar (0)
     2.324   0.286GB        11 Assignment y (0)
     2.231   0.286GB        10 Assignment z (4259200)
     0.780   0.470GB        23 Solve Fini slow (4482809)
     0.374   0.109GB         9 Assignment x (4259200)
     0.359   0.470GB        23 GAMS Fini
     0.032   0.213GB         1 InitE
```

This shows that the execution of the statements in line numbers 20, 18, 19, 26, 11 and 10 are the most expensive ones (in this order). One reason is an inconsistent order when sets are referenced; we will discuss this topic in section Ordering Indices Consistently below.

Note that the execution profile may contain many lines that are not informative since the execution times reported are negligible. These lines may be suppressed by using the the option profileTol to specify the minimum execution time (in seconds) that is to be included. Observe that the option `profileTol` is available as command line parameter and option statement.

Note further, that the command line parameter profileFile facilitates writing the profiling information to a separate file (instead of the listing file).

Isolating Terms in Slow Statements

In some cases the execution profile shows that the cause for a long execution time is connected with a very long statement. For example, the objective function in some models and some report calculations may take hundreds of lines of code and can contain many terms that are added. If such a long statement is problematic in terms of execution time, it will be necessary to deactivate parts of the code and run the program repeatedly to find the precise lines that are at the root of the problem. This can be done by using comments.

Observing the Log File

Some modelers choose to examine the log file or watch the screen during execution to find the causes for slow program execution. However, we advise against this approach for the following reasons:

- Statements that are executed slowly are easily missed and often statements are misidentified. In addition, screen watchers may be distracted and will have to repeat the process.
- GAMS line reporting can be misleading if flow control statements like if statements and loop statements are executed. For example, individual calculations in a loop are not reported to the screen. A user watching the screen would notice that the loop takes a lot of time, but there is no indication which statement within the loop is problematic. This applies to all GAMS control structures.

Therefore we recommend to use the option `profile` as the main tool for finding the causes for slow program execution. For details, see section Generating an Execution Profile above. In addition, see the techniques outlined in section Advice for Repairing Puzzling Nonworking Code below.

4.15.3.2 Eliminating the Main Causes for Slow Program Execution

The main reasons for a slow program execution include an inconsistent index order when sets are referenced and taking irrelevant cases into consideration. In this section we will give some guidance on how to eliminate these causes, and also point to problems due to the scaling of a model which could cause a unnecessarily long execution time for the solver.

Ordering Indices Consistently

GAMS employs a sparse matrix data storage scheme. For example, consider the parameter `p(a,b,c)`. Assume that the set `a` has k elements, the set `b` has n elements and `c` has m elements. Then the entries for `p` are stored in the following order:

```
a1 b1 c1
a1 b1 c2
...
a1 b1 cm
a1 b2 c1
...
a1 b2 cm
...
a1 bn cm
a2 b1 c1
...
ak bn cm
```

Note that it is a systematic order where the last entry varies the fastest and the first the slowest. Observe that GAMS will withdraw entries from memory fastest if they are referenced in an order consistent with the storage order. Thus, in the following example, the first assignment statement will be processed faster than the second assignment statement.

```
x(a,b,c) = p(a,b,c);
y(b,c,a) = p(a,b,c);
```

Note

> GAMS will execute a program fastest if the sets are always referenced in the same order in definitions, assignments and equations.

The example that follows illustrates this principle. First we will solve a program where the indices appear in an arbitrary order and we will record the output generated by setting the option profile to 1. Then we will reformulate the program so that the indices will always appear in an alphabetical order and solve it again, recording the `profile` output. In the final step, we will compare the execution times of the two runs. We will use the example introduced above.

Note that the indices in the parameters and equations appear in a random order. Here is the profile from the six most expensive statements again:

```
----     10 Assignment z              2.231     2.605 SECS    286 MB  4259200
----     11 Assignment y              2.324     4.929 SECS    286 MB        0
----     18 Equation   objeq          3.510     8.471 SECS    287 MB        1
----     19 Equation   r              3.088    11.559 SECS    464 MB     8800
----     20 Equation   q              5.741    17.300 SECS    470 MB     9680
----     26 Assignment sumofvar       2.620     2.652 SECS    287 MB        0
```

In the next step we reformulate the program such that the indices always appear in the same order. For example, we define the parameter `x` as `x(a,b,c,d,e)` instead of `x(e,d,c,b,a)`. Here is the complete rewritten model:

```
option profile = 1;
option limrow = 0; option limcol = 0;
option solprint = off;

Sets    a / 1*22 /, b / 1*22 /, c / 1*20 /,
        d / 1*20 /, e / 1*22 /;

Parameters x(a,b,c,d,e), y, z(a,b,c,d,e);
x(a,b,c,d,e) = 10;
z(a,b,c,d,e) = x(a,b,c,d,e);
y            = sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e));

Variable obj;
Positive Variable var(a,b,e);

Equations objeq, r(b,c,d), q(a,b,c);

objeq..      obj =e= sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e) * var(a,b,e));
r(b,c,d)..   sum((a,e), var(a,b,e)) =l= sum((a,e), x(a,b,c,d,e)*z(a,b,c,d,e));
q(a,b,c)..   sum((d,e), var(a,b,e)/x(a,b,c,d,e)*z(a,b,c,d,e)) =l= 20;

Model slow /all/;
solve slow maximizing obj using lp;

Parameter sumofvar;
sumofvar = sum((a,b,c,d,e), z(a,b,c,d,e)*x(a,b,c,d,e)*var.l(a,b,e));
display sumofvar;
```

After running the modified program, the profile for expensive statements looks like this:

```
----     10 Assignment z             0.593     0.983 SECS    215 MB  4259200
----     11 Assignment y             0.671     1.654 SECS    215 MB        0
----     18 Equation   objeq         1.778     3.432 SECS    215 MB        1
----     19 Equation   r             2.215     5.647 SECS    392 MB     8800
----     20 Equation   q             1.763     7.410 SECS    398 MB     9680
----     26 Assignment sumofvar      0.952     0.983 SECS    216 MB        0
```

Observe that executing for example the assignment to `z` took just 0.593 seconds compared to 2.231 seconds in the first run. Substantial percentage reductions were achieved in all time consuming cases by consistently referencing the sets in the same order.

Replace loops with assignments

The following statement assigns a constant value to a parameter.

```
loop((i,j,k),p(i,j,k)=2;)
```

The following assignment is preferred instead.

```
p(i,j,k)=2;
```

Restricting Assignments and Equations to Relevant Cases

Assignments

Assume that we have a set of cities with different production capacities and demands for various products. We want to know the maximum transportation cost (which depends on the distance, the amount shipped and a fixed factor) from each city to all others. This cost can be calculated in the following way:

```
Sets c "cities"   / c1*c800 /
     p "products" / p1*p10  /;
Alias (c,cc);

Parameter capacity(c,p)  "Production capacity for product p in city c"
          demand(c,p)    "Demand for product p in city c"
          distance(c,cc) "Distance between two cities";

*Generate some sparse, random data
capacity(c,p)$(uniform(0,1)<0.05) = uniformInt(150,250);
demand(c,p)$(uniform(0,1)<0.025)  = uniformInt(50,150);
distance(c,cc)$(not sameas(c,cc)) = uniformInt(10,800);

Parameter maxCost(c,cc) "Maximum transportation costs between two cities";

maxCost(c,cc) = sum(p, min(capacity(c,p), demand(cc,p))*distance(c,cc)*90);
```

The performance profile will tell us something like this:

```
----     16 Assignment maxCost       0.265     0.436 SECS     19 MB   8756
```

Since we know, that the parameter `maxCost` will be zero for a pair of cities if there is no product with production capacity in the first city and demand in the second one, we could reduce the execution time for the last assignment:

```
Sets c "cities"   / c1*c800 /
     p "products" / p1*p10 / ;
Alias (c,cc);

Parameter capacity(c,p)  "Production capacity for product p in city c"
          demand(c,p)    "Demand for product p in city c"
          distance(c,cc) "Distance between two cities";

*Generate some sparse, random data
capacity(c,p)$(uniform(0,1)<0.05) = uniformInt(50,150);
demand(c,p)$(uniform(0,1)<0.025)  = uniformInt(50,150);
distance(c,cc)$(not sameas(c,cc)) = uniformInt(10,800);

Parameter maxCost(c,cc) "Maximum transportation costs between two cities";

maxCost(c,cc)$sum(p, capacity(c,p)*demand(cc,p))
  = sum(p, min(capacity(c,p), demand(cc,p))*distance(c,cc)*90);
```

So we did not do the calculation of `maxCost` if we knew before, that it must be zero anyway. This results in a reduced runtime:

```
----     17 Assignment maxCost       0.031     0.187 SECS     19 MB   8756
```

Note

To restrict computations in assignment to the relevant cases, we recommend using dollar conditions and filtering sets. These concepts are introduced and discussed in detail in chapter Conditional Expressions, Assignments and Equations.

For more examples, see sections Conditional Assignments and Conditional Indexed Operations.

Variables and Equations

Like assignments, variables and equations need to be restricted to relevant cases to avoid unnecessary inefficiencies. Dollar conditions and filtering sets may be used over the domain of definition as well as in the body of an equation.

Lets extend the assignment example from the previous paragraph and use the generated data in a transportation model:

```
Sets c "cities"   / c1*c800 /
     p "products" / p1*p10 / ;
Alias (c,cc);

Parameter capacity(c,p)  "Production capacity for product p in city c"
          demand(c,p)    "Demand for product p in city c"
          distance(c,cc) "Distance between two cities";

*Generate some sparse, random data
capacity(c,p)$(uniform(0,1)<0.05) = uniformInt(150,250);
demand(c,p)$(uniform(0,1)<0.025)  = uniformInt(50,150);
distance(c,cc)$(not sameas(c,cc)) = uniformInt(10,800);

Parameter shipCost(c,cc) "Transportatin costs between two cities per case"
          maxCost(c,cc)  "Maximum transportatin costs between two cities";

shipCost(c,cc) = distance(c,cc)*90;
maxCost(c,cc)$sum(p, capacity(c,p)*demand(cc,p))
  = sum(p, min(capacity(c,p), demand(cc,p))*shipCost(c,cc));

Variables
     x(c,cc,p)  "shipment quantities in cases"
     z          "total transportation costs in thousands of dollars" ;

Positive Variable x ;

Equations
     cost         "define objective function"
     supply(c,p)  "observe supply limit at plant i"
     dem(cc,p)    "satisfy demand at market j" ;

cost..         z  =e=  sum((c,cc,p), shipCost(c,cc)*x(c,cc,p)) ;
supply(c,p)..  sum(cc, x(c,cc,p)) =l=  capacity(c,p) ;
dem(cc,p)..    sum(c,  x(c,cc,p)) =g=  demand(cc,p) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;
```

The `Profile Summary` tells us, that the equations are rather expensive to generate and also the reading of the solution takes some time because of the size of the model:

```
---- Profile Summary (18 records processed)
    98.780   1.070GB        34 Equation   dem (8000)
    26.864   0.515GB        38 Solve Read transport
    25.303   0.454GB        32 Equation   cost (1)
     6.599   0.813GB        33 Equation   supply (8000)
```

However, as in the previous example, we know, that a product `p` won't be shipped from city `c` to city `cc` if there is either no production capacity in the first city or no demand in the second one. So we could reduce the size of our model by not generating variables and equations from which we know, that they are irrelevant for the solution. Here is a improved formulations of the equations:

```
cost..                      z  =e=  sum((c,cc,p)$(capacity(c,p)*demand(cc,p)), shipCost(c,cc)*x(c,c
supply(c,p)$capacity(c,p).. sum(cc$demand(cc,p), x(c,cc,p))  =l=  capacity(c,p) ;
dem(cc,p)$demand(cc,p)..    sum(c$capacity(c,p),  x(c,cc,p)) =g=  demand(cc,p) ;
```

This decreases the size of the model and thus the execution time to generate the model and load the solution significantly:

```
---- Profile Summary (18 records processed)
     0.031   0.035GB        33 Equation   cost (1)
     0.031   0.034GB        39 Solve Read transport
     0.016   0.035GB        34 Equation   supply (380)
```

Note that the equation `dem` does not even show up in the summary anymore since its generation was done to quickly.

For more details on conditions in equations, see section Conditional Equations.

Keep the model well scaled

Model solutions within GAMS frequently require manipulation of large matrices and many computations. The heart of most solvers includes many numerical procedures such as a sparse matrix inverter and sets of convergence and infeasibility tolerances. Numerical problems often arise within such procedures. Poorly scaled models can cause excessive time to be taken in solving or can cause the solver to fail. GAMS can assist the user to formulate a well scaled model. Details about this can be found in the sections Model Scaling - The Scale Option and Scaling Variables and Equations.

4.15.3.3 Other Approaches

In addition to the techniques discussed in section Eliminating the Main Causes for Slow Program Execution above, the following approaches may help to reduce the time needed for program execution:

- Trying another appropriate solver.
- Reformulating the model. This may yield particularly good results, if the model is reformulated in such a way that another model type is used, that is easier to solve or for which more advanced solver technology is available.

- Using starting points for NLP models, as discussed in section Specifying Initial Values.
- Trading memory for time.

We conclude the discussion of this topic with an example that demonstrates how memory may be traded for time. If an extensive calculation is repeated many times in a model, it may be possible to restructure the code so that the calculation is performed only once, then the result is saved and accessed later. Consider the following equation:

```
obj..   z =e= sum[(i,j,k,l), a(i,j,k,l)*sum(m, u(m,i))];
```

The execution time may be substantially reduced by defining a new parameter, say `p`, for the second sum and using this parameter in the equation:

```
Parameter p(i);
p(i) = sum(m, u(m,i));
...
obj..   z =e= sum[(i,j,k,l), a(i,j,k,l)*p(i)];
```

There is only one caveat: Users need to carefully consider whether the input data, here `u(m,i)`, is modified between the assignment for the new parameter `p` and the equation where `p` is used. If `u` is updated, then the assignment statement needs to be repeated, otherwise the data that enters the equation will not be current.

4.15.4 Increasing Efficiency: Reducing Memory Use

Besides slow program execution, excessive memory use may be of concern for modelers. In this section we will present some approaches on how to find the causes for extraordinary memory use and give some advice on eliminating the main causes for it.

4.15.4.1 Finding the Causes for Excessive Memory Use

The main techniques for finding the causes for excessive memory use are the same as those for finding the causes for slow program execution. We discussed these techniques in section Finding the Causes for Slow Program Execution above.

In addition, the option dmpUserSym is useful in this context. GAMS will report the number of records stored for each symbol at the point in the program where the option `dmpUserSym` is inserted together with some rough memory estimate.

Consider the following example:

```
Sets i /1*5 /,  j /1*5 /,  k /1*5 /,  l /1*5 /,
     m /1*5 /,  n /1*5 /,  o /1*5 /;
Parameters y(i,j,k,l,m,n,o)
           q(i,j,k);
Variables  x(i,j,k,l,m,n,o)
           f(i,j,k)
           obj;
y(i,j,k,l,m,n,o)       = 10;
q(i,j,k)               = 10;
```

```
x.up(i,j,k,l,m,n,o)    = 10;
x.scale(i,j,k,l,m,n,o) = 1000;

Equations  z(i,j,k,l,m,n,o)
           res(i,j,k)
           ob;

ob..                obj  =e= sum((i,j,k,l,m,n,o), x(i,j,k,l,m,n,o));
z(i,j,k,l,m,n,o)..  x(i,j,k,l,m,n,o) =l= 8;
res(i,j,k)..        f(i,j,k) =l= 7;

Model memory /all/;
option dmpUserSym;
solve memory maximizing obj using lp;
```

Note that an option statement with `dmpUserSym` was added before the solve statement. It generates the following *memory dump* that is included in the execution output of the listing file:

```
SYMBOL TABLE DUMP (USER SYMBOLS ONLY), NR ENTRIES = 16
ENTRY                              ID   TYPE DIM  LENGTH MEMORYEST DEFINED ASSIGNED DATAKNOWN
  135                               i    SET   1       5       0 MB    TRUE    FALSE      TRUE
  136                               j    SET   1       5       0 MB    TRUE    FALSE      TRUE
  137                               k    SET   1       5       0 MB    TRUE    FALSE      TRUE
  138                               l    SET   1       5       0 MB    TRUE    FALSE      TRUE
  139                               m    SET   1       5       0 MB    TRUE    FALSE      TRUE
  140                               n    SET   1       5       0 MB    TRUE    FALSE      TRUE
  141                               o    SET   1       5       0 MB    TRUE    FALSE      TRUE
  142                               y  PARAM   7   78125       3 MB   FALSE     TRUE     FALSE
  143                               q  PARAM   3     125       0 MB   FALSE     TRUE     FALSE
  144                               x    VAR   7   78125       5 MB   FALSE     TRUE     FALSE
  145                               f    VAR   3       0       0 MB   FALSE     TRUE     FALSE
  146                             obj    VAR   0       0       0 MB   FALSE     TRUE     FALSE
  147                               z    EQU   7       0       0 MB   FALSE     TRUE     FALSE
  148                             res    EQU   3       0       0 MB   FALSE     TRUE     FALSE
  149                              ob    EQU   0       0       0 MB   FALSE     TRUE     FALSE
  150                          memory  MODEL   0       3               TRUE     TRUE      TRUE
END OF SYMBOL TABLE DUMP
```

The column `ID` contains the names of the symbols, the column `TYPE` gives the data type of the respective entry, the column `DIM` reports the number of indices and the column `LENGTH` gives the number of records that is related to memory use, which is estimated in the column `MEMORYEST`. Note that the other columns are not relevant for this discussion.

Observe that the rows with high counts in column `LENGTH` indicate symbols within the GAMS program which have large numbers of internal records that must be stored. This is associated with corresponding memory requirements. Note also that not all length counts are of equal significance. In particular, variables and equations use more memory per element than parameters, since they have bounds, levels, marginals and scales that are associated with them. Parameters use more memory per element than sets, since sets may need just one indicator for `yes` or `no`. However, the explanatory text for set elements might increase the memory requirements for set elements.

Nevertheless, users may use this report to identify items with many records and verify that all of them are actually needed. For more details, see section Eliminating the Main Causes for Excessive Memory Use below.

4.15.4.2 Eliminating the Main Causes for Excessive Memory Use

As detailed in section Eliminating the Main Causes for Slow Program Execution above, the main causes for a slow program execution include an inconsistent index order when sets are referenced and taking irrelevant cases into consideration. These programming habits also tend to cause excessive memory use. In this section we will give some advice on avoiding memory traps and show how the memory may be cleared of data that is no longer needed.

Avoiding Memory Traps

Users may inadvertently use a lot of memory if they import data from a database with long explanatory text for sets or set elements. In addition, setting variable attributes for scaling or bounds may be problematic. Consider the following example:

```
x.scale(i,j,k,l,m) = 100;
x.lo(i,j,k,l,m) = 10;
x.up(i,j,k,l,m) = 77;
```

These assignments will probably set many more values than are relevant for a particular problem. Therefore we recommend to carefully consider which label combinations are actually necessary and to restrict the assignments to these cases by the use of dollar conditions or filtering sets. For more information, see section Conditional Assignments.

Clearing Memory of Unneeded Data

Sometimes a lot of memory space is used for data that is needed at some point, but not later. Consider the following simple example:

```
set i /1*1000/
    j /1*1000/;
parameter distance(i,j)
          cost(i,j);
distance(i,j) = 100+ord(i)+ord(j);
cost(i,j)     = 4+8*dist(i,j);
```

Assume that the parameter `distance` is used only here, but nowhere else in the program. Therefore users may wish to free the memory space occupied by the data connected with `distance`. The option clear may be used to achieve this:

```
option clear = distance;
```

This will reset all entries in the matrix associated with `distance` to zero.

Alternatively, an identifier that is no longer needed could be reset to its default value(s) with an assignment statement. In the example above, we could write:

```
distance(i,j) = 0;
```

This statement will have the same effect as the option statement. The advantage of the option statements is that they offer a more compact alternative that is particularly useful if equations or variables are to be cleared and multiple equation attributes or variable attributes are affected.

Note that the dollar control options $clear and $kill may also be used to free memory. These are compile time directives, which have a similar effect on the memory consumption but have different side effects: while `$clear` will reset the values to their defaults, `$kill` will completely remove the identifier from the program. Hence an identifier that was "killed" may be used later in another declaration and definition statement. For example, the following code snippet is legal:

```
Set i /1, 2 ,3/;
$kill i
Set i /a, b, c/;
```

With `$clear` instead of `$kill` this would cause a compilation error.

4.15.4.3 Setting Memory Limits with HEAPLIMIT

In a server environment and in other cases (e.g. to avoid the use of virtual memory) the amount of memory a GAMS run is allowed to use may have to be limited. The command line parameter heapLimit serves this purpose: the amount of memory for GAMS compilation and execution is limited to a specified number (in MB). If the data storage exceeds this limit, the job will be terminated with return code 10, `out of memory`. In addition, the function heapLimit may be used to interrogate the current limit and to reset it.

Note that limiting memory use for *solver execution* is not possible from within the GAMS program. However, some solvers like the NLP solver CONOPT have their own heapLimit option which ensures that the solver will not use more dynamic memory than specified.

4.15.4.4 Advice for Repairing Puzzling Nonworking Code

Assume a GAMS run was terminated and we cannot get a profile output (e.g. because GAMS ran out of memory and crashes). A memory overrun error causes the operating system buffer handling procedures to generally lose the last few lines of profile information when the job malfunctioned. How do we find the problem in this case?

We recommend to use the techniques outlined in section Modeling Techniques above. In addition, successively deactivating code in search for the last GAMS statement that worked will help in most cases. This can be done by using comments. If at some point the run terminates properly, the user will slowly activate parts of the last statements that were deactivated until the code performance will get worse again. By iteratively activating and deactivating terms, the precise problematic terms may be found. The save and restart feature could also be used to save the results until a certain statement and then to execute only the statements that are suspected to be problematic.

4.16 Comparative Analyses with GAMS

Once a model is completed, it is almost always used to investigate alternative scenarios where the analyst compares the results of various scenario assumptions. In this tutorial we will show how such *comaparative analyses* (also called *sensitivity analyses*) are done with GAMS. We will first demonstrate an easy approach, where we will manually change input parameters, use repeated solves and generate reports. In a second step, we will introduce another approach, where a loop structure will be used to automatically cycle through the scenarios. We recommend to read the sections on the manual approach first, since the sections on the automated approach build on code blocks developed in the early sections.

4.16.1 Manual Approach

Suppose we wish to do a comparative analysis by altering some input data in a model. We will use as an example the farm profit-maximizing model farmcomp.gms. The following vector of prices for primary commodities is a part of the input data:

```
Parameter price(primary)  'prices for products in USD'
         / corn  2.20, soybeans   5.00, beef   0.50 /;
```

We will use these data as a base case and compare it with two alternative scenarios: in the first scenario we will change the price of beef to $0.70 and in the second scenario we will change the price of corn to $2.70.

The GAMS file farmrep.gms is related to our example model. It contains only calculations for report writing and may be included with the dollar control option $include. It will generate a report based on the solution of the last solve that was executed in the GAMS program `farmcomp.gms`. The report consists of several tables. We will focus on the table `Farm Summary` that is associated with the parameter `summary`. The relevant code is given below:

```
Set  alli 'allitems'
        / corn, soybeans, beef, cattle,
          water, cropland, pastureland,
          fertilizer, seed, othercost, veterinary, supplement,
          "April labor", "May labor", "summer labor", "Sept labor", "Oct labor",
          cattlefeed, total / ;

Set  measures 'output measures'
        / "Net Income",  "Land Use", "Dry Cropping", "Irr Cropping",
          "Livestock", "Resource Value", "Product Value" /

Parameter summary(alli,measures)   'Farm Summary';
```

Note that the table for the parameter `summary` will contain rows for the commodities which are elements of the set `alli` and columns for all elements of the set `measures`.

We will use a third GAMS file, mancomp.gms,for our comparative analysis. The code of this third GAMS file follows:

```
$include farmcomp.gms
display price;
$include farmrep.gms

price("beef") = 0.70;
solve farm using LP maximizing netincome;
display price;
$include farmrep.gms

price("corn") = 2.70;
solve farm using LP maximizing netincome;
display price;
$include farmrep.gms
```

Note that this code first solves the original model that also contains the set definitions for the report, displays the initial prices and generates a report. In a second step the price for `beef` is changed to \$0.70, the modified model is solved, the prices for the first alternative scenario are displayed and a report is generated. In a third step the price for `corn` is changed to \$2.70, the model is solved again, the prices for the second alternative scenario are displayed and a third report is generated. Note that in the second alternative scenario (the third solve) the beef price is \$0.70, since it was not reset to base levels after the second run.

There will be three tables associated with the parameter `summary` in the listing file, one for each solve. The first table reports the results associated with the base case:

```
----    279 PARAMETER summary  Farm Summary

             Net Income    Land use  Dry Cropp~  Irr Cropp~   Livestock  Resource ~  Product V~

Corn                                     20.00      200.00                             2.20
Soybeans                                480.00                                         5.00
Beef                                                                                   0.50
cattle                                                             615.79
Water                                                                           16.83
Cropland                    700.00                                             128.49
Pastureland                 130.00                                              84.26
April Labor                                                                     32.34
May Labor                                                                       27.01
Oct Labor                                                                       11.50
Cattlefeed                                                                             4.71
Total         162685.05                 500.00      200.00
```

The second table reports the results from the first alternative scenario where the price for beef was changed to \$0.70:

```
----    351 PARAMETER summary  Farm Summary

             Net Income    Land use  Dry Cropp~  Irr Cropp~   Livestock  Resource ~  Product V~

Corn                                     22.84      160.85                             2.34
Soybeans                                489.86                                         5.00
Beef                                                                                   0.70
cattle                                                             866.67
Cropland                    673.55
Pastureland                 130.00                                            1456.90
April Labor                                                                     82.29
May Labor                                                                       80.53
Sept Labor                                                                      53.57
Oct Labor                                                                       46.21
Cattlefeed                                                                             4.89
Total         373686.10                 512.70      160.85
```

And the third table reports the results from the second alternative scenario where the price for corn was changed to \$2.70 and the price for beef stayed at \$0.70:

```
----    423 PARAMETER summary  Farm Summary

             Net Income    Land use  Dry Cropp~  Irr Cropp~   Livestock  Resource ~  Product V~
```

```
Corn                                  31.98      200.00                              2.70
Soybeans                             410.24                                          5.00
Beef                                                                                 0.70
cattle                                                        866.67
Water                                                                    15.99
Cropland                  642.22
Pastureland               130.00                                       1316.09
April Labor                                                              61.39
May Labor                                                                61.72
Sept Labor                                                               84.92
Oct Labor                                                                87.21
Cattlefeed                                                                           5.36
Total        375839.30                442.22      200.00
```

This quick way to do a comparative analysis has the following drawbacks:

- The relevant output is spread over more than 140 lines.
- There is no cross-scenario report.
- The price of beef in the second alternative scenario is problematic, since it was not automatically reset to the original base price.
- The handling of solves and report writing is repetitive.

The first two issues will be addressed in section Writing Cross-Scenario Reports,a solution for the third issue will be given in section Resetting Data to Base Levels and an alternative approach that will resolve the last issue is presented in section An Automated Approach - Avoiding Repeated Work.

4.16.1.1 Writing Cross-Scenario Reports

We will generate a cross-scenario report by introducing two new sets and a new parameter in a revised version of the third GAMS file, mancompb.gms:

```
Set scenarios / base, beefp, beefcorn /;
Set ordr      / "Scenario Setup", "Scenario Results"/;

Parameter savsumm(ordr,*,alli,scenarios)  'Comparative Farm Summary';
savsumm("Scenario Setup","price",primary,"base") = price(primary);
savsumm("Scenario Results",measures,alli,"base") = summary(alli,measures);
```

Note that the set `scenarios` contains the base case and the two alternative scenarios and the set `ordr` introduces places to save the assumptions and results of the different runs. The new parameter `savsumm` is similar to the parameter `summary` introduced above, but it has two additional dimensions. Observe that the first assignment copies the current setup of the price vector and the second assignment copies the results that are stored in the parameter `summary`.

The full code follows:

```
$include farmcomp.gms
$include farmrep.gms
Set ordr / "Scenario Setup", "Scenario Results" /;
Set scenarios / base, beefp, beefcorn /;
Parameter savsumm(ordr,*,alli,scenarios)  'Comparative Farm Summary';
savsumm("Scenario Setup","price",primary,"base") = price(primary);
savsumm("Scenario Results",measures,alli,"base") = summary(alli,measures);

price("beef") = 0.70;
solve farm using LP maximizing netincome;
display price ;
$include farmrep.gms
savsumm("Scenario Setup","price",primary,"beefp") = price(primary);
savsumm("Scenario Results",measures,alli,"beefp") = summary(alli,measures);

price("corn") = 2.70;
display price ;
solve farm using LP maximizing netincome;
$include Farmrep.gms
savsumm("Scenario setup","price",primary,"beefcorn") = price(primary);
savsumm("Scenario Results",measures,alli,"beefcorn") = summary(alli,measures);

option savsumm:2:3:1;
display savsumm;
```

Observe that the last index in the assignments for `savsumm` is `"base"` after the first solve, `"beefp"` after the second solve and `"beefcorn"` after the third solve. Note that option statement in the penultimate line of the code customizes the output generated by the display statement that follows. For details see section Local Display Control. The listing file will contain the following output:

```
----    436 PARAMETER savsumm  Comparative Farm Summary

                                                    base       beefp    beefcorn

Scenario Setup  .price         .Corn                2.20        2.20        2.70
Scenario Setup  .price         .Soybeans            5.00        5.00        5.00
Scenario Setup  .price         .Beef                0.50        0.70        0.70
Scenario Results.Net Income    .Total          162685.05   373686.10   375839.30
Scenario Results.Land use      .Cropland          700.00      673.55      642.22
Scenario Results.Land use      .Pastureland       130.00      130.00      130.00
Scenario Results.Dry Cropping  .Corn              20.00       22.84       31.98
Scenario Results.Dry Cropping  .Soybeans          480.00      489.86      410.24
Scenario Results.Dry Cropping  .Total             500.00      512.70      442.22
Scenario Results.Irr Cropping  .Corn              200.00      160.85      200.00
Scenario Results.Irr Cropping  .Total             200.00      160.85      200.00
Scenario Results.Livestock     .cattle            615.79      866.67      866.67
Scenario Results.Resource Value.Water             16.83                   15.99
Scenario Results.Resource Value.Cropland          128.49
Scenario Results.Resource Value.Pastureland        84.26     1456.90     1316.09
Scenario Results.Resource Value.April Labor        32.34       82.29       61.39
Scenario Results.Resource Value.May Labor          27.01       80.53       61.72
Scenario Results.Resource Value.Sept Labor                     53.57       84.92
Scenario Results.Resource Value.Oct Labor          11.50       46.21       87.21
Scenario Results.Product Value .Corn                2.20        2.34        2.70
Scenario Results.Product Value .Soybeans            5.00        5.00        5.00
Scenario Results.Product Value .Beef                0.50        0.70        0.70
Scenario Results.Product Value .Cattlefeed          4.71        4.89        5.36
```

In this cross-scenario report all output is in one table and it is easy to compare the base case with the two alternative scenarios.

We could also add percentage change calculations by introducing a further parameter, savsummp:

```
Parameter savsummp(ordr,*,alli,scenarios) 'Comparative Farm Summary (percent chg)';

savsummp(ordr,measures,alli,scenarios)$savsumm(ordr,measures,alli,"base") =
   round{ [savsumm(ordr,measures,alli,scenarios) - savsumm(ordr,measures,alli,"base")]*100
   / savsumm(ordr,measures,alli,"base"),1 };

savsummp(ordr,measures,alli,scenarios)
   $[(savsumm(ordr,measures,alli,"base") = 0) and (savsumm(ordr,measures,alli,scenarios) <> 0)]
   = na;

option savsummp:1:3:1;
display savsummp;
```

Note that both assignment statements are conditional assignments. The first assignment computes percentage changes rounded to one decimal place and the second assignment sets the percentage change to NA if the value in the base case is zero. The output generated by the display statement follows:

```
----    450 PARAMETER savsummp  Comparative Farm Summary (percent chg)

                                                  beefp    beefcorn

Scenario Results.Irr Cropping  .Corn              -19.6
Scenario Results.Irr Cropping  .Total             -19.6
Scenario Results.Livestock     .cattle             40.7        40.7
Scenario Results.Resource Value.Water            -100.0        -5.0
Scenario Results.Resource Value.Cropland         -100.0      -100.0
Scenario Results.Resource Value.Pastureland      1629.0      1461.9
Scenario Results.Resource Value.April Labor       154.4        89.8
Scenario Results.Resource Value.May Labor         198.1       128.5
Scenario Results.Resource Value.Sept Labor           NA          NA
Scenario Results.Resource Value.Oct Labor         301.8       658.4
Scenario Results.Product Value .Corn                6.4        22.7
Scenario Results.Product Value .Beef               40.0        40.0
Scenario Results.Product Value .Cattlefeed          3.9        13.8
```

4.16.1.2 Resetting Data to Base Levels

In the example above the price for beef was changed to $0.70 for the first alternative scenario and it stayed at $0.70 in the second alternative scenario, since we did not reset it manually to the base level. However, in most cases users find it preferable to reset all data to base levels before a new scenario is run. This may be done by saving the base level data in a new parameter, say saveprice, and then resetting the data to base levels before each scenario. In the following final version of the third GAMS file, mancompc.gms, the levels of the commodity prices are reset before each new run:

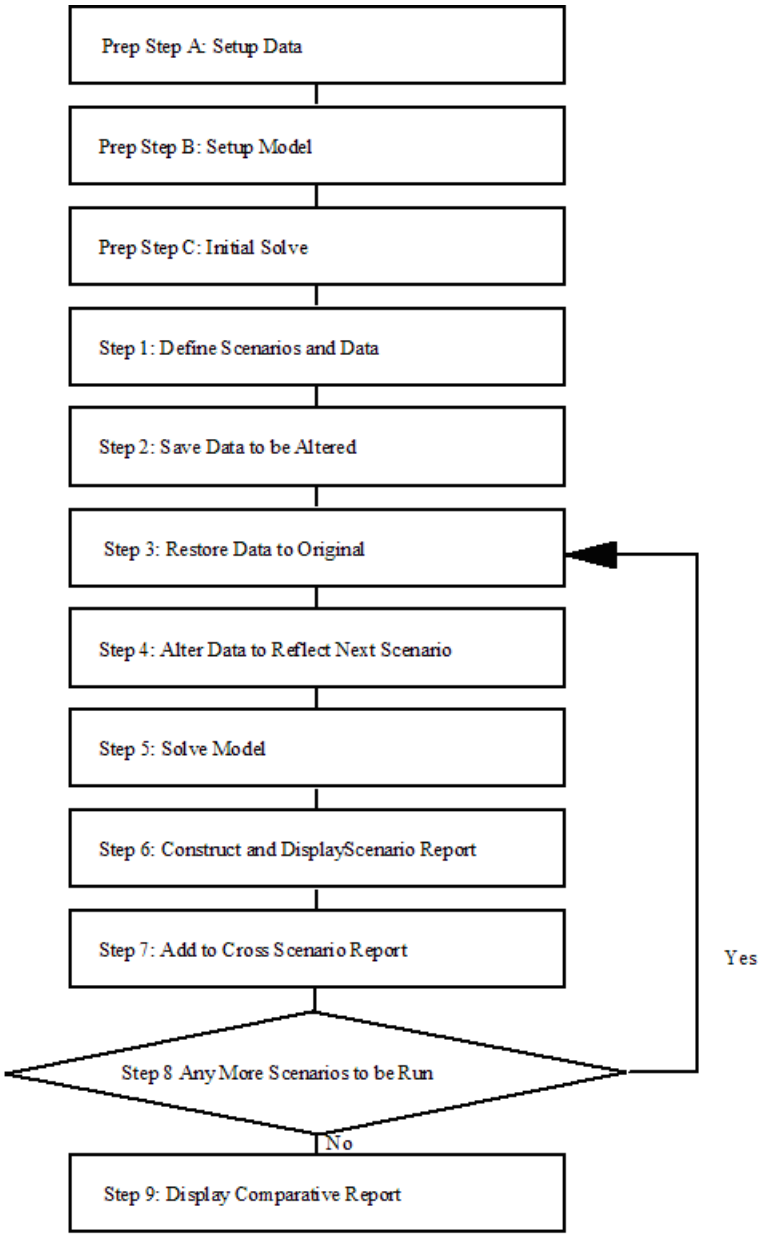
```
$include farmcomp.gms
$include farmrep.gms

Parameter saveprice(alli) 'saved prices';
saveprice(alli) = price(alli);
```

```
Set scenarios / base, beefp, beefcorn /;
Parameter savsumm(ordr,*,alli,scenarios)  'Comparative Farm Summary';
savsumm("Scenario Setup","price",primary,"base") = price(primary);
savsumm("Scenario Results",measures,alli,"base") = summary(alli,measures);

price(alli) = saveprice(alli);
price("beef" ) = 0.70;
solve farm using LP maximizing netincome;
display price ;
$include farmrep.gms
savsumm("Scenario setup","price",primary,"beefp") = price(primary);
savsumm("Scenario Results",measures,alli,"beefp") = summary(alli,measures);

price(alli) = saveprice(alli);
price("corn") = 2.70;
display price ;
solve farm using LP maximizing netincome;
$include farmrep.gms
savsumm("Scenario setup","price",primary,"beefcorn" ) = price(primary);
savsumm("Scenario Results",measures,alli,"beefcorn") = summary(alli,measures);

option savsumm:2:3:1;
display savsumm;
```

In this section we demonstrated how to do a comparitive analysis manually, including a cross-scenario report and resetting the data before each new scenario is run. However, there is still a lot of repetition in how the solves and reports are handled. This issue will be addressed in the next section.

4.16.2 An Automated Approach - Avoiding Repeated Work

The basic structure of a comparative analysis that avoids repetitive instructions for solves and report writing is outlined in the Figure below. Note that the first three boxes represent preparatory steps that are the usual parts of a GAMS program: the initial data is set up, the model is defined and solved. The comparative model analysis begins with the box labeled "Step 1". In Step 1 names for the scenarios are introduced and the scenario data is defined. In Step 2 the base data that will be changed during the scenario runs are saved in a new parameter. We will use this parameter later to restore the data to their base levels before each new scenario run. Starting with Step 3 we enter a loop, where the looping set is the set of scenarios introduced in Step 1. As usual, the statements in the loop are executed for each scenario to be analyzed. The first statement restores the data to their base levels (Step 3). Thus we will always start with the same data. In Step 4 the data and the model is updated for the current scenario and in Step 5 the model solved. In Step 6 a report for the individual scenarios is generated. In Step 7 parameters for cross-scenario comparative reports are saved. In Step 8 we check if more scenarios are to be solved and if this is the case, we return to repeat Steps 3-8 until all scenarios are completed. Finally, we display a comparative report that presents the information saved across scenarios.



This flow chart is implemented in the file `compare.gms` that is given below. Note that it is based on the farm profit-maximizing example discussed in section Manual Approach above. All sets and parameters were introduced in the sections above, only the table `scenprice` in the implementationof Step 1 is new.

```
*Step 1 - Setup scenarios
Set ordr      / "Scenario Setup", "Scenario Results" /;
Set scenarios / base, beefp, beefcorn /;
Parameter savsumm(ordr,*,alli,scenarios) 'Comparative Farm Summary';
Table     scenprice(primary,scenarios)   'price changes by scenario'
                base    beefp   beefcorn
corn                                2.70
soybeans
beef                    0.70            ;

*Step 2 - Save data
Parameter savprice(primary) 'save primary commodity prices';
savprice(primary) = price(primary);
```

```
*Step 3 - Reset data to base level
loop(scenarios,
    price(primary) = savprice(primary);

*Step 4 - Change data to levels needed in scenario
    price(primary)$scenprice(primary,scenarios) = scenprice(primary,scenarios);
    display price;

*Step 5 - Solve model
   solve farm using LP maximizing netincome;

*Step 6 - Single scenario report writing
   $include farmrep.gms

*Step 7 - Cross-scenario report writing
    savsumm("Scenario Setup","price",primary,scenarios) = price(primary);
    savsumm("Scenario Results",measures,alli,scenarios) = summary(alli,measures);

*Step 8 - End of loop
    );

*Step 9 - Compute and display final results
option savsumm:2:3:1;
display savsumm;
```

Note that the main feature that facilitates this automatic approach is a loop statement. Loop statements are introduced and discussed in section The Loop Statement. Note further, that in Step 4 we used a conditional assignment to restrict the change of data to those primary commodities that have nonzero entries for the respective scenario in parameter `scenprice`. For details on conditional assignments, see section Conditional Assignments.

4.16.2.1 Adding A Scenario

Given the implementation of an automated comparative analysis above, it is easy to add a new scenario. Only two small modifications are needed: a name for the new scenario has to be added to the set `scenarios` and the respective data has to be added to the table `scenprice`. Both changes are in Step 1. The lines of the respective code are given below:

```
Set scenarios / base, beefp, beefcorn, new/;
Table scenprice(primary,scenarios)  'price alterations by scenario'
                base    beefp   beefcorn  new
corn                                2.70
soybeans                                    4.32
beef                     0.70                   ;
```

Note that we added a third alternative scenario called `new`, where the price of soybeans is changed to \$4.32.Note further, that the remainder of the code is not changed. Once the new scenario is added to the set `scenarios` and the respective data is specified in the table, it will enter the loop and thus all the statements in the loop will be executed for the new scenario.

Observe that the new scenario above was similar to the other scenarios: the price for a primary commodity was modified. However, in some cases new scenarios require that other data, like resources, are changed. For example, assume that in a new scenario the landtype `"cropland"` is increased by 30%. Note that the available resources were defined in the parameter `available(alli)`, where `available("cropland")` equals 700. To accomodate this new scenario we will add the following lines of code to Steps 1 to 4:

```
*Step 1 - Setup scenarios
Set scenarios / base, beefp, beefcorn, new /;
Table scenavailable(alli,scenarios)  'resource alterations by scenario'
                base    beefp   beefcorn  new
cropland                                  1.3;

*Step 2 - Save data
Parameter saveavailable (alli) 'save available resources';
saveavailable (alli) = available (alli);

*Step 3 - Reset data to base level
loop(scenarios,
    available (alli) = saveavailable (alli);

*Step 4 - Change data to levels needed in scenario
    available(alli)$scenavailable(alli,scenarios) =
       available(alli)*scenavailable(alli,scenarios);
    display price, available;
```

Note that in Step 1 we added a new table, `scenavailable`, to specify the resource alternations by scenario. Observe that the values of `scenavailable` will be used as multipliers in Step 4. In Step 2 we introduced the parameter `saveavailable`, that will play a similar role for the available resources like the parameter `savprice` does for the prices of the primary commodities: the base levels are saved in this parameter and we will use it to restore the values of `available` to their base levels before each new scenario run. This is implemented with the new assignment in Step 3. In Step 4 the data levels of the parameter `available` are modified as required for the current scenario using a conditional assignment. Note the similarities to the assignment for updating the values of `price` in Step 4 above. Steps 5 to 9 remain unchanged. Thus we achieved an automated comparative analysis with an additional scenario where another parameter is modified by adding just a few lines of code. The full code is given in the GAMS file `compareother.gms`.

4.16.2.2 Changing the Structure of a Model

Many studies require modifications of the *structure* of a model. In GAMS, context-sensitive model structures may be implemented with dollar conditions. Dollar conditions may be used to control equations as well as specific terms.

To illustrate, we will extend the profi-maximizing farm model from the previous sections to include a conditional constraint that limits the number of cattle. This constraint will be only active if a scalar that varies with the scenarios is nonzero. Consider the lines of code that follow. Note that the set `animals` is a subset of the set `alli`, the elements of the set `livemanage` denote various ways of managing crops and animals, and `liveprod` is a positive variable for livestock production.

```
$include farmcomp.gms
$include farmrep.gms

Scalar   cowlim   'activates cowlimit constraint' /1/;
Equation cowlimit 'conditional equation on cow limits;
cowlimit$cowlim..   sum((animals,livemanage), liveprod(animals,livemanage)) =l= 100;
Model farmcowlim /all/;

Set ordr      / "Scenario setup", "Scenario Results" /;
Set scenarios / base, cowlim /;
Parameter savprice(primary) 'save primary commodity prices';
savprice(primary) = price(primary);
Parameter cowlims(scenarios) 'cowlimit by scenario'
```

```
          / base 0, cowlim 1/;

loop(scenarios,
    cowlim = cowlims(scenarios);
    solve farmcowlim using LP maximizing netincome;
);
```

Note that in the loop the value of the scalar `cowlim` is updated before each scenario is run. Note further, that this value determines whether the new equation `cowlimit` will be active or not. The complete code is given in the GAMS file `comparemod.gms`.

4.16.3 Ranging analysis

Some users are interested in getting ranging output in the form of LP cost and right hand side ranging results. Unfortunately, the base version of GAMS does not yield such information. The user wishing such information has two alternatives. First, one may cause the model to be repeatedly solved under a set of values for the key parameter using the procedures discussed above, but this is cumbersome if a lot of parameters are involved. Second, one can use solver dependent features of GAMS that can be retrieved into a GAMS parameter. Please refer to the solver manuals (e.g. for GAMS/CPLEX: Sensitivity Analysis) for further information how to use them.

4.17 Good NLP Formulations

In this tutorial we offer some advice and guidance on how to set up or formulate an NLP model so that a solver will be able compute a good solution and do so quickly, reliably, and predictably. Much of this applies to other model classes allowing nonlinear functions, but for ease and simplicity of exposition we focus on the NLP case here.

A good formulation for an NLP model typically involves several things, including specifying sensible initial values, setting variable bounds, and scaling variables and equations. Other factors to consider are techniques for blocking degenerate cycling and the potential benefits of avoiding expressions in nonlinear functions. Finally we look at reformulations and approximations for discontinuous functions like `abs`, `max` and `min` in section Reformulating DNLP Models.

4.17.1 Specifying Initial Values

The variable levels and equation marginals in GAMS are typically passed to a solver to be used as the initial point. The initial values specified are especially important for NLP models for several reasons:

- Non-convex models may have multiple local solutions. Local NLP solvers search for a local solution and return it when it is found. An initial point in the neighborhood of a desired solution is more likely to return that solution.
- Initial values that (almost) satisfy many of the constraints reduce the work involved in finding a first feasible solution.
- Initial values that are close to a local optimum reduce the work required to find that local optimum, therefore reducing solution time.
- The progress of the optimization algorithm is based on good directional information, i.e on good derivatives. The derivatives in a nonlinear model depend on the current point, so an improved initial point can improve solver performance.

Variable levels and equation marginals are specified by setting the variable attribute `.L` and the equation attribute `.m` before solution. This is often done with assignments that occur *before* the solve statement, e.g.:

```
domPrice.L(i,region,t) = domPrice0(i,region,t);
flowLim.m(arcs) = 1;
```

Note

The default value for the variable levels and equation marginals is 0.

The variable *bounds* also play a role in determining the initial point passed to the NLP solver. When a solve occurs, the levels for all variables *in the model* are first projected onto the set defined by the variable bounds. Thus, setting a variable's lower bound to a positive value ensures that the initial value of this variable will never be zero. This is very useful, since in many cases zero is an unsuitable initial value for nonlinear variables. For example, based on the product term $x \cdot y$, an initial value of zero for x will lead to an initial derivative value of zero wrt y, so it will appear as if the function does not depend on y. Variables at zero can also cause numerical difficulties with logarithms, real powers, or divisions. These difficulties occur not just at zero but also for very small values (i.e. values very close to zero) as well.

We recommend to specify as many sensible initial values for the nonlinear variables as possible. It may be desirable to initialize all variables to 1 or to the scale factor if the GAMS scaling option is used. A better alternative is to first select reasonable values for some variables that are known from context or experience to be important and then to use some of the equations of the model to compute the values for other variables. For example, consider the following equation, where `pm`, `pwm` and `er` are variables and `tm` is a parameter:

```
pmDef(i)..   pm(i) =e= pwm(i)*er*(1+tm(i));
```

The following assignment statements use the equation to derive consistent initial values for the variable `pm` from sensible initial values for the variables `pwm` and `er`:

```
pwm.L(i) = 1;
er.L = 1;
pm.L(i) = pwm.L(i)*er.L*(1+tm(i));
```

It would be a mistake to assign only to `pwm` and `er` and assume that the solver will choose to adjust the variable `pm` to make the equation feasible: it could choose to adjust `pwm` or `er` instead. With all the assignments above made, we can be assured that the equation `pmDef` will be satisfied at the initial point.

Setting the initial point by loading a prior solution that has been saved via the savepoint mechanism is also an effective strategy that is very easy to implement.

4.17.2 Setting Variable Bounds

Lower and upper bounds on variables are set by assigning values to the variable attributes `.lo` and `.up` in the following way:

```
price.lo(i,region,t) = 1e-4;
flow.up(arcs) = arcCap(arcs);
```

Lower and upper bounds on variables in nonlinear models serve multiple purposes. Some bounds represent constraints based on the reality that is being modeled. For example, a certain production level must be non-negative or an arc in a network has a flow capacity of at most ten. These bounds are called *model bounds*. Other bounds help the algorithm by preventing it from moving far away from any optimal solution and/or into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called *algorithmic bounds*. Solver performance can be improved and execution errors (see domLim and domUsd) avoided when algorithmic bounds on variables are introduced.

Model bounds are determined by the reality being modeled and do not cause any problems. However, algorithmic bounds must be carefully chosen by the modeler. We recommend to pay particular attention if a variable is the argument in log(x), log10(x) or exp(x) and if a variable occurs in the denominator of a division. If log(x) or log10(x) appears in a model, where `x` is a variable, we recommend a lower bound of `1.e-3` for `x` since `log(x)` gets very small as `x` approaches zero and is undefined for negative values of `x`. In addition, the first derivative gets very large as `x` approaches zero. If exp(x) features in a model, where `x` is a variable, we recommend an upper bound between 20 and 25 for `x`. If a variable `x` appears in the denominator, we recommend a lower bound of `1.e-2` for `x`, since `1/x` is extremely nonlinear for small arguments. Small values for variables used with negative exponents are also not desirable. Solver performance can be improved and execution errors avoided when one introduces algorithmic bounds on variables.

Note that lower and upper bounds facilitate finding a feasible solution as most solvers will honor bounds at all times, but inequalities are not necessarily satisfied at intermediate points. A further advantage of variable bounds compared to inequalities is improved presolve performance: NLP solver preprocessors will typically incur little or no computational overhead due to variable bounds.

4.17.3 Avoiding Expressions in Nonlinear Functions

It is often useful to avoid nonlinear functions of expressions (e.g. a division by the sum of many variables). Instead, an *intermediate* variable can be used for the expression. This applies in particular if the expressions depend on many variables. Consider the following example:

```
variable x(i), y;
equation ydef;
ydef..   y =e= 1 / sum(i, x(i));
```

This example could be reformulated via the intermediate variable `xsum` and its defining equation `xsumdef` in the following way:

```
variable x(i), y, xsum;
equation xsumdef, ydef;
xsumdef..   xsum =e= sum(i, x(i));
ydef    ..   y =e= 1/xsum;
xsum.lo = 1.e-2;
```

In the equation `ydef`, the intermediate variable `xsum` appears in the denominator instead of the original summation. This allows us to impose a lower bound on the variable `xsum` to avoid dividing by zero. Of course, the model will contain more rows and columns if intermediate variables are introduced, but this increase in size is offset by a decrease in complexity and, in many cases, by an increase in sparsity as well.

4.17.4 Scaling Variables and Equations

Recall that nonlinear programming algorithms use the derivatives of the objective function and the constraints to find good search directions and they use function values to determine if constraints are satisfied or not. The *relative size* of the derivatives and the function values is influenced by the units of measurement that are used for the variables and constraints, and will have an effect on the performance of the solver and the result computed. Therefore, a proper, consistent scaling of the model is important to the success of the solution algorithm and the quality of the answer returned.

For example, assume that two goods are equally costly: both cost $1 per kg. However, the first is specified in grams and the second in metric tons, so that their coefficients in the cost function will be vastly different: $1000 per gram and $0.001 per ton respectively. If cost is measured in $1000 units, then the coefficients will be 1 and 1.e-6 respectively. This discrepency in size may cause the algorithm to ignore the variable with the smaller coefficient, since the coefficient is comparable to some of the zero tolerances. To avoid such problems, the units of measurements need to be carefully chosen, that is, variables and constraints need to be properly scaled.

We recommend scaling with the following goals in mind:

- Solution level values of variables should fall into a range around 1, e.g. from 0.01 to 100.
- The magnitude of the nonzero constraint marginals at solution should fall into a range around 1, e.g. from 0.01 to 100.
- The magnitude of the derivatives of the nonlinear terms (i.e. the Jacobian elements) should fall into a range around 1, e.g. from 0.01 to 100, both at the initial point and at the solution.
- The constants in the equations should have absolute values around 1, e.g. from 0.01 to 100.

Well-scaled variables are measured in appropriate units. In most cases users should select the unit of measurement for the variables such that their expected value is around unity. Of course, there will always be some variation. For example, if `x(i)` is the production at location `i`, one could select the same unit of measurement for all components of `x`, say, a value around the average capacity.

In well-scaled equations the individual terms are measured in appropriate units. After choosing units for the variables users should choose the unit of measurement for the equations such that the expected values of the individual terms are around 1. For example, if these rules are followed, material balance equations will usually have coefficients of 1 and -1.

Usually well-scaled variables and equations result in well-scaled derivatives. To check whether the derivatives are well-scaled, we recommend running the model with a positive value for the option limrow and inspecting the coefficients in the equation listing of the GAMS list file.

For more about scaling in GAMS, see section Model Scaling - The Scale Option. Note that while many solvers have internal scaling procedures, a better result can generally be achieved by a judicious choice of units by the model developer.

4.17.5 Blocking Degenerate Cycling

Most commercial linear programming solvers use a perturbation technique to avoid degenerate cycling during the solution process: they temporarily add small numbers to the right-hand sides of equations. In general, NLP solvers do not have such an internal feature. Sometimes the success and performance of an NLP solver can be enhanced by a manual perturbation formulation.

In particular, if users observe that the NLP solution process has a large number of iterations where the solver does not make significant progress in altering the objective function value, we recommend to modify

the equations in the model by replacing the value of zero on the right-hand side with a small number. This may accelerate the solution process. Assume that we have the following equation:

$$f(x) \leq 0$$

This could be reformulated as:

$$f(x) \leq \delta * 0.001$$

Here we set δ to 1 if we wish to keep the addition and to zero otherwise. The value of 0.001 is just an example and needs to be adjusted based on the model context. The number should be chosen such that it does not introduce significant distortions into the problem solution. Such an addition quite frequently reduces solution time by helping the solver avoid degenerate cycling. If it is done correctly, the resulting model solution is not qualitatively different from the original model solution. Users may also first solve the model with $\delta = 1$ and subsequently with $\delta = 0$ to get rid of the effects of the small numbers.

We recommend that user avoid using the same number but instead use some systematically varying number or a random number. The technique of adding a small number on the right-hand side may also be used in problems where many equations have the same nonzero value on the right-hand side.

4.17.6 Reformulating DNLP Models

Nonlinear models in GAMS belong to one of the following two classes: *smooth* and *discontinuous*. Typically, all functions with endogenous arguments contained in the model are smooth functions (i.e. functions with continuous derivatives) like `sin`, `exp` and `log`. These models can be solved using NLP. If any of the endogenous functions in the model are not smooth (i.e. are discontinuous), the model cannot be solved as an NLP: the DNLP model type must be used instead. Examples of non-smooth functions include `ceil` and `sign`, where the function itself is not continuous, and `max`, `min`, and `abs`, where the derivatives are not continuous. Typically, NLP solvers are designed to work with continuous derivatives, and much of the convergence theory behind them assumes this continuity. Discontinuous functions or derivatives may cause numerical problems, poor performance, spurious (i.e. wrong) solutions, and other issues, so they should be used with special care and only if necessary.

N.B.: to avoid a proliferation of model types, nonlinear programming is the only model type split into smooth (NLP) and nonsmooth (DNLP) variants. All other model types allowing nonlinear functions (e.g. MINLP, MCP, CNS) include both smooth and nonsmooth functions. This is `not` because nonsmooth functions are less problematic in these contexts. It simply became too unwieldy to maintain this distinction across all types of nonlinear models.

A powerful and effective way to model discontinuous functions is with binary variables, which results in a model of type MINLP. The model **[ABSMIP]** demonstrates this formulation technique for the functions `abs`, `min`, `max` and `sign`. Alternatively, reformulations or approximations may be used to model discontinuous functions such that the resulting model is of type NLP. Here we offer some guidance on how to reformulate or approximate the discontinuous functions `abs`, `max` and `min` using only smooth functions and continuous variables and thus transform a DNLP model into an NLP. This transformation is generally more reliable than solving the original as a DNLP.

Note that some of the *reformulations* suggested below enlarge the feasible space. They rely on the objective function to choose a solution that is contained in the original feasible space, i.e. where the relationship defined by the nonsmooth function holds. If the objective cannot be relied on to do this, it is also possible to use one of the *smooth approximations* for the nonsmooth functions defined below.

4.17.6.1 Reformulating and Approximating the Function ABS

The function `abs(x)` returns the absolute value of the argument `x`. If we are minimizing an absolute value, then we can split this value into its positive and negative parts (both represented by positive variables) and minimize the sum of these variables. This formulation enlarges the feasible region but the optimal solution will be the one where the sum of the positive and negative parts is equal to the absolute value.

```
variables x, y, z;
equations
  obj  '1-norm'
  f ;
obj.. abs(x) + abs(y) =E= z;
f  .. sqr(x-3) + sqr(y+5) =L= 1;

model nonsmooth  / obj, f /;
solve nonsmooth using dnlp min z;

positive variables xPlus, xMinus, yPlus, yMinus;
equations
  obj2 'smooth version of 1-norm'
  xDef
  yDef
  ;
obj2.. xPlus + xMinus + yPlus + yMinus =E= z;
xDef.. x =E= xPlus - xMinus;
yDef.. y =E= yPlus - yMinus;

model smooth / obj2, xDef, yDef, f /;
solve smooth using nlp min z;
```

Note that the discontinuity in the derivative of the function `abs` has been converted into lower bounds on the new variables `xPlus`, `xMinus`, etc: these bounds are handled routinely by any NLP solver. Note too that the feasible space is larger than before. For example, many pairs `xPlus` and `xMinus` satisfy our equation `x =E= xPlus - xMinus;`. However, our objective ensures that one of `xPlus` and `xMinus` will be zero at the solution so that the sum of `xPlus` and `xMinus` wil be the absolute value of `x`.

In case the objective function does not contain a term that tries to minimize the absolute value, a smooth approximation can be used instead of the reformulation described above. This approximation should be close to the absolute value and also have smooth derivatives. Such an approximation for `abs(f(x))` is:

```
sqrt(sqr(f(x)) + sqr(delta))
```

Here `delta` is a small scalar. The value of `delta` controls the accuracy of the approximation and the curvature around `f(x)=0`. The approximation error is largest when `f(x)=0` and decreases as `f(x)` gets farther from zero. A value for `delta` ranging between 1.e-3 and 1.e-4 should be appropriate in most cases. Users could also use a larger value in an initial optimization, reduce it and solve the model again. If `delta` is reduced below 1.e-4, then large second order terms might lead to slow convergence or even prevent convergence. An example of this approximation used for the previous example is below:

```
$macro MYABS(t,d) [sqrt(sqr(t)+sqr(d))]
equation obj3;
obj3.. MYABS(x,1e-4) + MYABS(y,1e-4) =E= z;

model approx / obj3, f /;
solve approx using nlp min z;
```

Note the use of the macro facility to encapsulate the smooth reformulation. As mentioned above, this approximation has its largest error where `f(x)=0`. If it is important to get accurate values at this point, then we recommend the following alternative approximation:

```
sqrt(sqr(f(x)) + sqr(delta)) - delta
```

Note that the only difference is the subtraction of the constant term `delta`. In this case, the error will equal zero at `f(x)=0` and it will increase to `-delta` as `f(x)` moves away from zero.

4.17.6.2 Reformulating and Approximating the Function MAX

The function `max(x1,x2,x3,...)` returns the maximum value of the arguments, where the number of the arguments may vary. Typically, the equation

```
t >= max(f(x),g(y))
```

is replaced by two inequalities:

```
t >= f(x)
t >= g(y)
```

Here `x`, `y` and `t` are variables and `f(x)` and `g(y)` are some functions depending `x` and `y` respectively. Provided the objective function has some term that tries to minimize `t`, one of the constraints will become binding at solution and `t` will equal the maximum of the two terms. The extension to more than two arguments in the function `max` should be obvious. A simple example follows:

```
variables
  x  / LO 0, L 0.2, UP [pi/2] /
  mx 'max[sin(x),cos(x)]'
  z  'objective var'
  ;
equation oDef;
oDef.. x / 100 + max[sin(x),cos(x)] =E= z;

model nonsmooth  / oDef /;
solve nonsmooth using dnlp min z;

equations oDef2, sinBnd, cosBnd;
oDef2.. x / 100 + mx =E= z;
sinBnd.. mx =G= sin(x);
cosBnd.. mx =G= cos(x);

model smooth / oDef2, sinBnd, cosBnd /;
solve smooth using nlp min z;
```

In case the objective function does not force the max-term to be minimized, a smooth approximation for `max(f(x),g(y))` can be used, as in the following example code:

```
[f(x) + g(y) + sqrt(sqr(f(x)-g(y)) + sqr(delta))] /2
```

```
$macro MYMAX(t1,t2,d) [0.5 * [t1 + t2 + sqrt(sqr(t1-t2) + sqr(d))] ]
equation oDef3;
oDef3.. x / 100 + MYMAX(sin(x),cos(x),1e-4) =E= z;

model approx / oDef3 /;
solve approx using nlp min z;
```

Here `delta` is a small scalar, preferably ranging from 1.e-2 to 1.e-4. The approximation error takes its maximum of `delta/2` when `f(x)=g(y)` and decreases as one moves away from this point. To shift the error away from the point of discontinuity, the following approximation can be used:

```
[f(x) + g(y) + sqrt(sqr(f(x)-g(y)) + sqr(delta)) - delta] /2
```

4.17.6.3 Reformulating and Approximating the Function MIN

The reformulation of and approximation to the `min` function is similar to the max case above and will not be repeated in full here. Briefly,

```
t =e= min(f(x),g(y))
```

is replaced by:

```
t =l= f(x)
t =l= g(y)
```

and is effective as long as the objective maximizes `t`. If not, this smooth approximation for `min(f(x),g(y))` can be used:

```
[f(x) + g(y) - sqrt(sqr(f(x)-g(y)) + sqr(delta))] /2
```

4.18 Data Exchange with Other Applications

- Data Exchange with Text Files
- Data Exchange with Microsoft Excel
- Data Exchange with Databases
  - Data Exchange with DB2
  - Data Exchange with MS Access
  - Data Exchange with Oracle
  - Data Exchange with MySQL
  - Data Exchange with SQL Server
  - Data Exchange with SQLite
  - Data Exchange with Sybase

4.18.1 Data Exchange with Text Files

This tutorial describes some ways on how to exchange data between GAMS and text files (usually in ASCII format).

4.18.1.1 Reading Text Files During Compilation

GAMS can read arbitrary text files during compile time by inserting them into the compiler input stream. The file content is then assumed to be GAMS code. Thus, including a text file within a data statement (see also Data Entry: Parameters, Scalars and Tables) allows for an easy way to include data from a text file, as long as the syntax in the text file can be understood by the GAMS compiler. This way, model specification and data input can be separated into different files.

The $include compile-time command is used to instruct the GAMS compiler to include the context of a different file at the current position of the input stream. As a result, the GAMS code behaves as if the `$include` statement has been replaced by the content of the file to be included. This can be very handy when including data from a separate text file. For instance, when data for a table is actually coming from another environment, one could replace the TABLE statement by an include statement. A GAMS table is in fact very well suited for a human being to be read or written, but it is rather awkward for programs to generate (e.g., the numbers have to be approximately below the corresponding headers). Therefore, often parameters are used and long series of assignment statements are generated. For instance, consider the following fragment from model [**TRNSPORT**]:

```
Table d(i,j)  'distance in thousands of miles'
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4  ;
```

When the data for this table is coming from a program it is more convenient to say in the main program:

```
Parameter d(i,j) 'distance in thousands of miles';
$include data.inc
display d;
```

and to have the include file `data.inc` contain the machine generated statements:

```
d("seattle","new-york") = 2.5;
d("san-diego","new-york") = 2.5;
d("seattle","chicago") = 1.7;
d("san-diego","chicago") = 1.8;
d("seattle","topeka") = 1.8;
d("san-diego","topeka") = 1.4;
```

In fact, GAMS can deal quite comfortably with a large number of such assignment statements.

Note, that since the included file is considered as part of the GAMS input stream, it is also echoed in the listing file. When including large text files with data statements, echoing these files in the listing file can be undesired. To suppress echoing to the listing file, the `$include` statement can be surrounded by $offlisting and $onlisting instructions:

```
parameter d(i,j) 'distance in thousands of miles';
$offlisting
$include data.inc
$onlisting
display d;
```

In the listing file, line numbers are skipped where the $offlisting is in effect.

In some cases it may be more convenient to use the initialization syntax of parameters. That is, the main GAMS file could contain the fragment:

```
parameter d(i,j) 'distance in thousands of miles' /
$include data2.inc
/;
display d;
```

and the data file contains the following records:

```
seattle  .new-york  2.5
san-diego.new-york  2.5
seattle  .chicago   1.7
san-diego.chicago   1.8
seattle  .topeka    1.8
san-diego.topeka    1.4
```

This approach is preferable for large data sets as it is more efficient for GAMS.

Note

Tables and parameters are handled exactly the same way by GAMS internally. The only difference is in the specification of data.

When using a table statement, data can also be specified in CSV (Comma-separated values) format if the $ondelim command has been issued. This format can, for instance, be generated by spreadsheet programs.

As example, consider again the following fragment from model [**TRNSPORT**]:

```
Table d(i,j)  'distance in thousands of miles'
                 new-york       chicago      topeka
    seattle         2.5           1.7          1.8
    san-diego       2.5           1.8          1.4  ;
```

A file `data.csv` that specifies the data of this table in CSV format would have the content

```
,new-york,chicago,topeka
seattle,2.5,1.7,1.8
san-diego,2.5,1.8,1.4
```

Notice the empty first element in the first line, which corresponds to the top-left blank in the above table.

This file can now be included directly into GAMS by using the $ondelim and $offdelim commands:

```
Table d(i,j) 'distance in thousands of miles'
$ondelim
$include data.csv
$offdelim
;
```

Attention

The $ondelim command only enables the use of *commas* as a separation symbol. The use of a different separation character is not supported by GAMS. This can become an issue if, for example, a CSV file has been generated with a language setting where commas are used as decimal point and semicolons are used for separating entries in the CSV file. In such a situation, `data.csv` would have the content

```
,new-york;chicago;topeka
seattle;2,5;1,7;1,8
san-diego;2,5;1,8;1,4
```

Therefore, such text files need to be preprocessed before read into GAMS. The following code uses the POSIX tool `tr` to (simultaneously) replace commas by dots and semicolons by commas:

```
$call "tr ,; ., < data.csv > data2.csv"
Table d(i,j) 'distance in thousands of miles' ;
$ondelim
$include data2.csv
$offdelim
```

CSV files can also be used to input higher dimensional data into GAMS. For instance, consider the 3 dimensional table `yieldtl` from model [**TURKEY**]:

```
Table yieldtl(l,cl,ty)  'livestock yield time series (kg per head)'

                      1974      1975      1976      1977      1978      1979

 sheep.meat           10.60     11.42     10.60      9.38      8.97      6.93
 sheep.milk           23.7      24.1      24.2      24.2      24.0      23.9
 sheep.wool            1.3       1.3       1.3       1.3       1.3       1.3
 sheep.hide            0.5       0.6       0.6       0.5       0.6       0.4
 goat.meat             6.39      7.31      8.68      7.31      6.39      6.85
 goat.milk            37.7      38.1      38.2      38.2      38.3      37.8
 goat.wool             0.6       0.6       0.6       0.6       0.6       0.6
 goat.hide             0.2       0.3       0.3       0.3       0.2       0.3
 angora.meat           1.77      1.77      2.66      2.21      1.77      1.77
 angora.milk          14.9      15.2      14.8      15.2      14.8      15.0
 angora.wool           1.6       1.6       1.6       1.6       1.6       1.4
 angora.hide           0.1       0.1       0.1       0.1       0.1       0.1
 cattle.meat          24.59     25.12     21.42     23.00     18.25     25.12
 cattle.milk         210.0     208.1     219.8     213.8     214.8     217.5
 cattle.hide           3.3       3.4       2.9       3.0       2.6       3.3
 buffalo.meat         43.73     45.42     40.61     37.21     32.20     32.68
 buffalo.milk        267.1     269.2     263.8     219.6     275.5     285.1
 buffalo.hide          4.1       3.4       3.0       2.4       2.5       2.6
 poultry.meat          2.24      2.24      2.24      2.24      2.24      2.24
 poultry.egg          62.4      62.2      64.2      78.3      76.4      73.3
;
```

When the data for this parameter is prepared by another application (such as a relational database), it may be more convenient to write it out in a comma-separated value form, e.g., a file `data.csv` could have the content

```
"sheep","meat","1974",10.60
"sheep","meat","1975",11.42
"sheep","meat","1976",10.60
```

```
"sheep","meat","1977",9.38
"sheep","meat","1978",8.97
"sheep","meat","1979",6.93
"sheep","milk","1974",23.70
"sheep","milk","1975",24.10
"sheep","milk","1976",24.20
"sheep","milk","1977",24.20
...
```

Including such formatted data into a GAMS model is possible by using $ondelim for Parameter Data for Higher Dimensions. Thus, the GAMS code would be

```
Parameter yieldtl(l,cl,ty)  'livestock yield time series (kg per head)'
/
$ondelim
$include data.csv
$offdelim
/;
```

4.18.1.2 Writing Text During Compilation

The commands $echo, $onecho, and $offecho send text to named files during compilation. $echo sends one line and is invoked using the syntax:

```
$echo 'text to be sent' > externalfile
```

or

```
$echo 'text to be sent' >> externalfile
```

The use of ">" generates a new file, while ">>" appends to an existing file.

For multi-line messages, the commands $onecho and $offecho can be used, e.g.,

```
$onecho  > externalfile
line 1 of text to be sent
line 2 of text to be sent
...
last line of text to be sent
$offecho
```

A typical example for the usage of these commands is the generation of a solver options file.

Additionally, the $log command can be used to send messages to the log file during compilation.

4.18.1.3 Writing Text Files During Execution

The put writing facility allows customized text output. This is a fairly complex but powerful and flexible report writing facility.

Assume that the following GAMS code is added to the end of model [**TRNSPORT**]. It instructs GAMS to write the model and the solve status together with levels of the decision variables to a file `results.txt`:

```
File results / results.txt /;
put results;
put "Model status",  transport.modelstat /;
put "Solver status", transport.solvestat /;
put "Objective", z.l /;
put "Shipments" /;
loop((i,j),
  put i.tl, j.tl, x.l(i,j) /
);
putclose;
```

First, a file object `results` is declared by using the File statement. The data of the file statement specifies the name of the file (`results.txt`). Next, line `put results;` instructs GAMS that for the following put statements, the file `results` should be used. In the following, the model status and the solve status are written, together with some descriptive text. These model attributes are set by a solve statement. The character '/' instructs GAMS to add a linebreak (newline character) to the results file. The next thing to write out are some variable values. Here, first the level value of variable `z` is written, followed by a loop that writes for each element of sets `i` and `j` the name of the elements (accessed via the .tl attribute) and the level value of `x(i,j)`. Finally, the `putclose;` statement instructs GAMS to close the current file. This will ensure that possibly cached data is flushed to the file.

The output will look like:

```
Model status        1.00
Solver status        1.00
Objective      153.67
Shipments
seattle     new-york           50.00
seattle     chicago           300.00
seattle     topeka              0.00
san-diego   new-york          275.00
san-diego   chicago             0.00
san-diego   topeka            275.00
```

This form can be hard to read by other applications, e.g., because some space characters are to be considered as separators, while others are really part of a string (e.g. "Model status"). However, by using the print control option `.pc` of the put writing facility, comma-separated value files can be written. That is, by adding the line `results.pc = 5;`, i.e.,

```
File results / results.txt /;
results.pc = 5;
put results;
put "Model status",  transport.modelstat /;
put "Solver status", transport.solvestat /;
put "Objective", z.l /;
put "Shipments" /;
loop((i,j),
  put i.tl, j.tl, x.l(i,j) /
);
putclose;
```

one obtains the following output:

```
"Model status",1.00
"Solver status",1.00
"Objective",153.67
"Shipments"
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

If several parameters of the same dimension should be written to a file in a customized format, the put statements can become rather repetitive. For example, the level and marginal values of variable `x` and the parameters `c` and `d` from model [**TRNSPORT**] should be written. This can be coded easily as:

```
file results / results.txt /;
results.pc = 5;
put results;
loop((i,j), put "distance",  i.tl, j.tl, d(i,j)   / );
loop((i,j), put "cost",      i.tl, j.tl, c(i,j)   / );
loop((i,j), put "levels",    i.tl, j.tl, x.l(i,j) / );
loop((i,j), put "marginals", i.tl, j.tl, x.m(i,j) / );
putclose;
```

A separation of the code that writes the data (loop and put statements) from the data that is written (descriptive text, set names, etc.) can be achieved by using $batinclude. This command works similar to the $include statement (see also Section Reading Text Files During Compilation above), but allows for additional arguments (separated by blanks). While including the file, markers `%1`, `%2`, etc., are replaced by the value of the 1st, 2nd, etc., argument.

By using $batinclude, the above example could be simplified to

```
file results / results.txt /;
results.pc = 5;
put results;
$batinclude put.inc distance i j d
$batinclude put.inc cost     i j c
$batinclude put.inc level    i j x.l
$batinclude put.inc marginal i j x.m
putclose;
```

where the file `put.inc` contain the actual loop and put statements:

```
loop((%2,%3), put "%1", %2.tl, %3.tl, %4(%2,%3) / );
```

Finally, the The Put_Utility Statement is referred, which is loosely connected to the put statement and allows the dynamic generation of file names, etc.

4.18.2 Data Exchange with Microsoft Excel

This tutorial gives an overview on how to exchange data between GAMS and Microsoft Excel.

GAMS can communicate with Microsoft Excel via GDX (GAMS Data Exchange) files. In order to write data from GAMS and to Excel, selected GAMS data can be written into a GDX file and then to an Excel file: GAMS -> GDX -> Excel. Similarly selected Excel data can be written to a GDX file and then read into GAMS: Excel -> GDX -> GAMS.

Some of GAMS/Excel data exchange tools that provide functionality to exchange data between GAMS and Excel are also discussed in the section Data Exchange Tools. The data exchange between GAMS and a CSV (Comma-separated values) file format and GAMS is covered in Data Exchange with Text Files.

4.18.2.1 From GAMS to Excel

Consider the following modication of the [**TRNSPORT**] model from the gams model library.

```
Sets
     i   'canning plants'   / seattle, san-diego /
     j   'markets'          / new-york, chicago, topeka / ;

Parameters

     a(i)  'capacity of plant i in cases'
       /    seattle     350
            san-diego   600  /

     b(j)  'demand at market j in cases'
       /    new-york    325
            chicago     300
            topeka      275  / ;

Table d(i,j)  'distance in thousands of miles'
                 new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4  ;

Scalar f  'freight in dollars per case per thousand miles'  /90/ ;

Parameter c(i,j)  'transport cost in thousands of dollars per case' ;

          c(i,j) = f * d(i,j) / 1000 ;

Variables
     x(i,j)  'shipment quantities in cases'
     z       'total transportation costs in thousands of dollars' ;

Positive Variable x ;

Equations
     cost        'define objective function'
     supply(i)   'observe supply limit at plant i'
     demand(j)   'satisfy demand at market j' ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
```

```
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;

Display x.l, x.m ;


*=== Export to Excel using GDX utilities

*=== First unload to GDX file (occurs during execution phase)
execute_unload "results.gdx" x.L x.M

*=== Now write to variable levels to Excel file from GDX
*=== Since we do not specify a sheet, data is placed in first sheet
execute 'gdxxrw.exe results.gdx o=results.xlsx var=x.L'

*=== Write marginals to a different sheet with a specific range
execute 'gdxxrw.exe  results.gdx o=results.xlsx var=x.M rng=NewSheet!f1:i4'
```

After the solve statement, the data (`x.L` and `x.M`) from variable `x` can be written into a GDX file during the execution time using the command execute_unload:

```
execute_unload "results.gdx" x.L x.M
```

The execute_unload command above is executed during the actual execution phase to create a GDX file called `results.gdx`. The solution `x` and the marginals of `x` in the GDX file can be written to the Excel file `results.xlsx` using GDXXRW tool:

```
execute 'gdxxrw.exe results.gdx var=x.L'
execute 'gdxxrw.exe results.gdx var=x.M rng=NewSheet!f1:i4'
```

For the first call for `x.L`, there is no range specified and the data is written in cell `A1` and beyond in the first available sheet. For the second call for marginals `x.M`, data will be written to cells `F1:I4` in the sheet named `NewSheet`.

Note that GAMS can also write data into a GDX file during compile time. It is also possible to convert data stored in a GDX file into an Excel file spreadsheets using GDX2XLS tool and to write GAMS data to standard output formatted as a GAMS program with data statements using GDXDUMP tool.

4.18.2.2 From Excel to GAMS

Consider the following modifciation of the [**TRNSPORT**] model from the gams model library and the file `results.xlsx` file created from the previous example.

```
Sets
     i   'canning plants'   / seattle, san-diego /
     j   'markets'          / new-york, chicago, topeka / ;

Parameters
```

```
     a(i)  'capacity of plant i in cases'
       /    seattle     350
            san-diego   600  /

     b(j)  'demand at market j in cases'
       /    new-york    325
            chicago     300
            topeka      275  / ;

Table d(i,j)  'distance in thousands of miles'
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4  ;

Scalar f  'freight in dollars per case per thousand miles'  /90/ ;

Parameter c(i,j)  'transport cost in thousands of dollars per case' ;

          c(i,j) = f * d(i,j) / 1000 ;

Variables
     x(i,j)  'shipment quantities in cases'
     z       'total transportation costs in thousands of dollars' ;

Positive Variable x ;

Equations
     cost        'define objective function'
     supply(i)   'observe supply limit at plant i'
     demand(j)   'satisfy demand at market j' ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;

demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;


*=== Import from Excel using GDX utilities

*=== First unload to GDX file (occurs during compilation phase)
$call gdxxrw.exe results.xlsx par=Level rng=sheet1!A1:D3

*=== Now import data from GDX
Parameter Level(i,j);
$gdxin results.gdx
$load Level
$gdxin

*=== Fix variables to values from Excel file
x.FX(i,j) = Level(i,j);
display Level, x.L;


Solve transport using lp minimizing z ;
```

```
Display x.l, x.m ;
```

The data in the Excel file can be loaded into a GDX file using the $call command and GDXXRW tool:

```
$call gdxxrw.exe results.xlsx par=Level rng=A1:D3
```

The command $call above executes a program called GDXXRW during compile time. The GDXXRW reads data from the range `A1:D3` in `results.xlsx` into a GAMS parameter called `Level` in the GDX file `results.gdx`. As an output GDX file is not specified when calling GDXXRW, the output file will be derived from the input file by changing the file extension of the input file and removing any path information.

To import data from a GDX file into a parameter, the parameter must be defined over appropriate sets before read. The data from a GDX file can be read during the compile time using the commands $gdxin and $load:

```
Parameter Level(i,j);
$gdxin results.gdx
$load Level
$gdxin
```

The first command $gdxin specifies the name of the GDX file `results.gdx` to be read. The command $load reads parameter `Level` from the GDX file. The second command $gdxin closed the GDX file.

GAMS can read from a GDX file either during compile time or during execution time. See Example 4 - Reading a GDX File when reading data with domain information and Example 5 - Reading a GDX File when reading from a GDX file during execution time.

Note that it is also possible to write all worksheets of an Excel workbook into a GDX file using XLSDUMP tool.

4.18.2.3 Data Exchange Tools

There are a number of tools that provide functionality to exchange data between GAMS and an Excel file. This section discusses some of the data exchange tools with some examples. The complete list of the tools can be found at GAMS/Excel Data Exchange tools.

GDXXRW

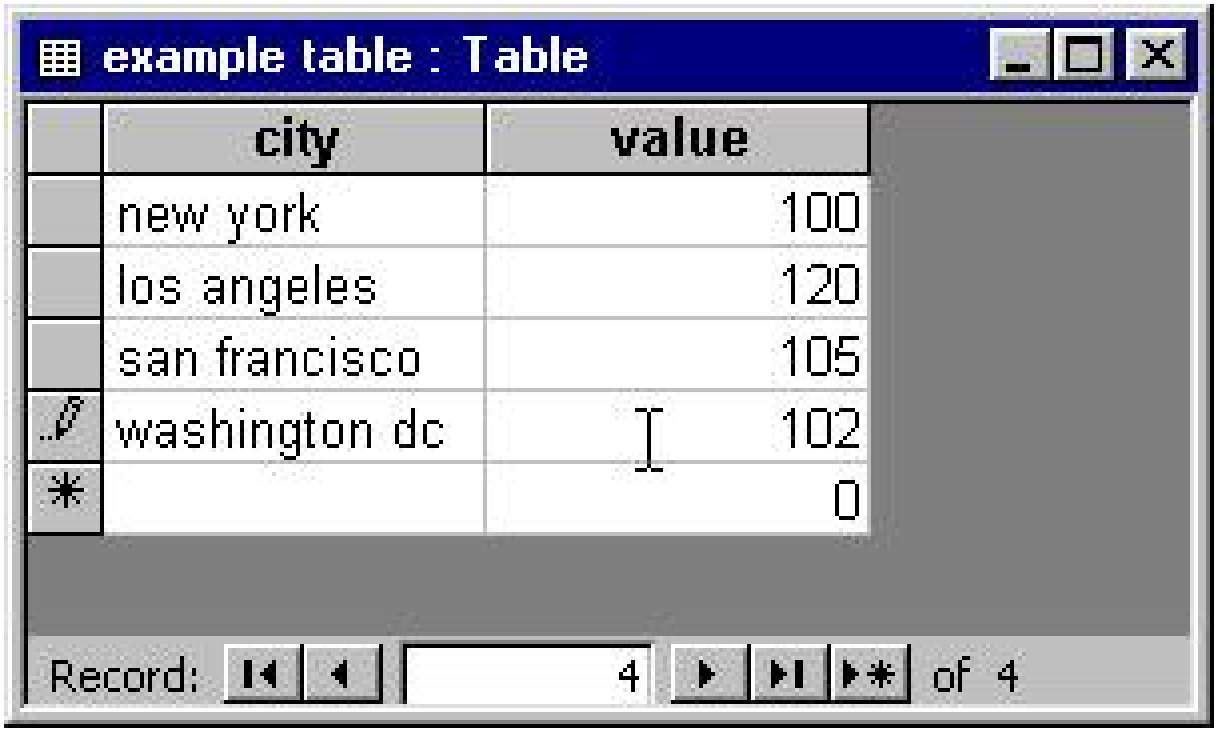
GDXXRW is a tool to read and write Excel spreadsheet data. GDXXRW can read multiple ranges in a spreadsheet and write the data to a 'GDX' file, or read from a 'GDX' file, and write the data to different ranges in a spreadsheet.

How to use GDXXRW to exchange data between GAMS and Excel is covered in the section From GAMS to Excel and the section From Excel to GAMS. More details on usage and examples of GDXXRW tool is covered in GDXXRW.

XLS2GMS

XLS2GMS is a simple utility that allows you to extract data from an Excel spreadsheet and convert it into a GAMS include file. XLS2GMS can be run interactively or in batch mode.

Consider the Excel data from the following spreadsheet:



The data can be imported from the Excel file into a GAMS include file by calling XLS2GMS tool and inserted an include file as parameter data elements using the command $include:

```
set ssi /
  'new york', 'washington dc', 'los angeles', 'san francisco'
/;
parameter ssdata(ssi) /
$call =d:\util\xls2gms I="c:\my documents\test2.xlsx" B O=d:\tmp\x.inc
$include d:\tmp\x.inc
/;
display ssdata;
```

Notice the B parameter, which is needed as there are embedded blanks in the labels.

Sometimes a translation between the labels used in the model and the ones used in the is needed. One way to do this is to use a mapping set in GAMS. Suppose the rest of the model is defined in terms of the set I which is defined as:

```
set i / ny, dc, la, sf/;
```

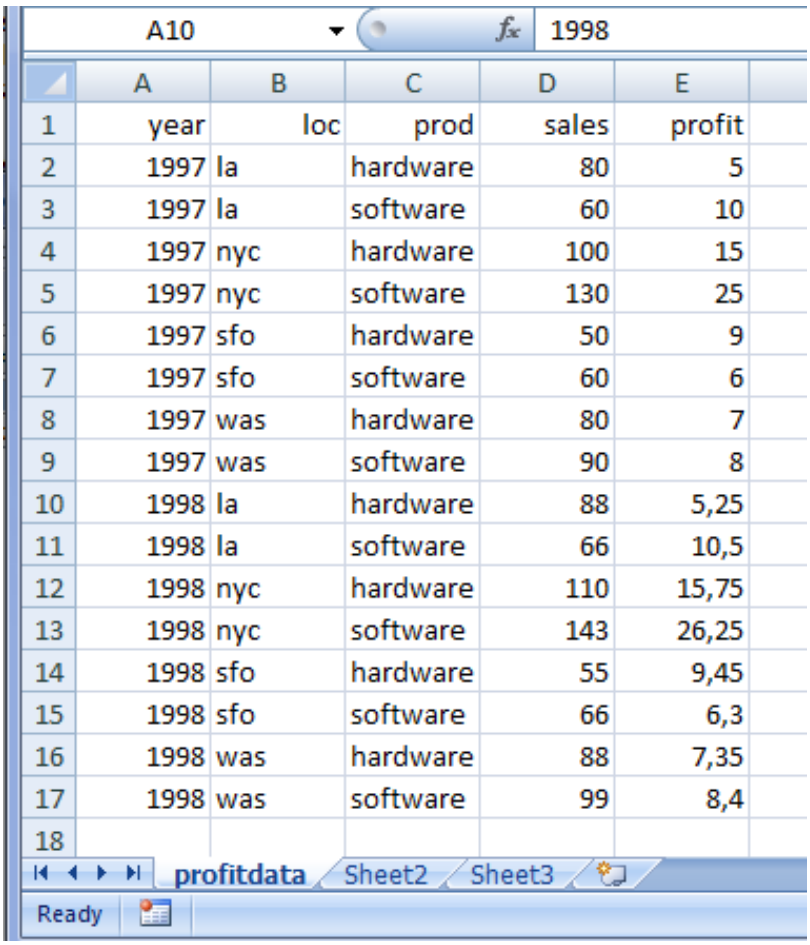
To map a parameter data defined over this set, the following simple GAMS fragment can be used:

```
set map(i,ssi) mapping set /
   ny.'new york'
   dc.'washington dc'
   la.'los angeles'
   sf.'san francisco'
/;
display map;

parameter data(i);
data(i) = sum(map(i,ssi), ssdata(ssi));
display data;
```

SQL2GMS

In some cases it is convenient to consider tabular data in an Excel spreadsheet as a database table and to import it via GDX file using the SQL2GMS tool.

Consider the following spreadsheet:



	A	B	C	D	E
1	year	loc	prod	sales	profit
2	1997	la	hardware	80	5
3	1997	la	software	60	10
4	1997	nyc	hardware	100	15
5	1997	nyc	software	130	25
6	1997	sfo	hardware	50	9
7	1997	sfo	software	60	6
8	1997	was	hardware	80	7
9	1997	was	software	90	8
10	1998	la	hardware	88	5,25
11	1998	la	software	66	10,5
12	1998	nyc	hardware	110	15,75
13	1998	nyc	software	143	26,25
14	1998	sfo	hardware	55	9,45
15	1998	sfo	software	66	6,3
16	1998	was	hardware	88	7,35
17	1998	was	software	99	8,4
18					

This table can be read using an SQL query:

```
SELECT year,loc,prod,'sales',sales FROM [profitdata$] \
  UNION SELECT year,loc,prod,'profit',profit FROM [profitdata$]
```

The table name is equal to the sheet name(profitdata). We can pass the query to the Excel ODBC driver using the tool SQL2GMS tool as follows:

```
set y 'years'   /1997,1998/;
set c 'city'    /la,nyc,sfo,was/;
set p 'product' /hardware,software/;
set k 'key'     /sales,profit/;

$onecho > excelcmd.txt
c=DRIVER=Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb);dbq=%system.fp%profit.xlsx;
q=SELECT year,loc,prod,'sales',sales FROM [profitdata$] \
  UNION SELECT year,loc,prod,'profit',profit FROM [profitdata$]
x=fromexcel.gdx
$offecho
$call =sql2gms @excelcmd.txt
parameter d(y,c,p,k) ;
$gdxin excel.gdx
$load d=p
display d;
```

and the DISPLAY results will be:

```
---     21 PARAMETER d  FROM SQL2GMS

INDEX 1 = 1997

                   sales      profit

la .hardware      80.000       5.000
la .software      60.000      10.000
nyc.hardware     100.000      15.000
nyc.software     130.000      25.000
sfo.hardware      50.000       9.000
sfo.software      60.000       6.000
was.hardware      80.000       7.000
was.software      90.000       8.000

INDEX 1 = 1998

                   sales      profit

la .hardware      88.000       5.250
la .software      66.000      10.500
nyc.hardware     110.000      15.750
nyc.software     143.000      26.250
sfo.hardware      55.000       9.450
sfo.software      66.000       6.300
was.hardware      88.000       7.350
was.software      99.000       8.400
```

GDXVIEWER

GDXVIEWER is a tool to view and convert data contained in GDX files. It can also export to csv, xlsx, xml-files and pivot tables. The usage and examples are covered in GDXVIEWER.

GDX2XLS

GDX2XLS tool to convert the contents of a GDX file into an Excel file or an xml-file. The usage and examples are covered in GDX2XLS.

4.18.3 Data Exchange with Databases

This tutorial provides a guidance on how to exchange data between GAMS and various Database Management System.

4.18.3.1 Data Exchange with DB2

DB2 is one of IBM's relational database management systems.

Import from DB2

DB2 has an `EXPORT` command that can be used to generate comma delimited files. An example of a DB2 session illustrating this is shown below:

```
------------------------------------ Command Entered -------------------------------------
describe table db2admin.dist
 ;
-----------------------------------------------------------------------------------------

Column                         Type      Type
name                           schema    name               Length   Scale Nulls
------------------------------ --------- ------------------ -------- ----- -----
LOCA                           SYSIBM    VARCHAR                 10     0 No
LOCB                           SYSIBM    VARCHAR                 10     0 No
DISTANCE                       SYSIBM    DOUBLE                   8     0 Yes

  3 record(s) selected.


---------------------------------- Command Entered ---------------------------------------
select * from dist ;
-----------------------------------------------------------------------------------------

LOCA       LOCB       DISTANCE
---------- ---------- ------------------------
seattle    new-york     +2.50000000000000E+000
seattle    chicago      +1.70000000000000E+000
seattle    topeka       +1.80000000000000E+000
san-diego  new-york     +2.50000000000000E+000
san-diego  chicago      +1.80000000000000E+000
san-diego  topeka       +1.40000000000000E+000

  6 record(s) selected.
```

```
---------------------------------- Command Entered ----------------------------------
export to c:\tmp\export.txt of del select * from dist ;
-------------------------------------------------------------------------------------
SQL3104N  The Export utility is beginning to export data to file
"c:\tmp\export.txt".

SQL3105N  The Export utility has finished exporting "6" rows.


Number of rows exported: 6
```

The resulting data file **export.txt** will look like:

```
"seattle","new-york",+2.50000000000000E+000
"seattle","chicago",+1.70000000000000E+000
"seattle","topeka",+1.80000000000000E+000
"san-diego","new-york",+2.50000000000000E+000
"san-diego","chicago",+1.80000000000000E+000
"san-diego","topeka",+1.40000000000000E+000
```

This file can be read into GAMS using $include :

```
parameter d(i,j) 'distance in thousands of miles' /
  $ondelim
  $include export.txt
  $offdelim
/;
display d;
```

Export to DB2

DB2 has an `IMPORT` command that can read delimited files. As an example consider the file generated by GAMS PUT statements:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

A transcript of a DB2 session to read this file, is given below:

```
---------------------------------- Command Entered ----------------------------------
create table results(loca varchar(10) not null,
                     locb varchar(10) not null,
                     shipment double not null)  ;
-------------------------------------------------------------------------------------
DB20000I  The SQL command completed successfully.


---------------------------------- Command Entered ----------------------------------
import from c:\tmp\import.txt of del insert into results ;
```

```
-----------------------------------------------------------------------------------------
SQL3109N  The utility is beginning to load data from file "c:\tmp\import.txt".

SQL3110N  The utility has completed processing.  "6" rows were read from the
input file.

SQL3221W  ...Begin COMMIT WORK. Input Record Count = "6".

SQL3222W  ...COMMIT of any database changes was successful.

SQL3149N  "6" rows were processed from the input file.  "6" rows were
successfully inserted into the table.  "0" rows were rejected.


Number of rows read         = 6
Number of rows skipped      = 0
Number of rows inserted     = 6
Number of rows updated      = 0
Number of rows rejected     = 0
Number of rows committed    = 6
```

For very large data sets it is advised to use the `LOAD` command:

```
------------------------------------ Command Entered ------------------------------------
load from c:\tmp\import.txt of del insert into results ;
-----------------------------------------------------------------------------------------
SQL3501W  The table space(s) in which the table resides will not be placed in
backup pending state since forward recovery is disabled for the database.

SQL3109N  The utility is beginning to load data from file "c:\tmp\import.txt".

SQL3500W  The utility is beginning the "LOAD" phase at time "03-20-2000
18:11:50.213782".

SQL3519W  Begin Load Consistency Point. Input record count = "0".

SQL3520W  Load Consistency Point was successful.

SQL3110N  The utility has completed processing.  "6" rows were read from the
input file.

SQL3519W  Begin Load Consistency Point. Input record count = "6".

SQL3520W  Load Consistency Point was successful.

SQL3515W  The utility has finished the "LOAD" phase at time "03-20-2000
18:11:50.337092".


Number of rows read         = 6
Number of rows skipped      = 0
Number of rows loaded       = 6
Number of rows rejected     = 0
Number of rows deleted      = 0
Number of rows committed    = 6
```

For smaller data sets one can also generate a series of INSERT statements using the PUT facility.

4.18.3.2 Data Exchange with MS Access

Microsoft Office Access, previously known as Microsoft Access, is a relational database management system from Microsoft. It is a member of the Microsoft Office system.

Import from MS Access

MDB2GMS

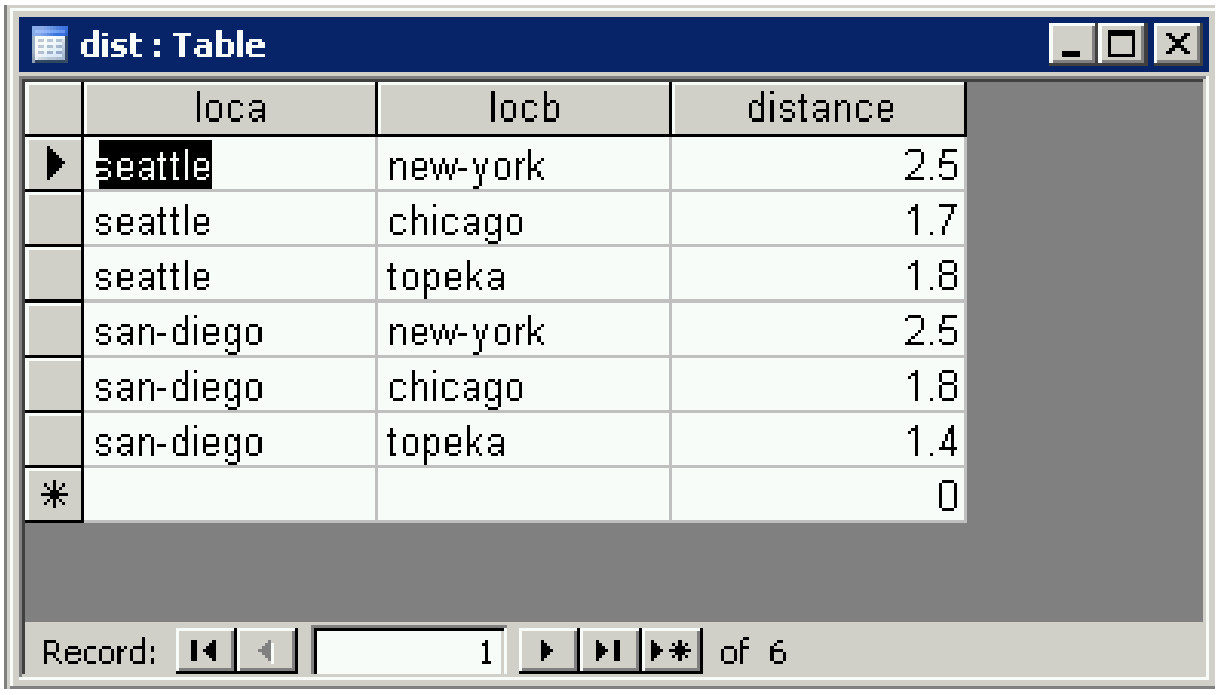
MDB2GMS is a tool to convert data from an Microsoft Access database into GAMS readable format. The source is an MS Access database file (∗.mdb or ∗.accdb) and the target is a GAMS Include File or a GAMS GDX File. MDB2GMS is part of the GAMS Data eXchange Tools, see documentation for more information.

SQL2GMS

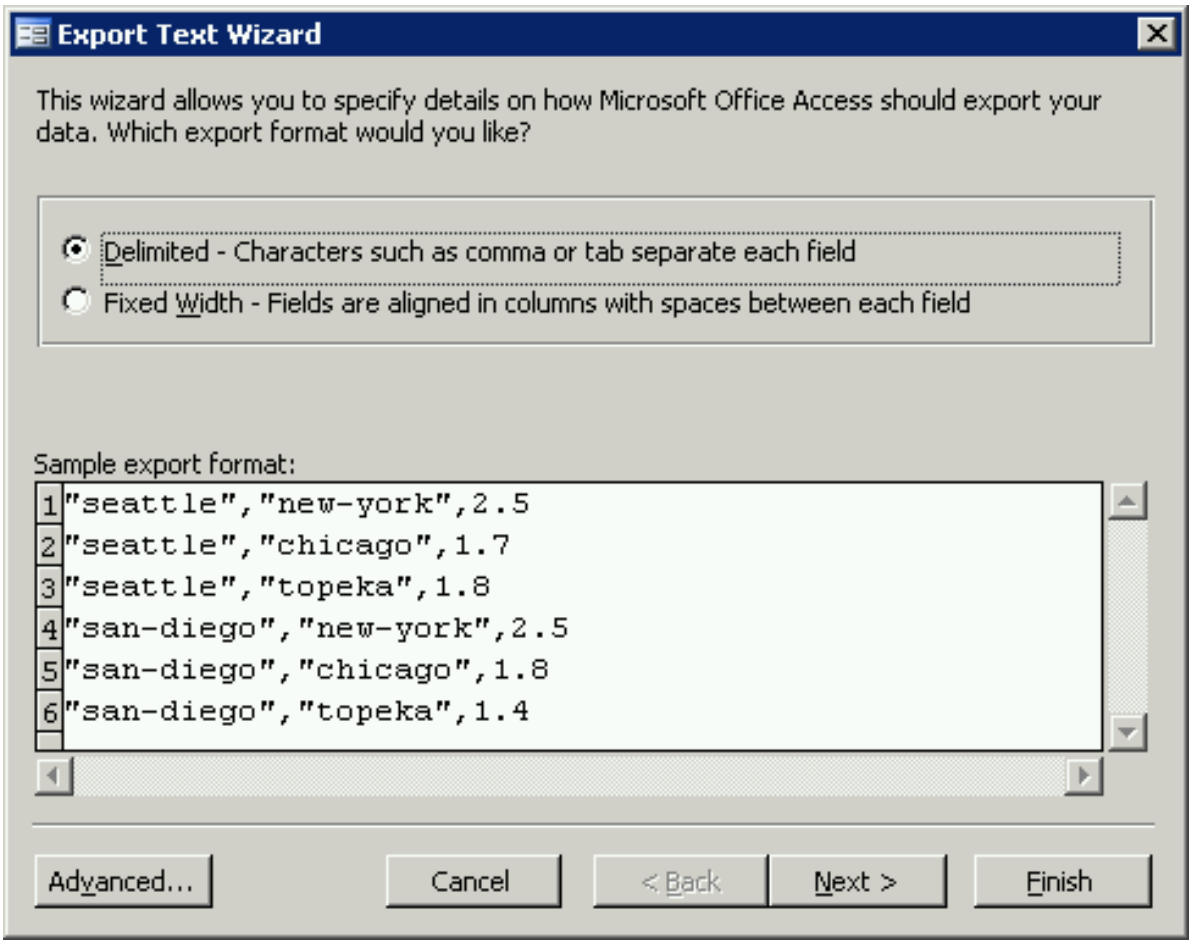
SQL2GMS is a tool to convert data from an SQL database into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including ADO, ODBC and OLEDB. The target is a GAMS Include File or a GAMS GDX File. SQL2GMS is part of the GAMS Data eXchange Tools, see documentation for more information.

CSV Files

Microsoft Access can export tables into comma delimited text files using its **Save As/Export** menu. Suppose we have the following table:



After choosing **Save As/Export** and selecting **Text Files** we get the following window:



Just using the default settings, we get the following file:

```
"seattle","new-york",2.50
"seattle","chicago",1.70
"seattle","topeka",1.80
"san-diego","new-york",2.50
"san-diego","chicago",1.80
"san-diego","topeka",1.40
```

which can be handled in GAMS by $ondelim/$offdelim and $include:

```
parameter d(i,j) 'distance in thousands of miles' /
    $ondelim
    $include dist.txt
    $offdelim
/;
display d;
```

Import Dates from Access

GAMS dates are one day off when importing from MS Access. Suppose we have an MS Access table with one single date column:

```
datefield
----------
3/12/2007
3/13/2007 10:00:00 AM
3/14/2007 8:30:00 PM
```

The date data above can be imported into GAMS using $call and MDB2GMS tool as follows:

```
$call =mdb2gms I="%system.fp%Sample.accdb" Q="select datefield,Cdbl(datefield) from datetable" O=x.i
parameter p(*) /
    $include x.inc
/;
display p;
alias(*,i);
parameter q(*,*);
loop(i$p(i),
    q(i,'year') = gyear(p(i));
    q(i,'month') = gmonth(p(i));
    q(i,'day') = gday(p(i));
    q(i,'hour') = ghour(p(i));
    q(i,'minute') = gminute(p(i));
);
display q;
```

Note that the `Cdbl()` function converts the date to a floating point number (double precision). The generated include file looks like:

```
* ------------------------------------------------------
* MDB2GMS Version 2.8, January 2007
* Erwin Kalvelagen, GAMS Development Corp
* ------------------------------------------------------
* DAO version: 3.6
* Jet version: 4.0
* Database:    D:\mdb2gms\examples\Sample.accdb
* Query:       select datefield,Cdbl(datefield) from datetable
* ------------------------------------------------------
'3/12/2007' 39153
'3/13/2007 10:00:00 AM' 39154.4166666667
'3/14/2007 8:30:00 PM' 39155.8541666667
* ------------------------------------------------------
```

which looks o.k. However, when we look at the GAMS results in the listing file we see:

```
----     28 PARAMETER p
3/12/2007             39153.000,    3/13/2007 10:00:00 AM 39154.417,    3/14/2007 8:30:00 PM  39155.8

----     39 PARAMETER q

                            year       month         day        hour      minute

3/12/2007                2007.000       3.000      13.000
3/13/2007 10:00:00 AM    2007.000       3.000      14.000      10.000
3/14/2007 8:30:00 PM     2007.000       3.000      15.000      20.000      30.000
```

Clearly the dates are off by one day: see the column `day`. We can fix this problem in different places, e.g. in the query or in the GAMS model by subtracting 1.0 from an imported date. This problem occurs not only in MS Access but also with other software packages.

Export to MS Access

GDX2ACCESS

GDX2ACCESS is a tool to dump the whole contents of a GDX file to a **new** MS Access file (.mdb or .accdb file). GDX2ACCESS is part of the GAMS Data eXchange Tools, see documentation for more information.

GDXVIEWER

Access tables in MS Access files can be directly generated by the GDXVIEWER tool. The GDXVIEWER tool uses OLE automation to export data to an MS Access database. This means that MS Access needs to be installed for the Access Export facility to work. GDXVIEWER is part of the GAMS Data eXchange Tools, see documentation for more information.

VBScript

VBScript is a scripting tool that can be used to talk to COM objects. In this case we use it to tell Access to import a CSV file.

```
$ontext
    Import a table into MS Access using VBscript
$offtext
$if exist new.accdb $call del new.accdb
set i /i1*i10/;
alias (i,j);
parameter p(i,j);
p(i,j) = uniform(-100,100);
display p;
file f /data.csv/;
f.pc=5;
put f,'i','j','p'/;
loop((i,j),
     put i.tl, j.tl, p(i,j):12:8/
);
putclose;

execute "=cscript access.vbs";

$onecho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"
dbLangGeneral = ";LANGID=0x0409;CP=1252;COUNTRY=0"
strSQL = "SELECT * INTO mytable FROM [Text;HDR=Yes;Database=%system.fp%;FMT=Delimited].[data#csv]"
Wscript.Echo "Query : " & strSQL
Set oJet = CreateObject("DAO.DBEngine.36")
Wscript.Echo "Jet version : " & oJet.version
Set oDB = oJet.createDatabase("new.accdb",dbLangGeneral)
Wscript.Echo "Created : " & oDB.name
oDB.Execute strSQL
Set TableDef = oDB.TableDefs("mytable")
Wscript.Echo "Rows inserted in mytable : " & TableDef.RecordCount
oDB.Close
Wscript.Echo "Done"
$offecho
```

The CSV file contains a header row with the names of the fields:

```
"i","j","p"
"i1","i1",-65.65057360
"i1","i2",68.65334160
"i1","i3",10.07507120
"i1","i4",-39.77241920
"i1","i5",-41.55757660
....
```

The text driver specification `HDR=Yes` makes sure the first row in the CSV file is treated specially. The log will look like:

```
U:\temp>gams vbaccess.gms
--- Job vbaccess.gms Start 01/28/08 16:57:37
GAMS Rev 149  Copyright (C) 1987-2007 GAMS Development. All rights reserved
...
--- Starting compilation
--- vbaccess.gms(4) 2 Mb
--- call del new.accdb
--- vbaccess.gms(38) 3 Mb
--- Starting execution: elapsed 0:00:00.109
--- vbaccess.gms(18) 4 Mb
Microsoft (R) Windows Script Host Version 5.6
Copyright (C) Microsoft Corporation 1996-2001. All rights reserved.

Running script: access.vbs
Query : SELECT * INTO mytable FROM [Text;HDR=Yes;Database=U:\temp\;FMT=Delimited
].[data#csv]
Jet version : 3.6
Created : U:\temp\new.accdb
Rows inserted in mytable : 100
Done
--- Putfile f U:\temp\data.csv
*** Status: Normal completion
--- Job vbaccess.gms Stop 01/28/08 16:57:38 elapsed 0:00:00.609
U:\temp>
```

Please note that although the $onecho/$offecho is at the bottom of the GAMS file, the file `access.vbs` is created at compile time. I.e. before the executable statements like `PUT`, `EXECUTE` are executed.

JScript

The same script using `JScript` is similar to the one with VScript. We only price the script itself.

```
$ontext
    Import a table into MS Access using JScript
$offtext
$if exist new.accdb $call del new.accdb

set i /i1*i10/;
alias (i,j);
parameter p(i,j);
```

```
p(i,j) = uniform(-100,100);
display p;

file f /data.csv/;
f.pc=5;
put f,'i','j','p'/;
loop((i,j),
     put i.tl, j.tl, p(i,j):12:8/
);
putclose;

execute "=cscript access.js";

$onecho > access.js
// this is a JScript script
WScript.Echo("Running script: access.js");

dbLangGeneral = ";LANGID=0x0409;CP=1252;COUNTRY=0";
strSQL = "SELECT * INTO mytable FROM [Text;HDR=Yes;Database=.;FMT=Delimited].[data#csv]";
WScript.Echo("Query : ",strSQL);

oJet = new ActiveXObject("DAO.DBEngine.36");
WScript.Echo("Jet version : ",oJet.version);

oDB = oJet.createDatabase("new.accdb",dbLangGeneral);
WScript.Echo("Created : ",oDB.name);

oDB.Execute(strSQL);
TableDef = oDB.TableDefs("mytable");
WScript.Echo("Rows inserted in mytable : ",TableDef.RecordCount);

oDB.Close();

WScript.Echo("Done");
$offecho
```

Combining GDX2ACCESS and VBscript

Data in a GDX file do not contain domain information. I.e. a parameter `c(i,j)` is really stored as `c(*,*)`. As a result GDX2ACCESS will invent field names like `dim1`, `dim2`, `Value`. In some cases this may not be convenient, e.g. when more descriptive field names are required. We will show how a small script in VBscript can handle this task. The script will rename the fields `dim1`, `dim2`, `Value` in table `c` to `i`, `j`, and `transportcost`.

```
$call "gamslib 1"
$include trnsport.gms
*
* export to gdx file.
* The domains i,j are lost: gdx only stores c(*,*)
execute_unload "c.gdx",c;
*
* move to access database
* column names are dim1,dim2
*
execute "=gdx2access c.gdx";
*
```

```
* rename columns
*
execute "=cscript access.vbs";

$onecho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"

' Office 2000 DAO version
' Change to local situation.
Set oDAO = CreateObject("DAO.DBEngine.36")
script.Echo "DAO version : " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%c.accdb")
Wscript.Echo "Opened : " & oDB.name

Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name

' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"

oDB.Close
Wscript.Echo "Done"
$offecho
```

The above VBscript fragment needs to be adapted according to the DAO `Data Access Objects` version available on the client machine. This can be implemented in a more robust fashion by letting MS Access find the DAO engine:

```
'this is a VBscript script
WScript.Echo "Running script: access.vbs"

set oa = CreateObject("Access.Application")
set oDAO = oa.DBEngine
Wscript.Echo "DAO Version: " & oDAO.version

Set oDB = oDAO.openDatabase("%system.fp%c.accdb")
Wscript.Echo "Opened : " & oDB.name

Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name

' rename fields
oTable.Fields.Item("dim1").name = "i"
oTable.Fields.Item("dim2").name = "j"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"

oDB.Close

Wscript.Echo "Done"
```

Please note that the macro `%system.fp%` is replaced by GAMS by the working directory (this is the project directory when running GAMS from the IDE).

4.18.3.3 Data Exchange with MySQL

MySQL is a multi-threaded, multi-user SQL database management system.

Import from MySQL

MySQL can write the results of a SELECT statement to a file as follows:

```
mysql> select * from dist;
+-----------+----------+----------+
| loca      | locb     | distance |
+-----------+----------+----------+
| seattle   | new-york |       50 |
| seattle   | chicago  |      300 |
| seattle   | topeka   |        0 |
| san-diego | new-york |      275 |
| san-diego | chicago  |        0 |
| san-diego | topeka   |      275 |
+-----------+----------+----------+
6 rows in set (0.01 sec)

mysql> select * from dist into outfile '/tmp/data.csv'
    -> fields terminated by ','
    -> optionally enclosed by '"'
    -> lines terminated by '\n';
Query OK, 6 rows affected (0.00 sec)
```

The resulting CSV file looks like:

```
"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275
```

which can be read by GAMS directly. This approach can be automated as follows:

```
[erwin@localhost erwin]$ cat myscript
use test
select * from dist into outfile '/tmp/data.csv'
     fields terminated by ','
     optionally enclosed by '"'
     lines terminated by '\n';
[erwin@localhost erwin]$ cat x.gms
set i /seattle, san-diego/;
set j /new-york, chicago, topeka/;

$call 'mysql -u root < myscript'
parameter dist(i,j) /
$ondelim
$include /tmp/data.csv
$offdelim
```

```
/;
display dist;
[erwin@localhost erwin]$ gams x
GAMS Rev 132  Copyright (C) 1987-2002 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC   G871201:0000XX-XXX
          Free Demo,  202-342-0180,  sales@gams.com,  www.gams.com   DC9999
--- Starting compilation
--- x.gms(5) 1 Mb
--- call mysql -u root < myscript
--- .data.csv(6) 1 Mb
--- x.gms(15) 1 Mb
--- Starting execution
--- x.gms(18) 1 Mb
*** Status: Normal completion
[erwin@localhost erwin]$
```

The listing file shows that the table is read correctly:

```
   1
   2  set i /seattle, san-diego/;
   3  set j /new-york, chicago, topeka/;
   4
   6  parameter dist(i,j) /
INCLUDE    /tmp/data.csv
   9  "seattle","new-york",50
  10  "seattle","chicago",300
  11  "seattle","topeka",0
  12  "san-diego","new-york",275
  13  "san-diego","chicago",0
  14  "san-diego","topeka",275
  16  /;
  17
  18  display dist;
  19
  20
  21

   SEQ   GLOBAL TYPE      PARENT   LOCAL  FILENAME

     1        1 INPUT          0       0  /home/erwin/x.gms
     2        5 CALL           1       5  mysql -u root < myscript
     3        8 INCLUDE        1       8  ./tmp/data.csv



----     18 PARAMETER dist

             new-york     chicago      topeka

seattle        50.000     300.000
san-diego     275.000                 275.000
```

Instead of maintaining the MySQL script in a separate file, it can also be written by GAMS using $onecho/$offecho and a statement like:

```
$onecho > myscript
```

```
use test
select * from dist into outfile '/tmp/data.csv'
     fields terminated by ','
     optionally enclosed by '"'
     lines terminated by '\n';
$offecho
```

This will write the script at compile time.

Export to MySQL

GAMS can export data to MySQL by creating a script containing a series of SQL `INSERT` statements, as shown in section Oracle CSV Import .

It is noted that MySQL does have a REPLACE statement which is a useful blend of an `INSERT` and `UPDATE` statement: update a row if it already exists, otherwise insert it. This is not standard SQL however, so it can cause problems when moving to another database.

For larger result sets it may be better to use the `LOAD DATA INFILE` command. This command can read directly ASCII text files such as comma delimited CSV files.

Consider again the data file created by the PUT statement:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

The following transcript shows how to import this into MySQL:

```
myql> create table dist(loca varchar(10), locb varchar(10), distance double precision);
Query OK, 0 rows affected (0.00 sec)

mysql> show tables;
+----------------+
| Tables_in_test |
+----------------+
| dist           |
+----------------+
1 row in set (0.00 sec)

mysql> describe dist;
+----------+-------------+------+-----+---------+-------+
| Field    | Type        | Null | Key | Default | Extra |
+----------+-------------+------+-----+---------+-------+
| loca     | varchar(10) | YES  |     | NULL    |       |
| locb     | varchar(10) | YES  |     | NULL    |       |
| distance | double      | YES  |     | NULL    |       |
+----------+-------------+------+-----+---------+-------+
3 rows in set (0.00 sec)

mysql> load data infile '/tmp/data.txt' into table dist
    -> fields terminated by ','
```

```
    -> optionally enclosed by '"'
    -> lines terminated by '\n';
Query OK, 6 rows affected (0.00 sec)
Records: 6  Deleted: 0  Skipped: 0  Warnings: 0

mysql> select * from dist;
+-----------+----------+----------+
| loca      | locb     | distance |
+-----------+----------+----------+
| seattle   | new-york |       50 |
| seattle   | chicago  |      300 |
| seattle   | topeka   |        0 |
| san-diego | new-york |      275 |
| san-diego | chicago  |        0 |
| san-diego | topeka   |      275 |
+-----------+----------+----------+
6 rows in set (0.00 sec)
```

Note that we used no keys in our table definition. In practice it is of course highly recommended to define proper keys.

4.18.3.4 Data Exchange with Oracle

The Oracle Database (commonly referred to as Oracle RDBMS or simply as Oracle) is a relational database management system (RDBMS) software product released by Oracle Corporation.

Import from Oracle

SQL∗Plus

To export an Oracle table a simple solution is to write an SQL∗Plus script. E.g. if our table looks like:

```
SQL> describe dist;
 Name                              Null?    Type
 --------------------------------- -------- ------------------------
 LOCA                              NOT NULL VARCHAR2(10)
 LOCB                              NOT NULL VARCHAR2(10)
 DISTANCE                                   NUMBER

SQL> select * from dist;

LOCA       LOCB        DISTANCE
---------- ---------- ---------
seattle    new-york         2.5
seattle    chicago          1.7
seattle    topeka           1.8
san-diego  new-york         2.5
san-diego  chicago          1.8
san-diego  topeka           1.4

6 rows selected.

SQL>
```

then the following script will export this table:

```
set pagesize 0
set pause off
set heading off
spool data
select loca||',',||locb||',',||distance from dist;
spool off
```

The resulting data file "data.lst" will look like:

```
seattle,new-york,2.5
seattle,chicago,1.7
seattle,topeka,1.8
san-diego,new-york,2.5
san-diego,chicago,1.8
san-diego,topeka,1.4
```

This almost looks like our data initialization syntax for parameters:

```
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
```

The only differences are in the delimiters that are being used. These differences are easily digested by GAMS once it is in ondelim mode. I.e. the following syntax can be used to read the *data.lst* file:

```
parameter d(i,j) 'distance in thousands of miles' /
$ondelim
$include data.lst
$offdelim
/;
display d;
```

SQL2GMS

An alternative way to import data from Oracle is to use the tool SQL2GMS which can talk to any database with an ADO or ODBC interface.

Import dates from Oracle databases and converting them to GAMS dates

For most softwares it is easy to generate dates that GAMS can import and understand. The most common issue is that GAMS is one day off compared to Excel, Delphi, Access, ODBC etc. Oracle is somewhat more involved. First it is useful to have the date/time exported as a Julian date. This can be done with the following stored procedure:

```
-- julian representation of a date/time
-- Erwin Kalvelagen, feb 2007
create or replace function to_julian(d IN TIMESTAMP)
return number
is
begin
return to_number(to_char(d,'J')) + to_number(to_char(d,'SSSSS'))/86400;
end;
```

This function can be used to export dates as simple floating point numbers. In GAMS we need just a simple adjustment by adding a constant `''datediff''` defined by:

```
scalar
   refdategams   "march 16, 2006, 00:00"
   refdateoracle "march 16, 2006, 00:00"  /2453811/
   datediff      "difference between GAMS and Oracle date"
;
refdategams = jdate(2006,3,16);
datediff = refdategams-refdateoracle;
```

This trick has been applied in a complex scheduling application where dates are important data types that must be exchanged between the application logic and database tier and the optimization engine.

Export to Oracle

Oracle CSV Import

A familiar way of moving data into Oracle is to generate standard SQL `INSERT` statements. The PUT facility is flexible enough to handle this. For instance the following code:

```
file results /results.sql/;
results.lw=0;
results.nw=0;
put results;
loop((i,j),
     put "insert into result (loca, locb, shipment) ";
     put "values ('",i.tl,"','",j.tl,"',",x.l(i,j),");"/
);
putclose;
```

will generate these SQL statements:

```
insert into result (loca, locb, shipment) values ('seattle','new-york',50.00);
insert into result (loca, locb, shipment) values ('seattle','chicago',300.00);
insert into result (loca, locb, shipment) values ('seattle','topeka',0.00);
insert into result (loca, locb, shipment) values ('san-diego','new-york',275.00);
insert into result (loca, locb, shipment) values ('san-diego','chicago',0.00);
insert into result (loca, locb, shipment) values ('san-diego','topeka',275.00);
```

The `.lw` and `.nw` attributes for the put file indicate that no extra spaces around the labels and the numeric values are needed. These field width attributes have a default value of 12 which would cause the values to look like:

```
'seattle     ','new-york    ',       50.00
```

If the amount of data is large the utility SQL∗Loader can be used to import comma delimited input. I.e. the GAMS code:

```
file results /results.txt/;
results.pc=5;
put results;
loop((i,j),
     put i.tl, j.tl, x.l(i,j)/
);
putclose;
```

produces a file results.txt:

```
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00
```

The following SQL∗Loader control file will read this file:

```
LOAD DATA
INFILE results.txt
INTO TABLE result
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
(loca,locb,shipment)
```

GDX to Oracle

Database tables in an SQL RDBMS can be directly generated by the GDXVIEWER tool. The GDXVIEWER can use three methods to export to Oracle and other RDBMS:

1. The direct ADO/ODBC link can create a new table and populate it.
2. The SQL INSERT script generator can create a script with a number of `INSERT` statements.
3. The SQL `UPDATE` script generator can create a script with a number of UPDATE statements.

4.18.3.5 Data Exchange with SQL Server

Import from SQL Server

Microsoft SQL Server is Microsoft's flagship database. It comes in different flavors, including SQL Server, MSDE and SQL Server Express.

Using SQL2GMS

A good way to import SQL server data into GAMS is using the SQL2GMS tool. Below is an example of its use:

```
$set commandfile commands.txt
$onecho > %commandfile%
C=provider=sqloledb;data source=athlon\SQLExpress;Initial catalog=test;user id=sa;password=password
O=C:\WINNT\gamsdir\xx.inc
Q=SELECT * FROM x
$offecho
$call =sql2gms @%commandfile%
parameter p(i,j) /
$include "C:\WINNT\gamsdir\xx.inc"
/;
display p;
```

Using the BCP utilty and CSV files

To export SQL Server data to CSV files we can use the BCP utility.

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>bcp test..results out x.csv \
  -S athlon\sqlexpress -c -U sa -P password -t,

Starting copy...

6 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total     : 10     Average : (600.00 rows per sec.)

C:\Program Files\Microsoft SQL Server\90\Tools\binn>type x.csv
seattle,new-york,50.0
seattle,chicago,300.0
seattle,topeka,0.0
san-diego,new-york,275.0
san-diego,chicago,0.0
san-diego,topeka,275.0
```

It is somewhat more difficult to create a proper CSV file. A format specification file can help here. For an example see the next section on Data Exchange with Sybase . Other tools to export files include DTS (Data Transformation Services) and linked ODBC data sources.

A direct interface between SQL server tables and GAMS GDX files

Finally we can program directly an interface between SQL server tables and GAMS GDX files. A small example in C# can look like:

```
gdxio = new csharpclient();
//
// read a set
//
gdxio.gdxdatawritestrstart(ap, "location", "from db", 1, csharpclient.dt_set, 0);
String q = "select distinct(location) from exporttable";
SqlCommand cmd = new SqlCommand(q, conn);
SqlDataReader myReader = cmd.ExecuteReader();
String[] astrelements = new String[10];
for (int i = 0; i < 10; ++i)
    astrelements[i] = "";
double[] avals = new double[5];
while (myReader.Read())
{
    astrelements[0] = myReader.GetString(0);
    avals[0] = 0.0;
    Boolean ok = gdxio.gdxdatawritestr(ap,astrelements,avals);
}
gdxio.gdxdatawritedone(ap);
myReader.Close();
//
// read a data table
//
gdxio.gdxdatawritestrstart(ap, "data", "from db", 2, csharpclient.dt_par, 0);
q = "select location, capacity, cost exporttable";
cmd = new SqlCommand(q, conn);
myReader = cmd.ExecuteReader();
while (myReader.Read())
{
    astrelements[0] = myReader.GetString(0);
    astrelements[1] = "capacity";
    avals[0] = myReader.GetInt32(1);
    Boolean ok = gdxio.gdxdatawritestr(ap, astrelements, avals);
    if (!myReader.IsDBNull(2)) {
        astrelements[1] = "cost";
        avals[0] = myReader.GetDouble(2);
        ok = gdxio.gdxdatawritestr(ap, astrelements, avals);
    }
}
gdxio.gdxdatawritedone(ap);
myReader.Close();
gdxio.gdxclose(ref ap);
```

Export to SQL Server

SQL Server has two basic facilities to import CSV files: the `BCP` tool and the `BULK INSERT` statement. Advanced SQL Server users may also be able to use DTS (Data Transformation Services) or linked ODBC data sources. Of course for small data sets we can create standard SQL INSERT statements. In addition the tool GDXVIEWER can be used to get GAMS data into SQL Server.

Export using the BCP tool

A transcript showing the use of BCP is shown below:

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
1> use test;
2> create table x(loca varchar(10), locb varchar(10), shipment float);
3> go
Changed database context to 'test'.
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\binn>type c:\winnt\gamsdir\results.csv
seattle,new-york,50.00
seattle,chicago,300.00
```

```
seattle,topeka,0.00
san-diego,new-york,275.00
san-diego,chicago,0.00
san-diego,topeka,275.00

C:\Program Files\Microsoft SQL Server\90\Tools\binn>bcp test..x in c:\winnt\gamsdir\results.csv \
  -S athlon\sqlexpress -c -U sa -P password -t,

Starting copy...

6 rows copied.
Network packet size (bytes): 4096
Clock Time (ms.) Total     : 10     Average : (600.00 rows per sec.)

C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
1> use test
2> select * from x;
3> go
Changed database context to 'test'.
loca       locb       shipment
---------- ---------- ------------------------
seattle    new-york                       50
seattle    chicago                       300
seattle    topeka                          0
san-diego  new-york                       275
san-diego  chicago                          0
san-diego  topeka                         275

(6 rows affected)
1> quit
```

Unfortunately, dealing with quoted strings is not straightforward with this tool (an example using a format file is shown in the next section on Data Exchange with Sybase). The same thing holds for `BULK INSERT`, which can read:

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>type c:\winnt\gamsdir\results.csv
seattle,new-york,50.00
seattle,chicago,300.00
seattle,topeka,0.00
san-diego,new-york,275.00
san-diego,chicago,0.00
san-diego,topeka,275.00

C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
1> use test
2> create table x(loca varchar(10), locb varchar(10), shipment float)
3> go
Changed database context to 'test'.
1> bulk insert x from 'c:\winnt\gamsdir\results.csv' with (fieldterminator=',')
2> go

(6 rows affected)
1> select * from x
2> go
loca       locb       shipment
---------- ---------- ------------------------
seattle    new-york                       50
```

```
seattle    chicago                       300
seattle    topeka                          0
san-diego  new-york                      275
san-diego  chicago                         0
san-diego  topeka                        275

(6 rows affected)
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\binn>
```

Export using the ODBC Text Driver

A slower but flexible way to load CSV files is to use a linked server through the ODBC Text Driver. First create an ODBC DSN using the Text Driver. This can be done through the ODBC Data Source Administrator Data Sources (ODBC) Then we can use the system procedure SP_AddLinkedServer.

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>type c:\winnt\gamsdir\results.csv
"seattle","new-york",50.00
"seattle","chicago",300.00
"seattle","topeka",0.00
"san-diego","new-york",275.00
"san-diego","chicago",0.00
"san-diego","topeka",275.00

C:\Program Files\Microsoft SQL Server\90\Tools\binn>type trnsport.sql
--
--  test database
--
use test

--
-- create table in SQL server
--
create table results(loca varchar(10), locb varchar(10), ship float)
GO

--
-- Create a linked server
--
EXEC sp_addlinkedserver txtsrv,'Jet 4.0','Microsoft.Jet.OLEDB.4.0','c:\winnt\gamsdir',NULL,'Text'
GO

--
-- copy data from text file c:\winnt\gamsdir\results.csv
--
insert into results(loca,locb,ship)
select * from txtsrv...results#csv
go

--
-- check if all arrived
--
select * from results
go
```

```
--
-- release linked server
--
EXEC sp_dropserver txtsrv
GO

--
-- clean up
--
drop table results
go

C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\sqlexpress
1> :r trnsport.sql
Changed database context to 'test'.

(6 rows affected)
loca       locb       ship
---------- ---------- -------------------------
seattle    new-york                       50
seattle    chicago                       300
seattle    topeka                          0
san-diego  new-york                       275
san-diego  chicago                          0
san-diego  topeka                         275

(6 rows affected)
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\binn>
```

A slightly different approach is the following:

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\sqlexpress
1> create table t(loca varchar(10), locb varchar(10), ship float)
2> go
1> insert into t
2> select * from
3> OpenRowSet('Microsoft.Jet.OLEDB.4.0',
4> 'Text;Database=c:\winnt\gamsdir\;HDR=NO',
5> 'select * from results.csv')
6> go

(6 rows affected)
1> select * from t
2> go
loca       locb       ship
---------- ---------- -------------------------
seattle    new-york                       50
seattle    chicago                       300
seattle    topeka                          0
san-diego  new-york                       275
san-diego  chicago                          0
san-diego  topeka                         275

(6 rows affected)
1> drop table t
```

```
2> go
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\binn>
```

This can be automated from GAMS as follows:

```
file results /results.csv/;
results.pc=5;
put results;
loop((i,j),
     put i.tl, j.tl, x.l(i,j)/
);
putclose;

file sqlinsert /insert.sql/;
put sqlinsert;
put "use test"/;
put "insert into t select * from OpenRowSet('Microsoft.Jet.OLEDB.4.0',"
    "'Text;Database=c:\winnt\gamsdir\;HDR=NO','select * from results.csv')"/;
putclose;

execute '="C:\Progra~1\Microsoft SQL Server\90\Tools\binn\sqlcmd" -S athlon\SQLExpress -i insert.sql
```

Export using the GDXVIEWER

When using GDXVIEWER to export data to MS SQL server it is noted that MSSQL Server does not accept the default SQL type double for double precision numbers. You will need to set this setting to float or double precision.

When we export variable `x` from the trnsport model, we see:

```
C:\Program Files\Microsoft SQL Server\90\Tools\binn>sqlcmd -S athlon\SQLExpress
1> use test
2> go
Changed database context to 'test'.
1> select * from x
2> go
dim1                           dim2                           level
------------------------------ ------------------------------ -----------------------
seattle                        new-york                                            50
seattle                        chicago                                            300
seattle                        topeka                                               0
san-diego                      new-york                                           275
san-diego                      chicago                                              0
san-diego                      topeka                                             275

(6 rows affected)
1> quit

C:\Program Files\Microsoft SQL Server\90\Tools\binn>
```

4.18.3.6 Data Exchange with SQLite

See GDX2SQLITE for more information.

4.18.3.7 Data Exchange with Sybase

Import from Sybase

Import using the bcp utility

Sybase is largely the same as SQL Server. For exporting ASCII files from a Sybase table, the utility (Using the BCP utilty and CSV files) can be used.

An example of use of this utility is shown below:

```
[erwin@fedora sybase]$ isql -U sa -S LOCALHOST -D testdb -P sybase -J iso_1
1> select * from results
2> go
 loca       locb       shipment
 ---------- ---------- --------------------
 seattle    new-york             50.000000
 seattle    chicago             300.000000
 seattle    topeka                0.000000
 san-diego  new-york             275.000000
 san-diego  chicago                0.000000
 san-diego  topeka               275.000000

(6 rows affected)
1> quit
[erwin@fedora sybase]$ cat bcp.fmt
10.0
4
1       SYBCHAR 0       0       "\""    1       loca
2       SYBCHAR 0       10      "\",\"" 1       loca
3       SYBCHAR 0       10      "\","   2       locb
4       SYBCHAR 0       17      "\n"    3       shipment
[erwin@fedora sybase]$ bcp testdb..results out res.txt -S LOCALHOST  -U sa -P sybase \
  -J iso_1 -f bcp.fmt

Starting copy...

6 rows copied.
Clock Time (ms.): total = 1  Avg = 0 (6000.00 rows per sec.)
[erwin@fedora sybase]$ cat res.txt
"seattle","new-york",50.0
"seattle","chicago",300.0
"seattle","topeka",0.0
"san-diego","new-york",275.0
"san-diego","chicago",0.0
"san-diego","topeka",275.0
[erwin@fedora sybase]$
```

Note: the first column in the format file is a dummy (it has length 0). This is in order to write the leading quote, as bcp only allows for termination symbols.

This can be automated using the following GAMS code:

```
sets
  i   'canning plants'   / seattle, san-diego /
  j   'markets'          / new-york, chicago, topeka / ;


$onecho > bcp.fmt
10.0
4
1       SYBCHAR 0       0       "\""    1       loca
2       SYBCHAR 0       10      "\",\"" 1       loca
3       SYBCHAR 0       10      "\","   2       locb
4       SYBCHAR 0       17      "\n"    3       shipment
$offecho
$call "bcp testdb..results out res.txt -S LOCALHOST  -U sa -P sybase -J iso_1 -f bcp.fmt"
parameter d(i,j)  'distance in thousands of miles'
/
$ondelim
$include res.txt
$offdelim
/;
display d;
```

Import using the SQL2GMS utility

The SQL2GMS tool uses ADO or ActiveX Data Objects to extract data from relational databases. It can connect to almost any database from any vendor as it supports standards like ODBC. See documentation for more information.

Import using a 'SQL2GMS' VBS script

The following GAMS code will generate and execute a script written in VBScript. It mimics the behavior of SQL2GMS.EXE and can be used for debugging or the script can be passed on to the IT support people in case there are problems with accessing the database.

```
$ontext
   This script mimics SQL2GMS.
   Erwin Kalvelagen
   November 2006
$offtext

$onecho > sql2gms.vbs
'
' parameters
'
t1 = 3                    ' connection timeout
t2 = 0                    ' command timeout
c = "Provider=MSDASQL;Driver={SQL Server};Server=DUOLAP\SQLEXPRESS;Database=testdata;
Uid=gams;Pwd=gams;"  ' connection string
q = "select * from data"  ' query
o = "output.inc"          ' the output file to be generated
b = false                 ' whether to quote indices (e.g. because of embedded blanks)

'
' create ADO connection object
```

```
'
set ADOConnection = CreateObject("ADODB.Connection")
ADOVersion = ADOConnection.Version
WScript.Echo "ADO Version:",ADOVersion

'
' make db connection
'
ADOConnection.ConnectionTimeout = t1
ADOConnection.ConnectionString = c
ADOConnection.Open

'
' Open file
'
set fso = CreateObject("Scripting.FileSystemObject")
set outputfile = fso.CreateTextFile(o,True)
outputfile.writeLine "*-------------------------------------------------"
outputfile.writeLine "*  SQL2GMS/Vbscript 1.0"
outputfile.writeLine "*  Connection:"&c
outputfile.writeLine "*  Query:"&q
outputfile.writeLine "*-------------------------------------------------"

'
' setup query
'
starttime = time
ADOConnection.CommandTimeout = t2
const adCmdText = 1
set RecordSet = ADOConnection.Execute(q,,adCmdText)

'
' get results
'
NumberOfFields = RecordSet.Fields.Count
eof = RecordSet.EOF
if eof then
   WScript.Echo "No records"
   Wscript.quit
end if

'
'  loop through records
'
NumberOfRows = 0
do until eof
   NumberOfRows = NumberOfRows + 1
   Row = RecordSet.GetRows(1)
   if NumberOfFields > 1 then
      s = Row(0,0)
      if b then
        s = quotestring(s)
      end if
      Outputfile.Write s
   end if

   for i=2 to NumberOfFields-1
       s = Row(i-1,0)
```

```
       if b then
         s = quotestring(s)
       end if
       Outputfile.Write "."
       Outputfile.Write s
   next

   s = Row(NumberOfFields-1,0)
   OutputFile.Write " "
   OutputFile.Writeline s

   eof = RecordSet.EOF
loop

OutputFile.Close

Wscript.echo "Records read:"&NumberOfRows
Wscript.echo "Elapsed time:"&DateDiff("s",starttime,time)&" seconds."


function quotestring(s)
has_single_quotes = false
has_double_quotes = false
needs_quoting = false

'
' check input string for special characters
'
for j=1 to len(s)
   ch = Mid(s,j,1)
   select case ch
     case "'"
         has_single_quotes = true
     case """"
         has_double_quotes = true
     case " ","/",";",","
         needs_quoting = true
     case else
         k = asc(ch)
         if (k<=31) or (k>=127) then
            needs_quoting = true
         end if
   end select
next

'
' check if we have if gams keyword
'
kw = array("ABORT","ACRONYM","ACRONYMS","ALIAS","BINARY","DISPLAY","ELSE", _
           "EQUATION","EQUATIONS","EXECUTE","FILE","FILES","FOR","FREE",  _
           "IF","INTEGER","LOOP","MODEL","MODELS","NEGATIVE","OPTION",    _
           "OPTIONS","PARAMETER","PARAMETERS","POSITIVE","PROCEDURE",     _
           "PROCEDURES","PUT","PUTCLEAR","PUTCLOSE","PUTHD","PUTPAGE",    _
           "PUTTL","SCALAR","SCALARS","SEMICONT","SET","SETS","SOS1",     _
           "SOS2","TABLE","VARIABLE","VARIABLES","WHILE")

if not needs_quoting then
   for j = 0 to Ubound(kw)
```

```
      if strcomp(s,kw(j),1)=0 then
        needs_quoting = true
        exit for
      end if
   next
end if

'
' already quoted?
'
ch = left(s,1)
select case ch
   case "'", """"
       quotestring = s
       exit function
end select

' check for special case
if has_single_quotes and has_double_quotes then
   quotestring = """" & replace(s, """", "'") & """"
elseif has_single_quotes then
   quotestring = """" & s & """"
elseif has_double_quotes then
   quotestring = "'" & s & "'"
elseif needs_quoting then
   quotestring = "'" & s & "'"
else
   quotestring = s
end if


end function
$offecho

execute '=cscript sql2gms.vbs';
```

4.19 Executing GAMS from other Environments

4.19.1 Some General Comments

Nowadays the objected-oriented APIs are the most efficient and elegant way to interact with the GAMS system. They allow the effective communication of data, and do parameterized runs of GAMS. This whole chapter describes a much more basic interaction of GAMS through calling the GAMS executable from different environments. This still can be useful, e.g. if no object-oriented API is available for the particular target language (e.g. VBA).

While the principle of calling the GAMS executable holds for all operating systems, this chapter often focuses on the Windows platform.

One of the interesting problems one faces when spawning GAMS.EXE in a Windows environment is multi-threading. If one does not take precautions, a call to Shell (VB function) or CreateProcess (Win32 API function) causes GAMS to run asynchronously: the function will return while GAMS is still running. In order to read the results one would need to wait until the GAMS job has finished. The machinery for this requires some Windows trickery, and for Visual Basic version 6 and Delphi version 4 we have implemented some small examples that illustrate how this can be done.

Another issue that needs to be addressed is that GAMS needs a place to put its scratch files. By default this is the current directory, a concept that is not always clear in a windowing environment. A good way of dealing with his is to set both the current drive and the current directory before running the GAMS job. It should be noted that GAMS needs write permission there. In the examples we use the Windows TEMP directory for this, but in a real application you may want to use a designated directory. The Windows TEMP directory is found by calling the API function GetTempPath.

4.19.1.1 The GAMS architecture

GAMS itself is a console mode application. In fact it is not a single program, but a driver program (GAMS.EXE on Windows, otherwise GAMS) that executes in turn the GAMS language compiler (GAMSCMEX) or one of the solvers. For a model with a single solve statement, GAMS.EXE will first call GAMSCMEX to compile the GAMS model. Then GAMSCMEX will start executing the internal code that was generated by the compiler. As soon as it hits the instructions belonging to a SOLVE statement it will generate model instance, and GAMSCMEX will exit. Then GAMS.EXE will spawn a solver capable of solving the model. As soon as the solver is finished, GAMS.EXE will execute GAMSCMEX again so it can read the solution and can continue with executing instructions.

4.19.2 Spawning GAMS from VBA

Visual Basic for Applications (VBA) is a programming language that is built into most Microsoft Office applications, e.g. Excel and Access. A VBA program may include modules, which can be imported from files, i.e. in the VBA editor you can choose menu "File"→"Import File". The GAMS distribution includes some VBA modules, which can be found from:

```
<GAMS System Directory>\apifiles\VBA\api, e.g. "C:\GAMS\win64\24.5\apifiles\VBA\api"
```

Attention

- In order to avoid issues, it's recommended to use the latest version of the modules , i.e. the modules found in the latest GAMS release.

For example, the following modules can be found:

- gamsglobals.bas: Global constants that are used in other modules
- gdxvba.bas: GAMS Data Exchange Object
- idxvba.bas: GAMS IDX Object
- optvba.bas: GAMS Option Object

  For more information, see Expert-Level APIs. VBA programs that use the API can be found from:

  ```
  <GAMS System Directory>\apifiles\VBA, e.g. "C:\GAMS\win64\24.5\apifiles\VBA"
  ```

  The models can also be retrieved via GAMS Studio -> GAMS --> Model Library Explorer -> GAMS Data Utilities Models.

4.19.2.1 Spawning GAMS from Excel

Calling GAMS out of Excel requires some more work than just Data Exchange with Microsoft Excel. The application calling GAMS out of Excel has to:

1. Locate the GAMS system directory and adjust the system path accordingly.
2. Copy the GAMS model into a temporary directory (by default the temporary directory of Windows)
3. Extract the model data from the spreadsheet into a GAMS readable format (gdx)
4. Execute GAMS (solve the model, write the slution back to gdx file)
5. Import the model results back from the gdx files into the spread sheet
6. Update the spreadsheet (graphics, tables)

Using VisualBasic for Applications (VBA) this can be implemented with a few lines of code:

```
Sub solve()
   Dim WorkDir As String
   WorkDir = TempDir()
   Call ClearSolution
   If (Not AddGAMSPath()) Then ' needed to find gdxcclib64.dll and gams.exe
      Exit Sub
   End If
   Call ExportGDXFile(WorkDir)
   Call WriteGAMSModel(WorkDir & "portfolio.gms")
   Call RunGAMS(WorkDir & "portfolio.gms", WorkDir)
   Call ReadListing(WorkDir & "portfolio.lst")
   Call ImportGDXFile(WorkDir)
End Sub
```

For further details, please inspecte the VBA part of the examples below.

4.19.2.2 A Simple Example

This very simple example shows how GAMS can be invoked from an Excel spreadsheet. The "example spreadsheet" has a button, which will cause GAMS to run the trnsport.gms model stored in c:/tmp. There is no data exchange.

A more complete application will write an include file for a GAMS model, and will import a comma delimited file with results when the run is finished. An example of such a complete application is described in http://www.gams.com/mccarl/excelgams.pdf.

4.19.2.3 Sudoku Example

This spreadsheet is a complete example that uses GDX files to exchange information solves a 25x25 Sudoku problem using CPLEX. It comes with the GAMS distribution in *apifiles/VBA/sudoku.xlsm*. You will need a GAMS/CPLEX license to be able to run the spreadsheet. The MIP model solves very easily: the solution is found in the presolve phase.

Note: This spreadsheet requires distribution 22.6 or younger to work properly.

Problem

Solution

4.19.2.4 Efficient frontier example

This example solves a series of NLPs to create an efficient frontier of a portfolio optimization problem. It comes with the GAMS distribution in *apifiles/VBA/portfolio.xlsm*.

Note: This spreadsheet requires distribution 22.6 or younger to work properly.

4.19.3 Spawning GAMS from C

The example below shows an absolute minimal version of calling GAMS from a C program:

```
#include <stdio.h>

int main(int argc, char** argv)
{
   system("gams trnsport lo=2");
   return 0;
}
```

As can be seen the GAMS executable gams.exe is called via a simple invocation of system(). The extra command line parameter `lo=2` indicates that the log file is not written to the screen (this is the default) but to a file `model.log`.

If we want to generate some data from the application program (e.g. parameter `DEMAND`), we can use an include file which is written by the application. Of the way back results can be read from a PUT file. The GAMS file can now look like:

```
Sets
       i   canning plants   / seattle, san-diego /
       j   markets          / new-york, chicago, topeka / ;

  Parameters

       a(i)  capacity of plant i in cases
         /    seattle     350
              san-diego   600  /

       b(j)  demand at market j in cases
         /
$include demand.inc
         / ;

  Table d(i,j)  distance in thousands of miles
                    new-york       chicago      topeka
      seattle          2.5           1.7          1.8
      san-diego        2.5           1.8          1.4  ;

  Scalar f  freight in dollars per case per thousand miles  /90/ ;

  Parameter c(i,j)  transport cost in thousands of dollars per case ;

            c(i,j) = f * d(i,j) / 1000 ;

  Variables
       x(i,j)  shipment quantities in cases
       z       total transportation costs in thousands of dollars ;

  Positive Variable x ;

  Equations
       cost        define objective function
       supply(i)   observe supply limit at plant i
       demand(j)   satisfy demand at market j ;

  cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

  supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
```

```
  demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

  Model transport /all/ ;

  Solve transport using lp minimizing z ;

  Display x.l, x.m ;

file fout /results.txt/;
put fout;
loop((i,j),
   put x.l(i,j):17:5/;
);
putclose;
```

The corresponding C program could look like:

```
//
// example of running a GAMS model
//
#include <stdio.h>
// number of supply and demand locations
#define M 2
#define N 3
char*  dist[M] = {"seattle", "san-diego" };
char*  city[N] = {"new-york","chicago", "topeka" };
double demand[N] = { 325.0, 300.0, 275.0 };
double ship[M][N];
int main(int argc, char** argv)
{
   FILE *f;
   int i,j;
   //
   // write include file
   //
   f = fopen("demand.inc","wt");
   if (f == NULL) {
      perror("fopen");
      exit(1);
   }
   for (i=0; i<N; ++i)
       fprintf(f,"%s %g\n",city[i],demand[i]);
   fclose(f);
   //
   // call GAMS
   //
   system("gams trnsport lo=2");
   //
   // read solution
   //
   f = fopen("results.txt","rt");
   if (f == NULL) {
      perror("fopen");
      exit(1);
   }
   for (i=0; i<M; ++i)
      for (j=0; j<N; ++j)
          fscanf(f,"%lg",&(ship[i][j]));
   fclose(f);
   //
   // display results
   //
   for (i=0; i<M; ++i)
      for (j=0; j<N; ++j)
          printf("%s->%s %g\n",dist[i],city[j],ship[i][j]);
   return 0;
}
```

4.19.4 Spawning GAMS from Visual Basic

This example shows how GAMS can be invoked from a VB program.

When executed a simple window appears where a GAMS model can be specified and possible other command line options. The [GAMS] button will execute the model.

When the Normal display is used, a console window will be opened. This console window will be closed automatically at the end of the run if the Close process window on completion check box is checked.

Minimized does not show a window, but in the taskbar an icon will appear. If the job takes a long time, the user can click on this icon to make the console window visible. The Hidden option prevents any GAMS associated window or icon to appear.

Note: it is required for this program to work, that the GAMS system directory is in the path. If this is not the case, an error code of -1 is returned.

The main program is attached below.

```
Attribute VB_Name = "vbGAMS"
Option Explicit
' ------------------------------
' Module Name: VBGams
' File:  gams.bas
' Version: 1.1
'
'
' Functions:
'   VB_Gams32   - sets up process directory, parameter file, and console, and manages
'                 looping of gams compiler and solver.
'
' --------
' Function VB_Gams32
'   Sets up process directory, parameter file, and console, and manages
'   looping of gams compiler and solver.
'
' Parameters
'   sParams       String      Command line arguments
'   nDisplayMode  Integer     GAMS_NORMAL, GAMS_MIN, or GAMS_HIDDEN
'   bCloseWin     Boolean     Close the console on completion of gams job
'
' Returns
'   1000 missing input string
'   2000 16 bit spawn failed
'   -or-
'   nCmexRC + 100 * nVB_GamsRC
'
'   nCmexRC : return code from last run of GAMS compile/execute module
'             (for DOS/Win XX)
'         0 : normal return
'         1 : solve is next (should not happen)
'         2 : compilation error
'         3 : execution error
'         4 : system limits
'         5 : file error
'         6 : parameter error
'         7 : licensing error
'         8 : GAMS system error
'         9 : GAMS could not be started
'
'    nVB_GamsRC:  vbGams specific error codes
'         0 : normal return
'         1 : could not create process dir
'         2 : could not run gamsparm script
'         3 : could not append user input to parameter scratch file
'         4 : could not spawn gamscmex.exe
'         5 : could not shell off "gamsnext" script
'         6 : could not delete process directory
'
Public Function VB_Gams32(sParams As String, nDisplayMode As Integer, _
                          bCloseWin As Boolean) As Long
Dim lAlloc As Long, lRetAPI As Long
Dim nSpawn As Long
    If nDisplayMode = GAMS_NORMAL Then   ' allocate a console
        lAlloc = AllocConsole()
        Dim sConTitle As String, hWnd As Long, nTop As Long, nLeft As Long
        sConTitle = "VB GAMS "
        lAlloc = SetConsoleTitle(sConTitle)  ' so I can find it
        ' if the console window is to stay open, then, need to reposition console window.
        If Not bCloseWin Then
            Call Sleep(100)   ' necessary so console can come up
            hWnd = FindWindow(vbNullString, sConTitle)
            If hWnd <> 0 Then
                nTop = 20
                nLeft = 20
                lAlloc = SetWindowPos(hWnd, HWND_BOTTOM, nTop, nLeft, 0, 0, SWP_NOZORDER + SWP_NOSIZE)
                Call CloseHandle(hWnd)
            End If
        End If
    End If
    nSpawn = Spawn("gams.exe " & sParams, nDisplayMode)

    If nDisplayMode = GAMS_NORMAL Then
        If Not bCloseWin Then
            Beep
```

```
            MsgBox "Click here to close console.", vbOKOnly + vbInformation, "VB GAMS"
        End If
        lAlloc = FreeConsole()
    End If
    VB_Gams32 = nSpawn
    ShowWait False
End Function
```

4.19.5 Spawning GAMS from Delphi

This is a Delphi 4 application that has similar features as the Visual Basic application described in the previous paragraph.

Note: it is required for this program to work, that the GAMS system directory is in the path. If this is not the case, an error code of -1 is returned. The main code is attached below.

```
unit main;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
type
  TMainForm = class(TForm)
    FileEdit: TEdit;
    FileOpenDialog: TOpenDialog;
    filelabel: TLabel;
    BrowseButton: TButton;
    CmdLineOptionsEdit: TEdit;
    CmdLineLabel: TLabel;
    RunButton: TButton;
    ReturnLabel: TLabel;
    ConsoleComboBox: TComboBox;
    ConsoleLabel: TLabel;
    CloseConsoleButton: TButton;
    procedure BrowseButtonClick(Sender: TObject);
    procedure RunButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure CloseConsoleButtonClick(Sender: TObject);
  private
    { Private declarations }
    function GSExec(const ProgName,ProgParams: string;
                const wShowWindow : word;
                var rc: integer): integer;
    function ExecuteGAMS(
         const ProgName,ProgParams: string;
         wShowWindow : word;  { SW_NORMAL, SW_HIDE or SW_SHOWMINIMIZED }
         AutoClose : boolean;  { close console at end }
         var rc: integer): integer;
  public
    { Public declarations }
  end;
var
  MainForm: TMainForm;
implementation
{$R *.DFM}
function TMainForm.GSExec(const ProgName,ProgParams: string;
                        const wShowWindow : word;
                        var rc: integer): integer;
{execute program:
  Result: error code for starting the program
  rc    : error code returned by the program}
var
   Command            : String;
   ProcessInformation : TProcessInformation;
   StartupInfo        : TStartupInfo;
   exitcode           : dword;
begin
// Initialise the startup information to be the same as that of the
// calling application.  This is easier than initialising the many
// individual startup information fields and should be fine in most
// cases.
GetStartupInfo(StartupInfo);
// StartupInfo.wShowWindow determines whether the called application
// will be initially displayed normal, maximises, minimised or some
// other subtle variations
StartupInfo.wShowWindow := wShowWindow;
StartupInfo.dwFlags := StartUpInfo.dwFlags or STARTF_USESHOWWINDOW;
Command := ProgName + ' ' + ProgParams;
if not CreateProcess(
    Nil,                   {ApplicationName}
    PChar(Command),        {lpCommandLine }
    Nil,                   {lpProcessAttributes}
```

```
    Nil,                    {lpThreadAttribute}
    false,                  {bInheritedHandles}
    NORMAL_PRIORITY_CLASS,  {dwCreationFlags}
    Nil,                    {lpEnvironment}
    Nil,                    {lpCurrentDirectory}
    StartupInfo,            {lpStartupInfo}
    ProcessInformation      {lpProcessInformation}
    )
then
   begin
   rc := 0;
   Result := GetLastError  {failed to execute}
   end
else
   begin
   with ProcessInformation
   do begin
      WaitForSingleObject(hProcess,INFINITE);
      GetExitCodeProcess(hProcess,exitcode);
      CloseHandle(hThread);
      CloseHandle(hProcess);
      end;
   Rc := exitcode;
   Result := 0;
   end;
end;
function TMainform.ExecuteGAMS(
         const ProgName,ProgParams: string;
         wShowWindow : word;  { SW_NORMAL, SW_HIDE or SW_SHOWMINIMIZED }
         AutoClose : boolean;  { close console at end }
         var rc: integer): integer;
var ok : BOOL;
begin
   if AutoClose then begin
      { this is the easy one }
      result := GSExec(ProgName, ProgParams, wShowWindow, rc);
      exit;
   end;
   { in the case we want to let the user close the window,
     we need to allocate the console ourselves }
    ok := AllocConsole();
    result := GSExec(ProgName, ProgParams, wShowWindow, rc);
    { if our console was used, show button to get rid of it }
    if (ok) then
       CloseConsoleButton.Enabled := true;
end;
procedure CdTemp;
{ cd to windows temp directory }
const maxpath=260;
var path : string;
begin
  setlength(path,maxpath);
  GetTempPath(maxpath,Pchar(path));
  path := ExpandFileName(path); { just to be sure }
  ChDir(path);  { will also change drive }
end;
procedure TMainForm.BrowseButtonClick(Sender: TObject);
begin
    { popup file open dialog }
    if FileOpenDialog.Execute then
       FileEdit.text := FileOpenDialog.Filename;
end;
procedure TMainForm.RunButtonClick(Sender: TObject);
var
  rc : integer; { return code from GAMS.EXE }
  result : integer; { return code from GsExe }
  params : string; { command line params for GAMS.EXE }
  s : string; { for assembly of error messages }
  wShowWindow: word;
  AutoClose : boolean;
begin
  { set current directory }
  CdTemp;
  { extract command line parameters from edit controls }
  { there may be blanks in the filenames, so add quotes }
  params := '"' + FileEdit.Text + '" ' + CmdLineOptionsEdit.Text;
  { get wShowWindow information }
  case ConsoleComboBox.ItemIndex of
    0, 1 : wShowWindow := SW_NORMAL;
    2 : wShowWindow := SW_SHOWMINIMIZED;
    3 : wShowWindow := SW_HIDE;
  end;
  AutoClose := not (ConsoleComboBox.ItemIndex = 1);
  { inform user GAMS is running }
  ReturnLabel.Caption := 'Running GAMS...';
  ReturnLabel.Font.Color := ClGreen;
  Refresh;
```

```
  { run GAMS.EXE }
  result := ExecuteGAMS('gams.exe', params, wShowWindow, autoClose, rc);
  { check for results }
  if (result <> 0) then begin
     str(result,s);
     ReturnLabel.Caption := 'Exec failed: result = '+s;
     ReturnLabel.Font.Color := ClRed;
  end else if (rc <> 0) then begin
     str(rc,s);
     ReturnLabel.Caption := 'GAMS failed: rc = '+s;
     ReturnLabel.Font.Color := ClRed;
  end else begin
     ReturnLabel.Caption := 'OK';
     ReturnLabel.Font.Color := ClBlack;
  end;
end;
procedure TMainForm.FormCreate(Sender: TObject);
begin
   ConsoleComboBox.ItemIndex := 0;
end;
procedure TMainForm.CloseConsoleButtonClick(Sender: TObject);
begin
   FreeConsole();
   CloseConsoleButton.Enabled := false;
end;
end.
```

4.19.6 Spawning GAMS from Visual C++

In "this example" we call GAMS from a simple Visual C++ application.

The additional concern is here that we want to intercept the GAMS screen output so it can be written to a multi-line edit control. For more information about the used technique see the Microsoft publication: [HOWTO: Spawn Console Processes with Redirected Standard Handles (Q190351)] (https://support.microsoft.com/en-us/kb/190351) in the Microsoft Knowledge Base.

4.19.7 Spawning GAMS from C#

Below is a simple example:

```
using System.Diagnostics;
void RunGamsModel()
{
    Process p = new Process();
    p.StartInfo.FileName = gamsexe;
    p.StartInfo.WorkingDirectory = tempdir;
    p.StartInfo.Arguments = "\"" + modelname + "\" LO=0 --runid="+runid;
    p.Start();
    p.WaitForExit();
}
```

The command line formed here is `gams modelname LO=0 --runid=xxx`. The option `LO` (logoption) will disable writing a log, and the option `--runid` passes a parameter to the GAMS model (inside the model you can access this through `%runid%`).

4.19.8 Spawning GAMS from Java

Java has a class *Runtime* that implements spawning of processes using the Exec() method. This is quite trivial to use in applications, but in applets the default security settings don't allow this operation (in general). The problem can be solved by loading the applet from a local drive or by using a signed applet. Similarly if the Java classes are loaded from inside a database, this operation may require additional privileges. For an example see the previous section, where GAMS is called from Oracle using a Java Stored Procedure. The relevant code may look like:

```
import java.io.File;
class RunGAMS {
        public static void main(String[] args) {
                System.out.println("Start");
```

```
                String[] cmdArray = new String[5];
                cmdArray[0] = "<PATH/TO/GAMS>" + File.separator + "gams";
                cmdArray[1] = "<PATH>" + File.separator + "trnsport.gms";
                cmdArray[2] = "WDIR=<PATH>" + File.separator + "TMP";
                cmdArray[3] = "SCRDIR=<PATH>" + File.separator + "TMP";
                cmdArray[4] = "LO=2";
                try {
                        Process p = Runtime.getRuntime().exec(cmdArray);
                        p.waitFor();
                }
                catch (java.io.IOException e )
                {
                        System.err.println(">>>>" + e.getMessage() );
                        e.printStackTrace();
                }
                catch (InterruptedException e )
                {
                        System.err.println(">>>>" + e.getMessage() );
                        e.printStackTrace();
                }
                System.out.println("Done");
        }
}
```

Below another example, which avoids problems, if the model has a long screen log (the buffer gets filled and locks the execution), which could happen if GAMS does not write the log to the file like above. This example is based on suggestion made by Edson Valle.

```
import java.io.File;
import java.io.BufferedReader;
import java.io.InputStreamReader;
class RunGAMS {
      public static void main(String[] args) {
              System.out.println("Start");
              String[] cmdArray = new String[5];
              cmdArray[0] = "<PATH/TO/GAMS>" + File.separator + "gams.exe";
              cmdArray[1] = "<PATH>" + File.separator + "trnsport.gms";
              cmdArray[2] = "<PATH>" + File.separator + "tmp";
              cmdArray[3] = "LO=3";
              try {
                      Process p = Runtime.getRuntime().exec(cmdArray);
                      BufferedReader stdInput = new BufferedReader(new InputStreamReader(p.getInputStream()));
                      String s = null;
                      while((s=stdInput.readLine()) !=null){
                         System.out.println(s);
                      }
                      p.waitFor();
              }
              catch (java.io.IOException e )
              {
                      System.err.println(">>>>" + e.getMessage() );
                      e.printStackTrace();
              }
              catch (InterruptedException e )
              {
                      System.err.println(">>>>" + e.getMessage() );
                      e.printStackTrace();
              }
              System.out.println("Done");
      }
}
```

For an example of GAMS usage from a Java based server environment see Alexander Sokolov, Information environment and architecture of decision support system for nutrient reduction in the Baltic Sea, Department of Systems Ecology, Stockholm University.

4.19.9 Spawning GAMS from a Web Server

Running GAMS remotely using a Web based thin-client architecture requires that GAMS is executed directly or indirectly from the Web server or HTTP server. A simple way of doing this is via a CGI process. Common Gateway Interface (CGI) programs can be written in many languages such as C, Perl or Delphi. CGI is relatively slow, as for each interaction, even the most simple one, a process needs to be started. Alternatives exist in the form of CGI extensions such as FastCGI or using DLLs or shared libraries. A basic algorithm for a CGI script could be:

- Create a unique directory, and CD to that directory.

- Get information from user forms and save it as GAMS readable files.
- Run GAMS making sure it does not write the log to the screen (i.e. use the option LO=2 (log to a file) or LO=3 (log to stdout)).
- Let the model write solution info to text files using the PUT statement.
- Read solution, and create formatted HTML to send back to the user.
- Remove temp files and directory.

Complications arise when there is a need to show graphics (files need to be stored somewhere and discarded after a while), when jobs take a long time to finish (you will need to add a facility where the user can pick up results at a later moment) or when the server resources can be exhausted (e.g. because of a large number of simultaneous users or because of large models).

An important complicating issue in the above list are jobs that take more time to finish: web servers like to respond to the user within a short time, and time out errors will occur if a job takes a long time. The solution is to use a queue based approach. An actual implementation available is the NEOS Server - there is also a GAMS interface to NEOS: KESTREL - Remote Solver Execution on NEOS Servers.

4.19.10 Spawning GAMS from PHP

A minimal example

index.html

```
<html>
  <body>
    <form action="calling_gams.php" method="post">
      Solve transport with f = <select name="f">
          <option>70</option>
          <option>80</option>
          <option>90</option>
          <option>100</option>
        </select>
      <input type="submit" value="call GAMS"/>
    </form>
  </body>
</html>
```

calling_gams.php

```
<?php
$f = $_POST['f'];
//some model data
$modelfile = 'trnsport_php.gms';
$city   = array('new-york','chicago', 'topeka');
$demand = array(325.0, 300.0, 275.0);
//write demand.inc
$fh = fopen('./demand.inc', 'w+');
for($i=0; $i<count($city); $i++){
  fwrite($fh, $city[$i]." ".$demand[$i]."\n");
}
fclose($fh);
//write f.inc
$fh = fopen('./f.inc', 'w+');
fwrite($fh, $f);
fclose($fh);
//call gams
system('</path/to/gams>/gams '.$modelfile.' lo=2');
//read solutions
$fh = fopen('./results.txt', 'r');
echo '<p><b>result of '.$modelfile.' (f='.$f.') :</b></p>';
while (!feof($fh)){
   $line = fgets($fh);
   echo '<p>'.$line.'</p>';
}
fclose($fh);
?>
```

trnsport_php.gms

```
Sets
       i   canning plants   / seattle, san-diego /
       j   markets          / new-york, chicago, topeka / ;

  Parameters

       a(i)  capacity of plant i in cases
         /    seattle     350
              san-diego   600  /

       b(j)  demand at market j in cases
         /
$include demand.inc
         / ;


  Table d(i,j)  distance in thousands of miles
                    new-york       chicago      topeka
      seattle          2.5           1.7          1.8
      san-diego        2.5           1.8          1.4  ;

  Scalar f  freight in dollars per case per thousand miles  /
$include f.inc
         / ;

  Parameter c(i,j)  transport cost in thousands of dollars per case ;

            c(i,j) = f * d(i,j) / 1000 ;

  Variables
       x(i,j)  shipment quantities in cases
       z       total transportation costs in thousands of dollars ;

  Positive Variable x ;

  Equations
       cost        define objective function
       supply(i)   observe supply limit at plant i
       demand(j)   satisfy demand at market j ;

  cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;

  supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;

  demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

  Model transport /all/ ;

  Solve transport using lp minimizing z ;

  Display x.l, x.m ;

  file fout /results.txt/;
  put fout;
  loop((i,j),
    put i.tl:0 '->' j.tl:0 ':  ' x.l(i,j)/;
  );
  put 'cost: 'z.l/;
```

```
putclose;
```

4.20 Using GAMS Studio

This tutorial on using GAMS Studio was written initially by and is distributed by courtesy of Bruce A. McCarl [2] . Bruce has extensive experience with GAMS, both as a modeler and an educator, and many GAMS users know, use, and benefit from his work. This tutorial presents a guided tour to the usage of GAMS Studio and on some items it reflects Bruce's personal view and preferences.

4.20.1 What Is It?

A multi platform (not only PCs with Windows) graphical interface to run GAMS

GAMS uses two phases.

1. Text edit

   First, one uses a **text editor** and creates a file which contains GAMS instructions.

2. File submission

   Second, one **submits that file** to GAMS which executes those instructions causing calculations to be done, solvers to be used and a solution file of the execution results to be created.

Approaches to use GAMS.

1. More batch oriented traditional method use a text editor set up the model then use **DOS (or UNIX)** command line instructions to find errors in and run the model.

2. GAMS STUDIO or GAMS IDE alternatives. Graphical interfaces to create, debug, text edit and run GAMS files.

[2] Specialist in Applied Optimization, Distinguished Professor of Agricultural Economics, Texas A&M University, Principal, McCarl and Associates, mccarl[at]tamu.edu, brucemccarl[at]gmail.com, http://agecon2.tamu.edu/people/faculty/mccarl-bruce/

Summary of steps to using

1. Install GAMS and STUDIO on your computer

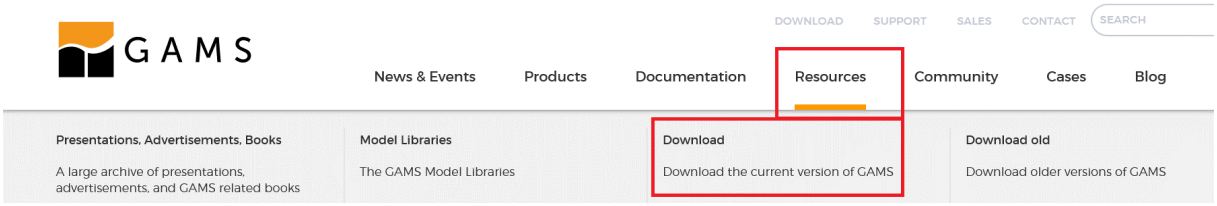
2. Open STUDIO through the icon
3. Open or create a file by going to the file selection in the upper left corner.
4. Prepare the file so you think it is ready for execution
5. Run the file with GAMS by clicking the run button or pressing F9
6. Open and navigate around the output

4.20.2 Installation

Install GAMS and STUDIO

GAMS STUDIO is **automatically installed** when GAMS is installed and you get the **choice of whether to use STUDIO or the IDE as primary editor**. To install do the following steps

1. Download the GAMS installation file from `www.gams.com` under `Resources and Download`

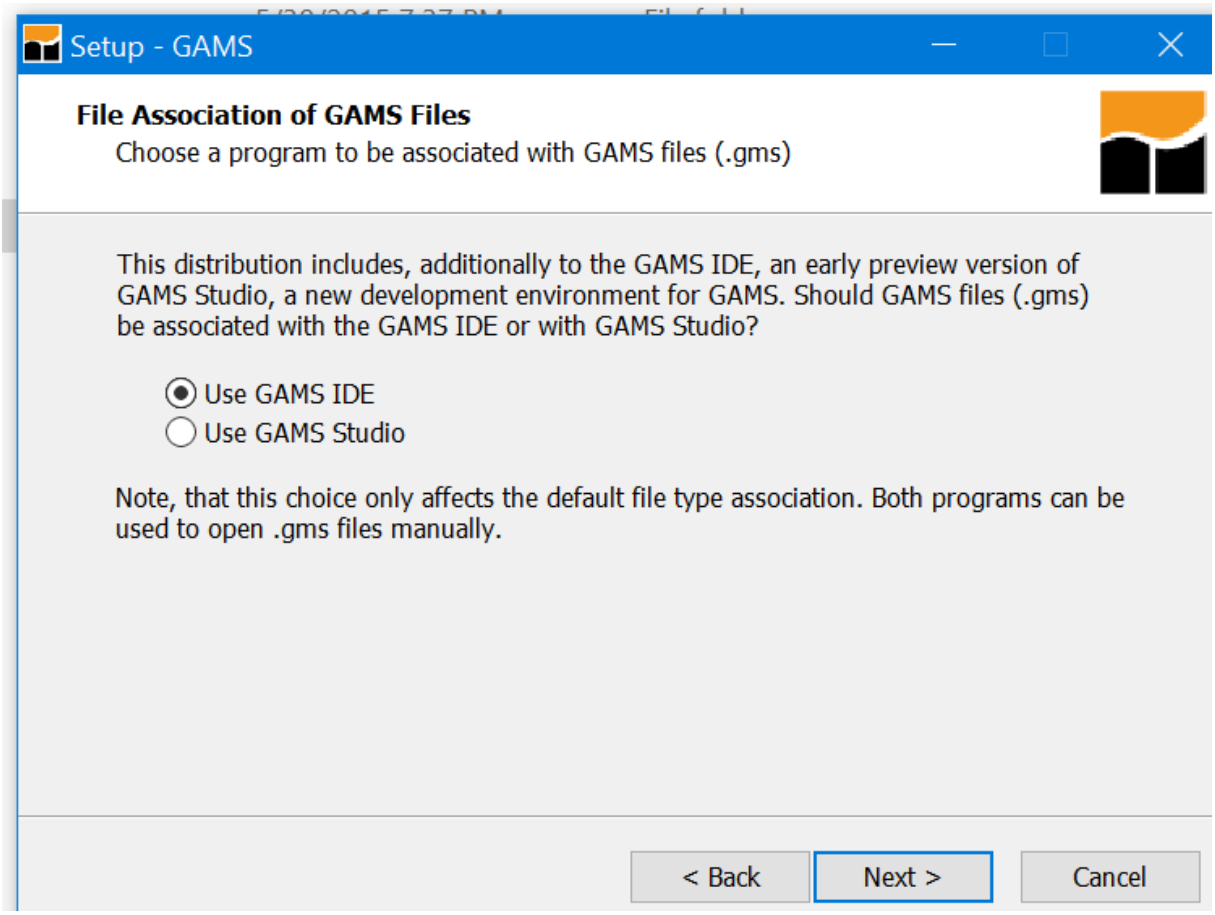


2. Start the installation

Installer should place an icon on the desktop yielding

4.20.3 Using STUDIO after Installation

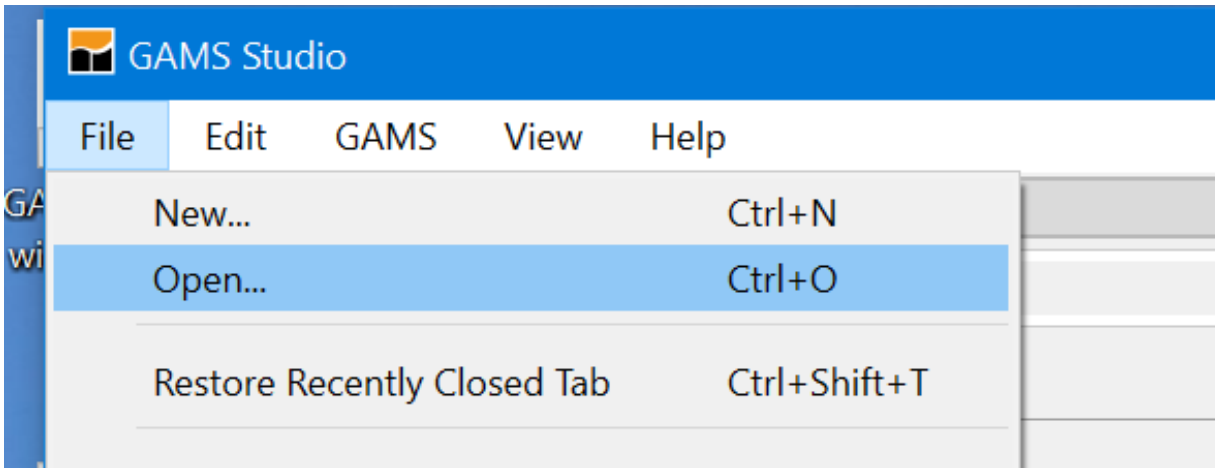
During the install you will find



Which allows you to choose which graphical interface to use as the default. For now choose STUDIO.

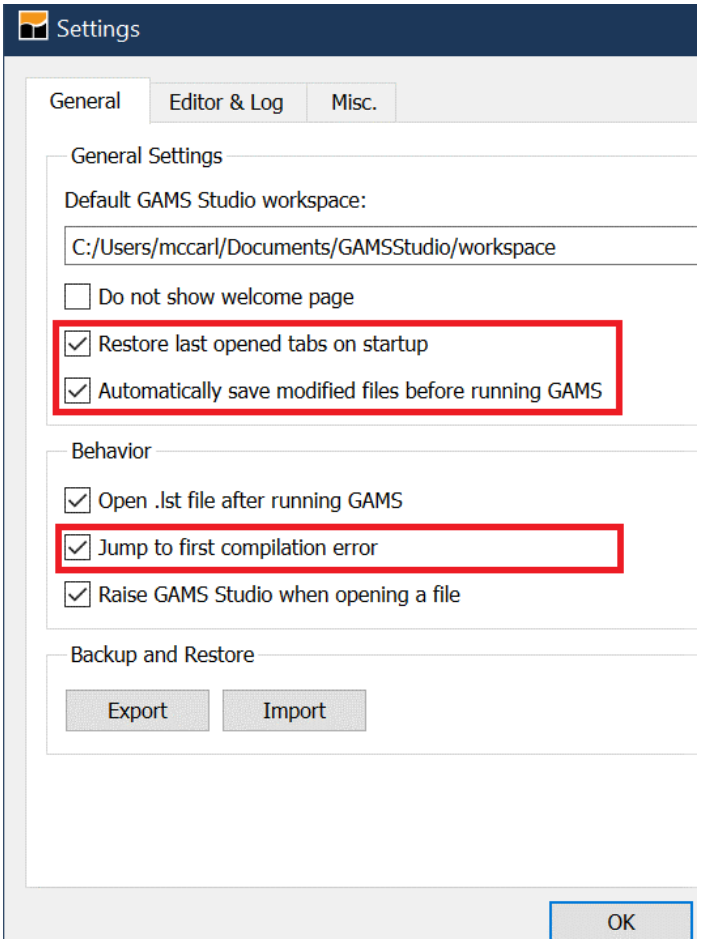
Open STUDIO through the icon

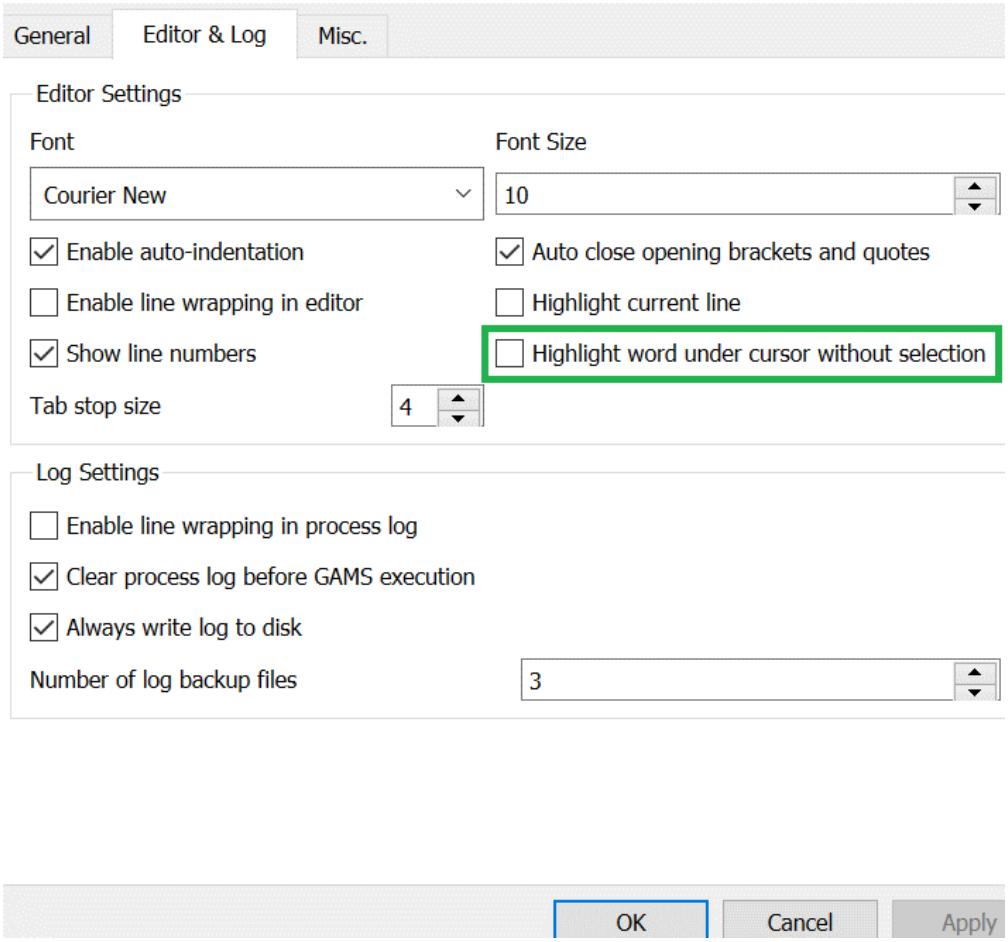
Open the File menu choice. Select a file.



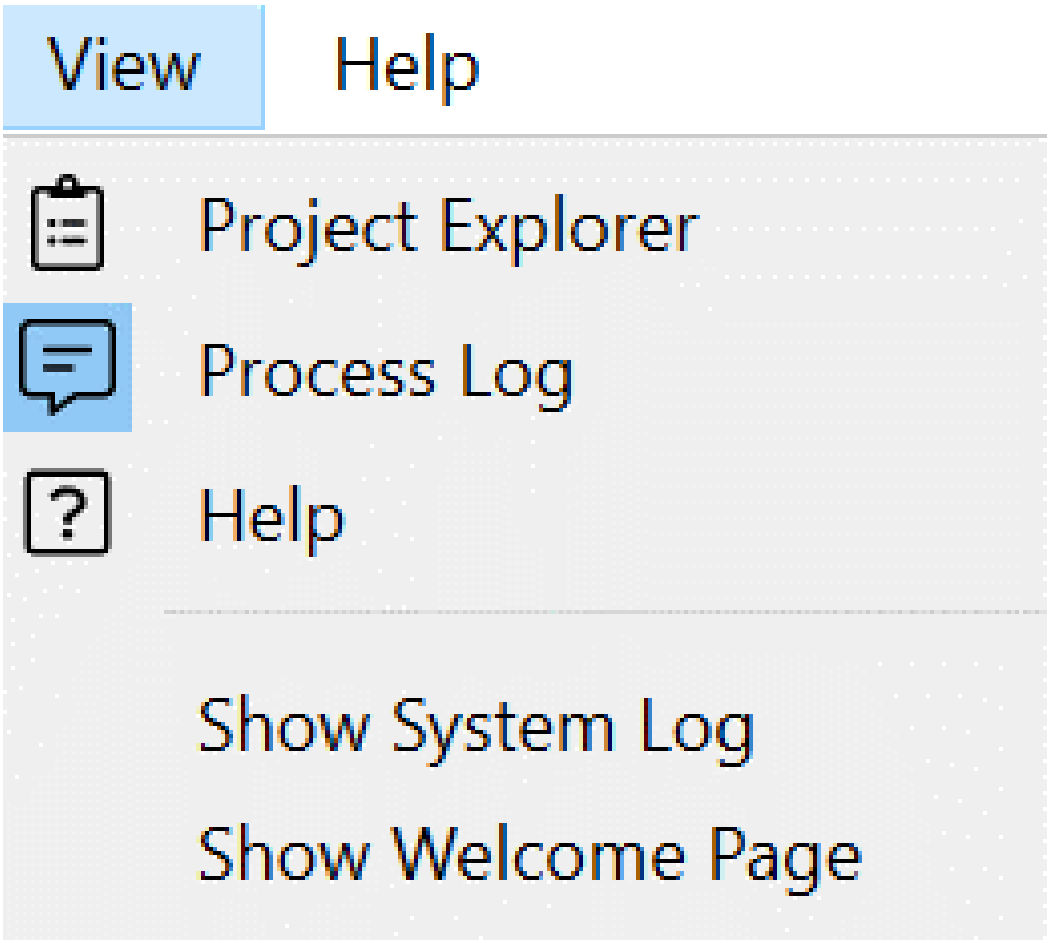
Note, the file location determines where all generated files are placed (to place files elsewhere use the save as dialogue) and where GAMS looks for files when executing. *I recommend that you make sure your file is in a place where you want it also making sure that place contains all associated files ie this is the file storage directory for this endeavor.*

Fix defaults and suppress cursor driven object highlighting. When first opening Studio go to settings and I would make sure check marks are as follows especially the checked ones in the red boxes and the unchecked green one





Here selecting the **Process Log** and possibly the **Explorer** are of interest.



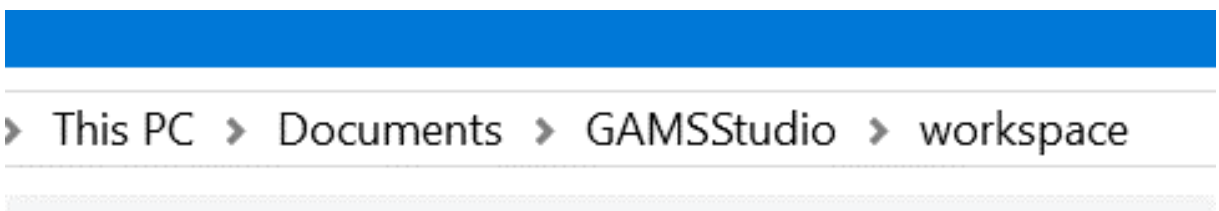
The Explorer will open automatically when a file is opened and the **Process Log** will open when a solve is done. I usually hide the Explorer and sometimes want to get rid of the Process window.

4.20.3.1 Getting Started

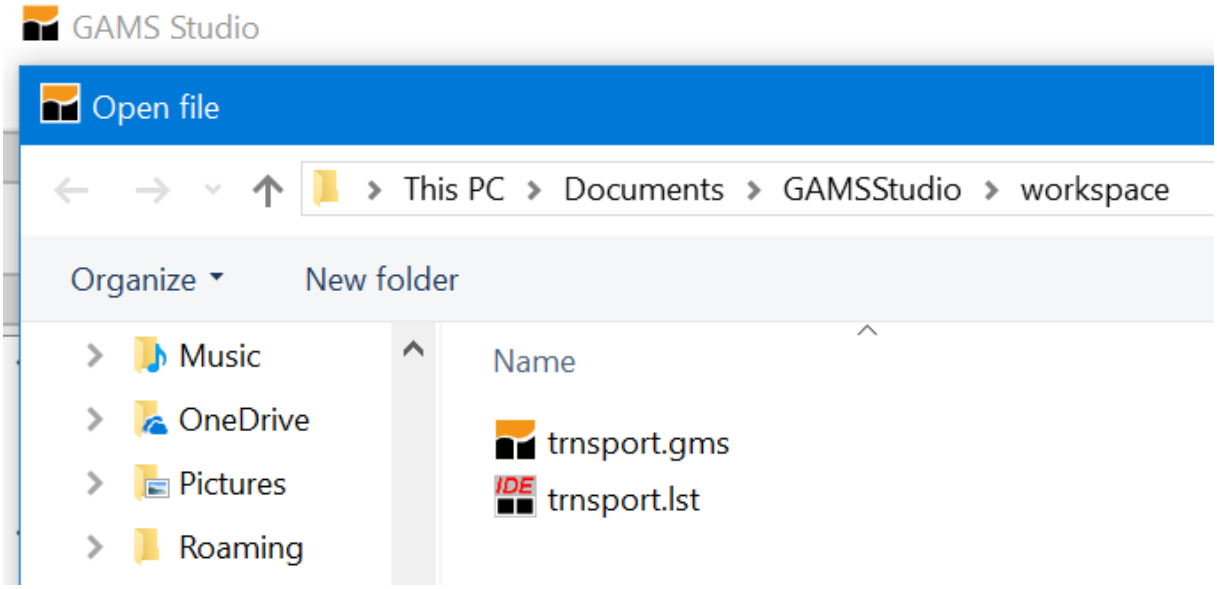
Create or open an existing GMS file

Several cases are possible

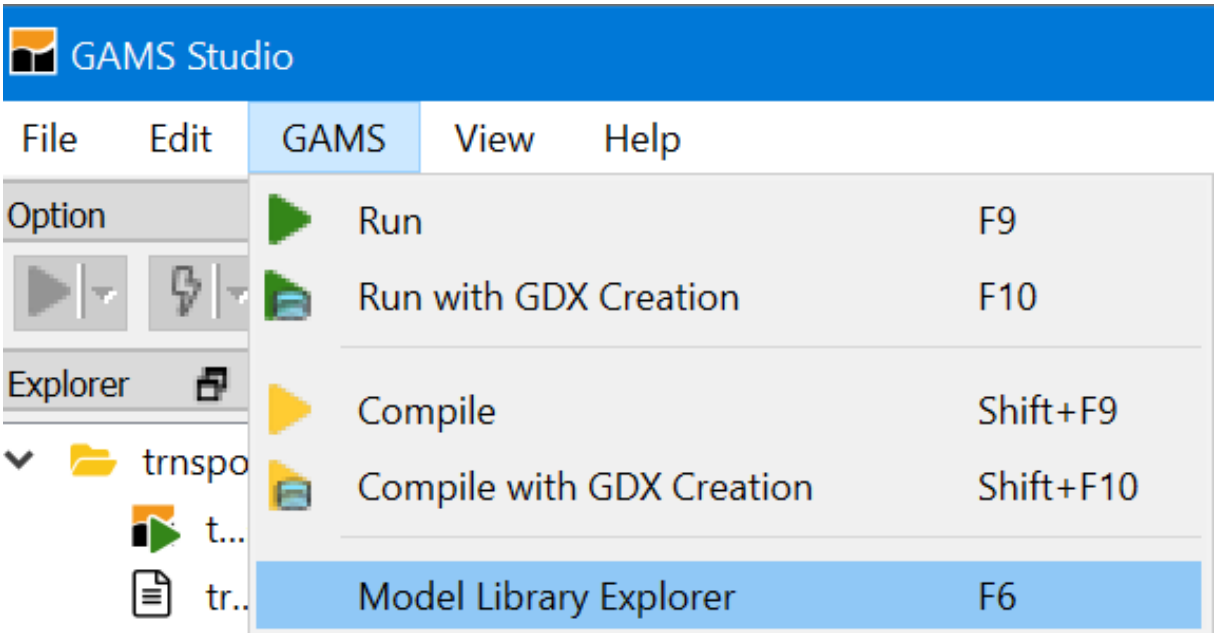
1. Create a new file (covered later)
2. Open an existing file – at first this will be in



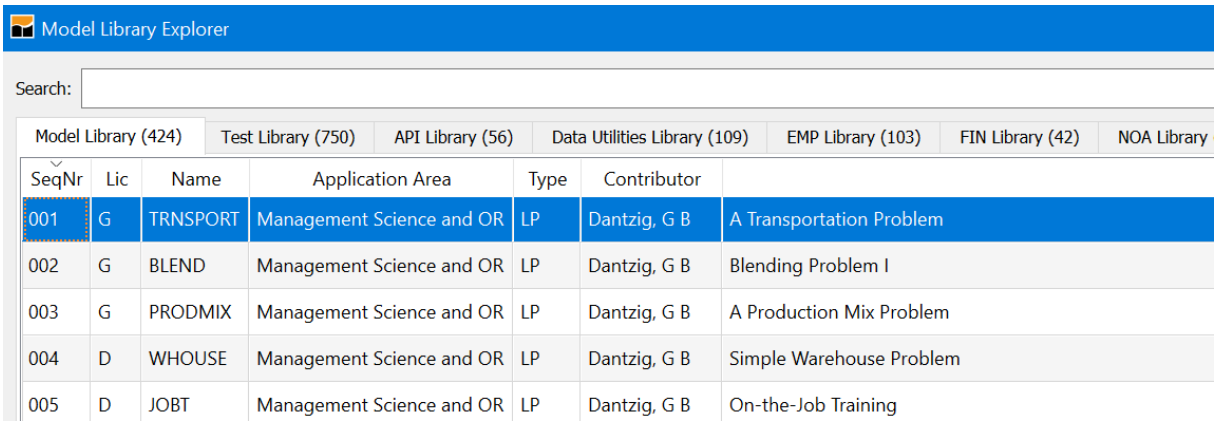
then you see something like



3. Open the GAMS model library explorer under the GAMS tab or press F6 (this is the simplest for first time users and the one we will use)



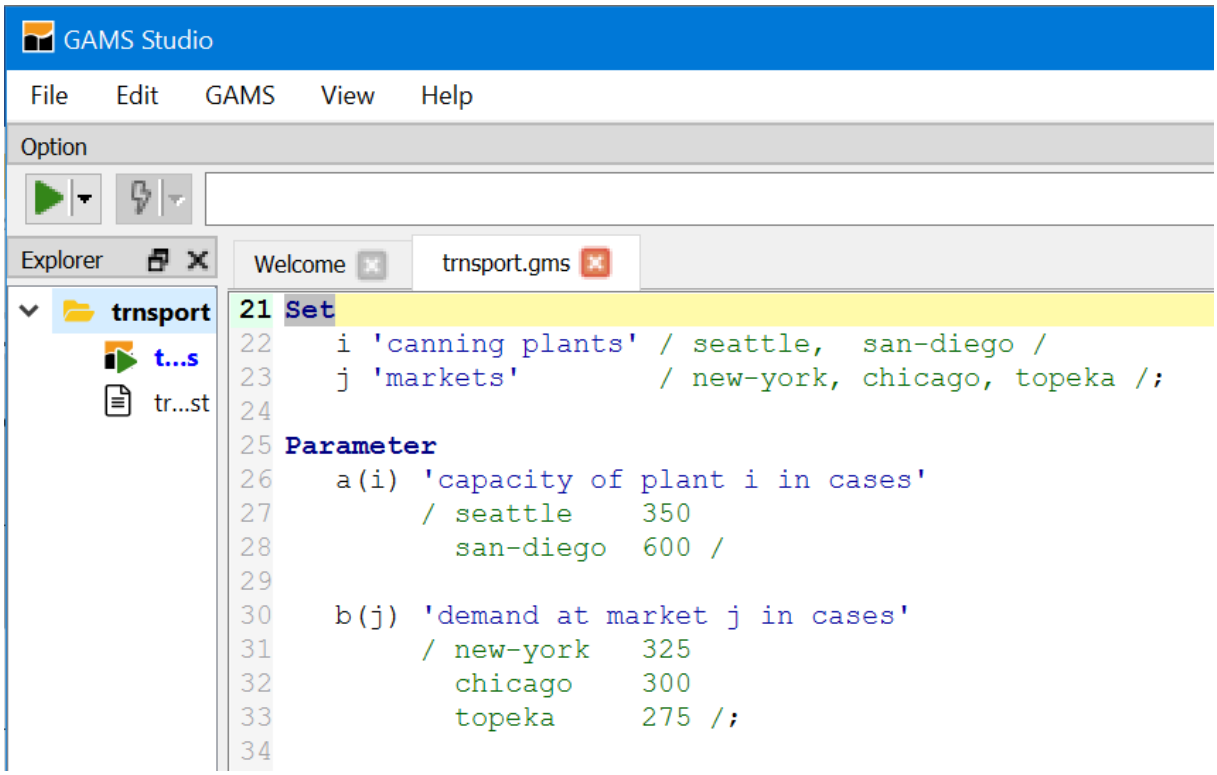
Select a model like *trnsport.gms*



It will be automatically saved in the last directory you used or in the STUDIO default Workspace directory.

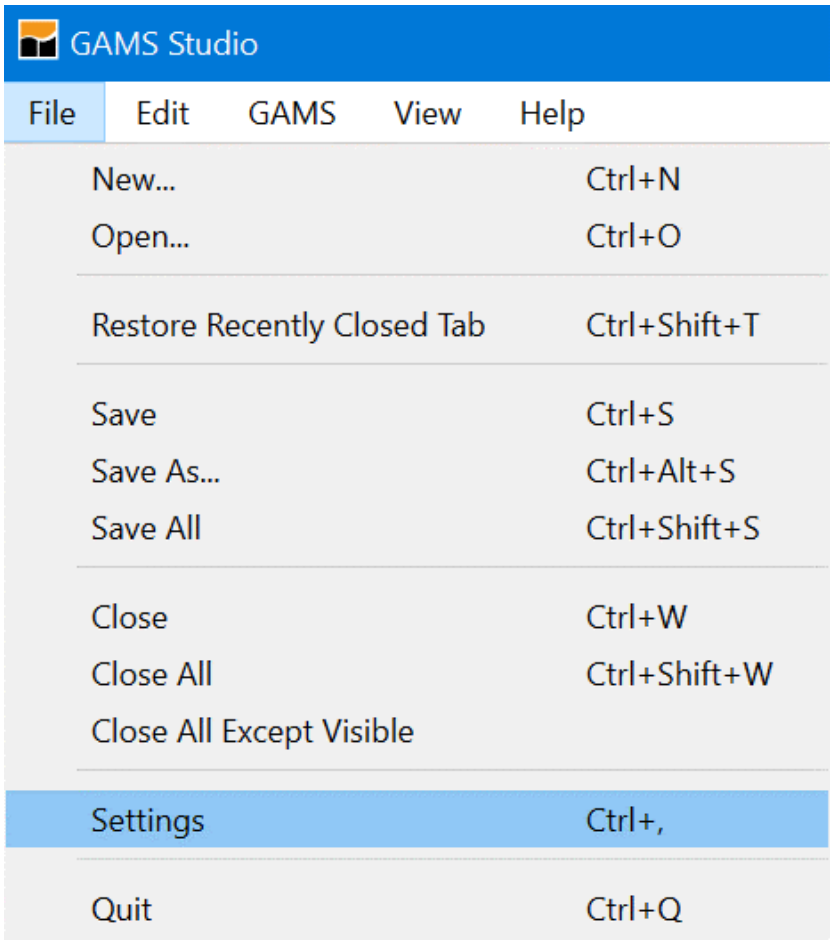
Prepare file for execution

When using model library *trnsport.gms* should now appear as part of your STUDIO screen.



The STUDIO contains a full-featured editor. Go through the file and change what you want.

You can also customize the appearance somewhat through the settings choice under the File menu.

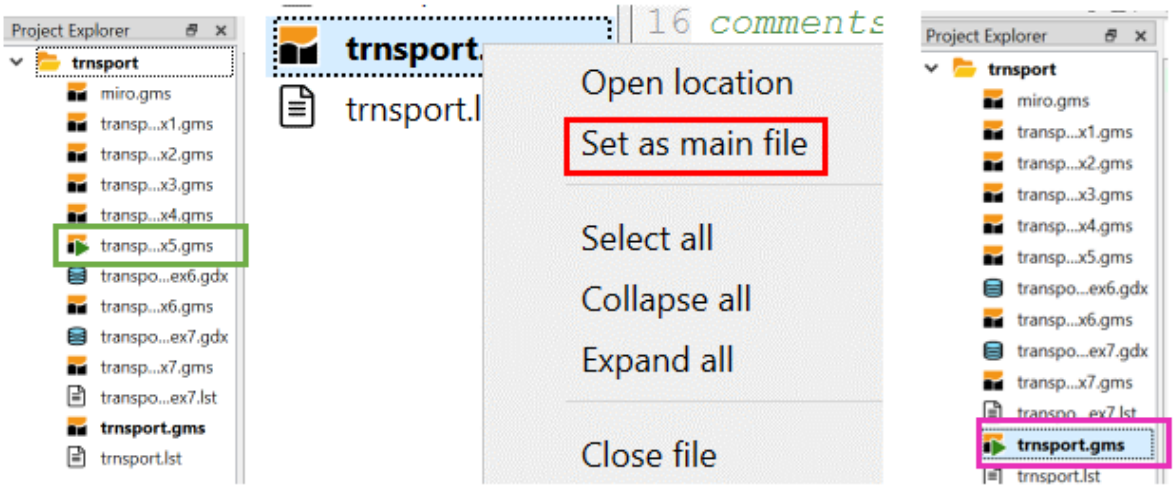


There you can change font size (under editor), tabs, default file location (called workspace) and some other features on how GAMS runs. For now leave it alone.

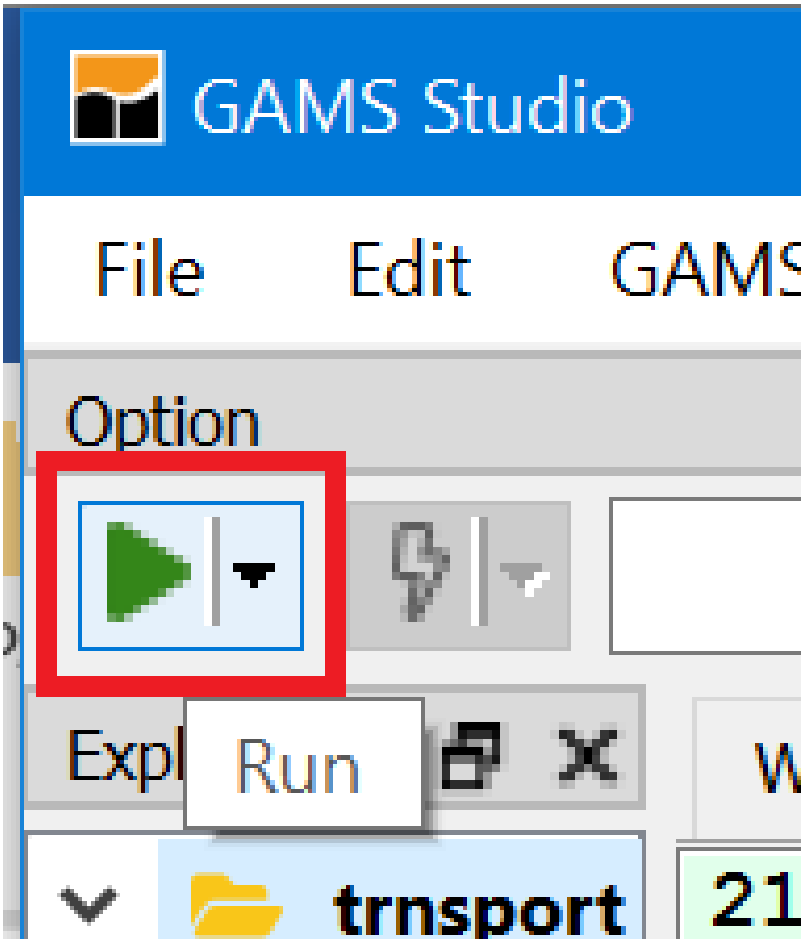
Make sure right file is executed

Unlike a lot of other programs, STUDIO does not necessarily execute the file you have been working on rather executing what it calls the "main" file.

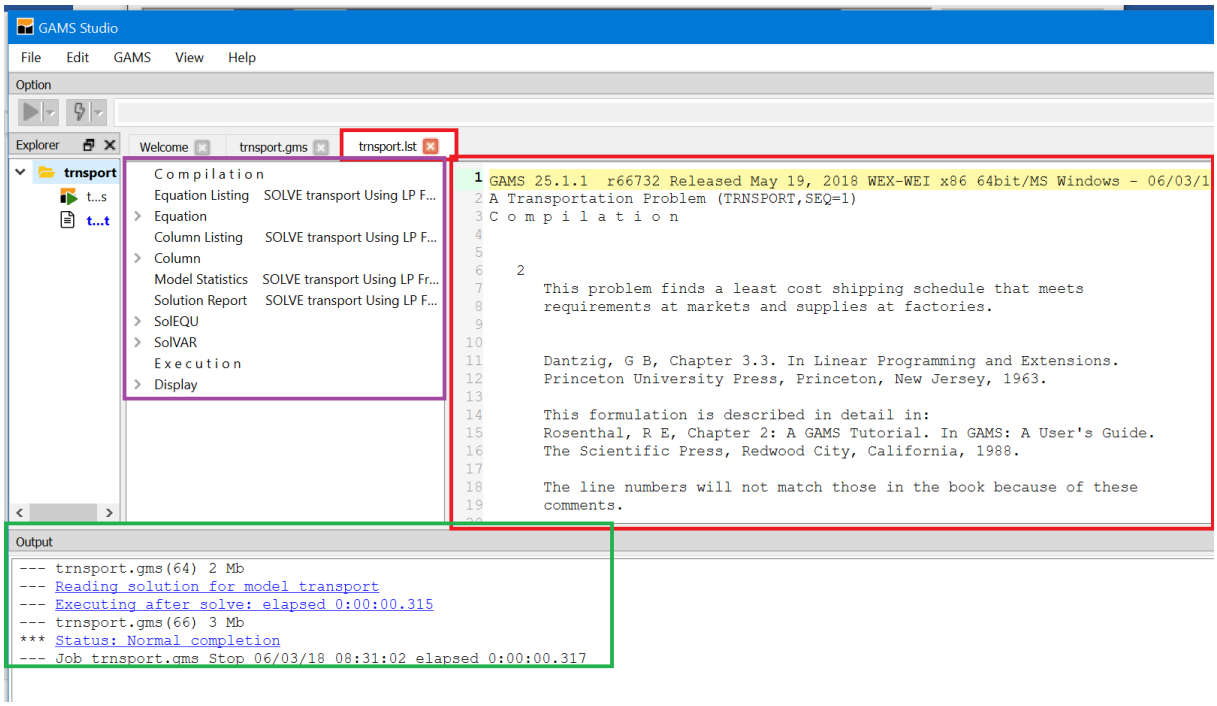
In particular, if you activate the explorer view then the main file is the one marked with a **green triangle** as below. If the triangle is not marking the file to be run you need to right click on that file and then choose **Set as main file**. In turn the triangle will move.



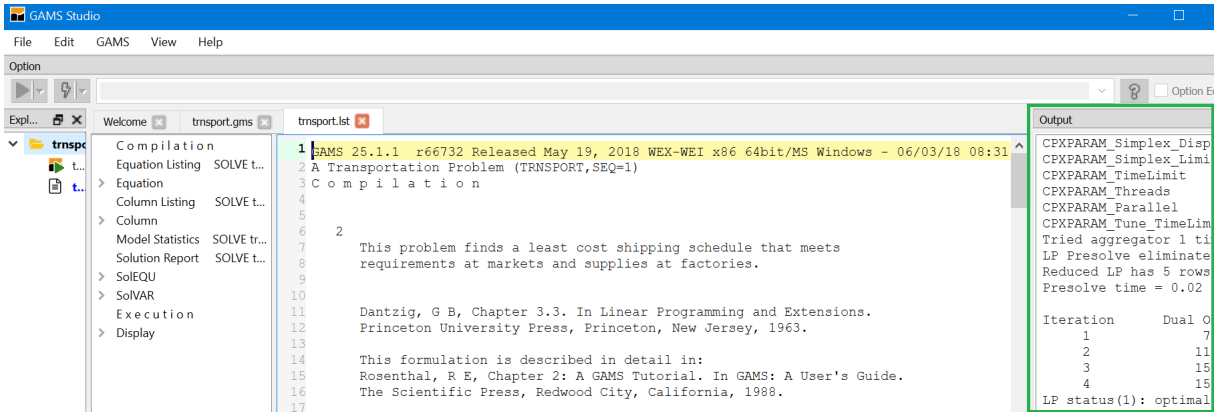
Run GAMS by clicking the run button



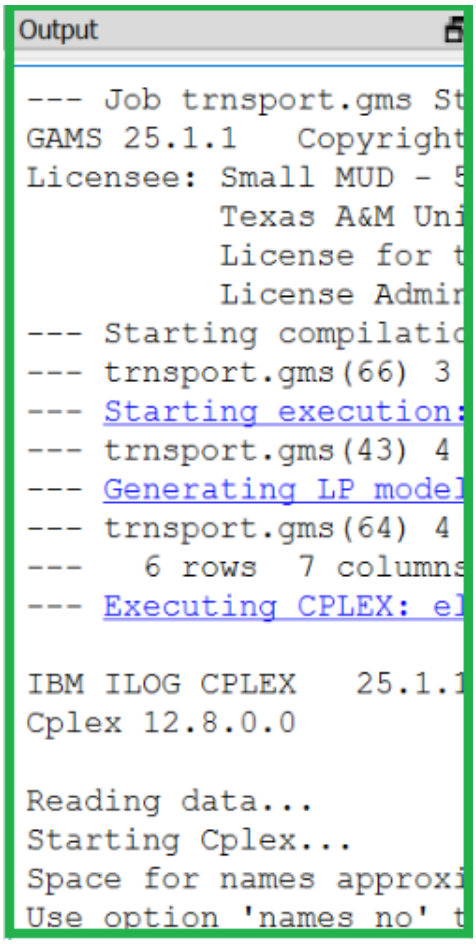
The **Process Log** window (green) will then appear which logs the steps executed in running the model. The **LST file** (red) and the **navigation window** (magenta) also appear. Note to save space I would close the Explorer window and drag the **Process Log** window (green) to the bottom or the right hand side. The process window on the bottom is illustrated here and it is on the right side on the following picture.



Here it is after reconfiguring by dragging the *Process Log* up and to the right then resizing and repositioning.



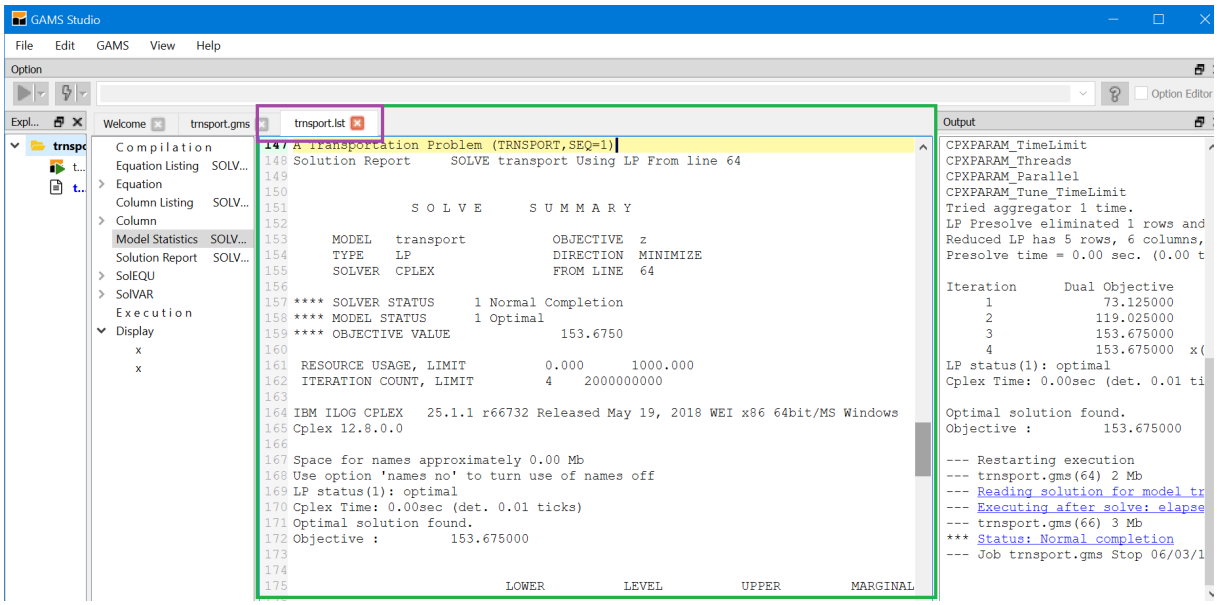
Now use the *process window* (green) to navigate the LST file or open created files. By double clicking on lines in that window you access the LST file at particular locations or open created files. What and where you access is determined by the color of the line you click on.



Color of Line	Function and Destination When Double Clicked
Blue line	Opens LST file and jumps to corresponding place.
Non-bolded black line	Control click on this takes you to LST file. Double click on error message takes you to spot in LST file.
Red line	Marks errors encountered. Cursor is positioned at first red line. When clicked on cursor jumps to Source (GMS) file location of error. Control click takes you to error location in LST file. Error description text is in execution process log and in LST file. You can use a little red icon in source to jump to error message in LST file or a green arrow in the LST to go to error location in source.
Green line	Causes studio to open a file that was created during the run like a GDX, PUT or REF file

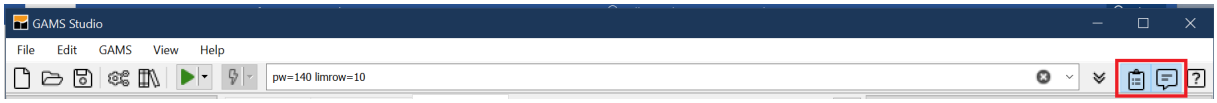
Navigate with process window

After double-clicking on any non-red line, or if we used the automatically open list file option our main editing window is augmented by the **LST file** (green) (see the magenta tabs)



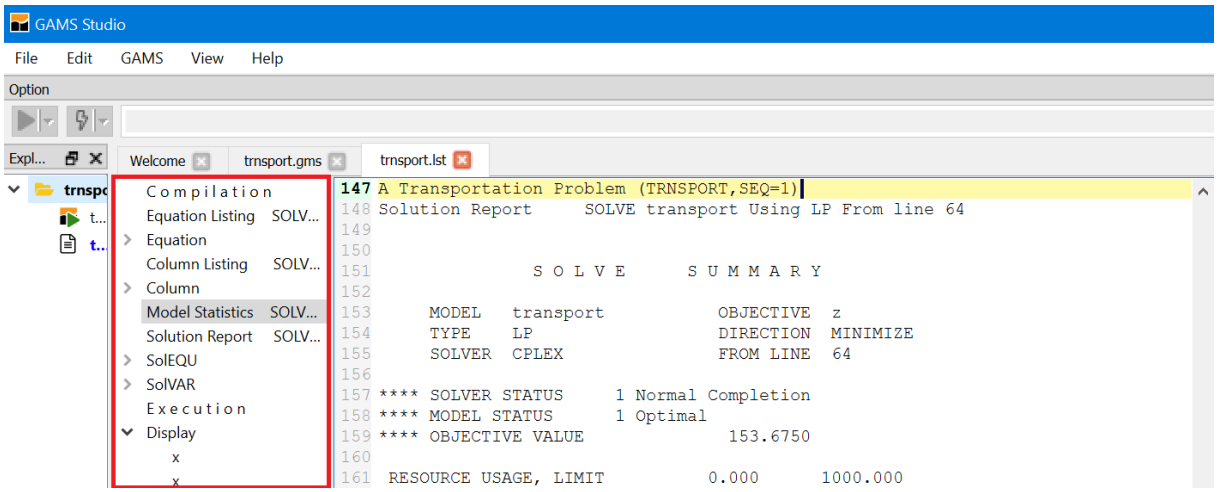
We can navigate as we would with an editor or word processor, as we are automatically in STUDIO text editor.

Sometimes I want to get the process window and explorer out of the way. I can do this through the view window and also through the **icons** (red) on the right side of the top bar



Clicking on these opens and closes the explorer and the process window.

Navigate with LXI window



Navig. Line	Function and Destination When Clicked.
Compilation	Jumps to top of echo print in LST file
Error Mess.	Jumps to list of error messages when errors present

Navig. Line	Function and Destination When Clicked.
Equat. listing	Jumps to list of equation contents
Equation	Expandable allowing jump list of equation contents
Variab. listing	Jumps to list of variable
Variable	Expandable allowing jump to list of variable contents
Model stat.	Jumps to model statistics part of LST file
Solut.Report	Jumps to model summary solution report
SolEQU	Expandable allowing jump solution for each equation
SolVAR	Expandable allowing jump to solution for variable
Execution	Jumps to beginning of post solve execution
Display	Jump to displays of specific parameters and other items

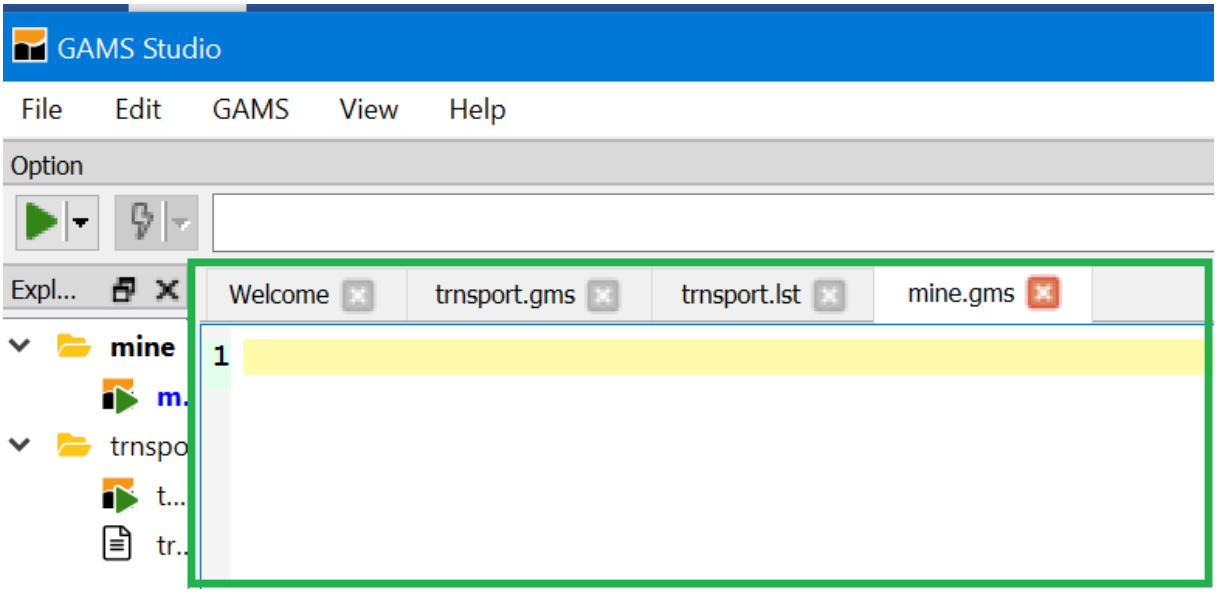
4.20.3.2 Working with your own files

Now you are ready to work with your own files. You may already have a file or you may need to create one.

Creating a new file - Two principal ways

1. Open an existing file through the library or from a location on your hard disk. Then use the **save as** dialogue from the **file** menu to change its name. Now modify contents to what you want. You may cut and paste as in other Windows programs.

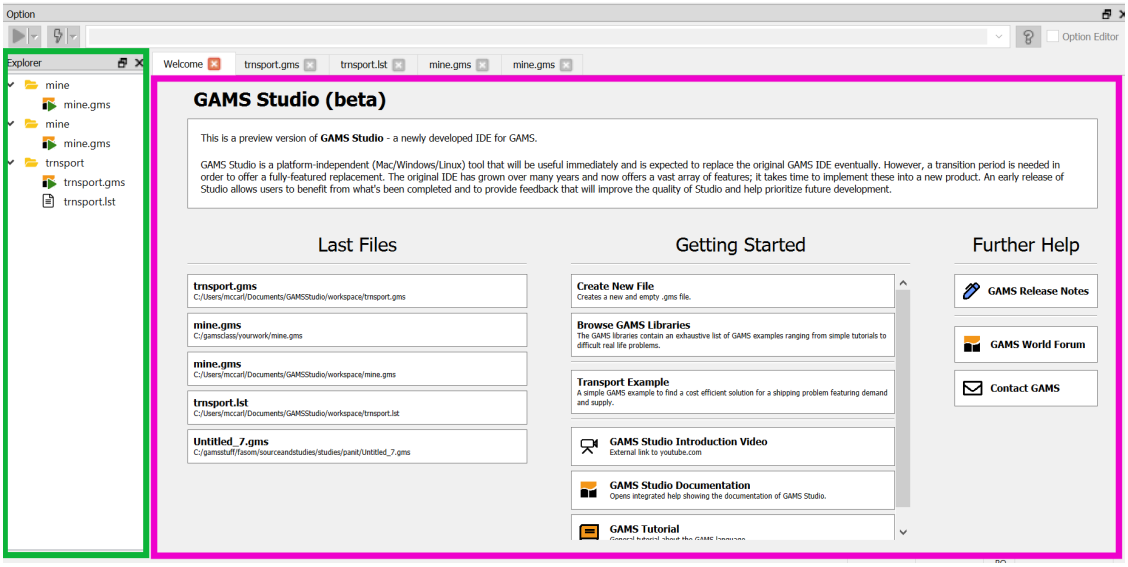
2. Open the **file** menu and use the **new** option. You will then be prompted for a name (I called it mine) with the extension .gms and then have a *blank screen* (green) into which you may type GAMS instructions



Save that file wherever you want and reopen it that will become the new default directory (**this only occurs upon an open not a save as**) and any subsequent non library opens, simple includes or saves will go there. Imports from the library go to the default workspace.

4.20.4 The Welcome and Explorer Windows

As you have worked with STUDIO you will find it shows a **Welcome** (magenta) window (*unless you told it not to in settings*), that remembers previous files plus gives access to documentation.



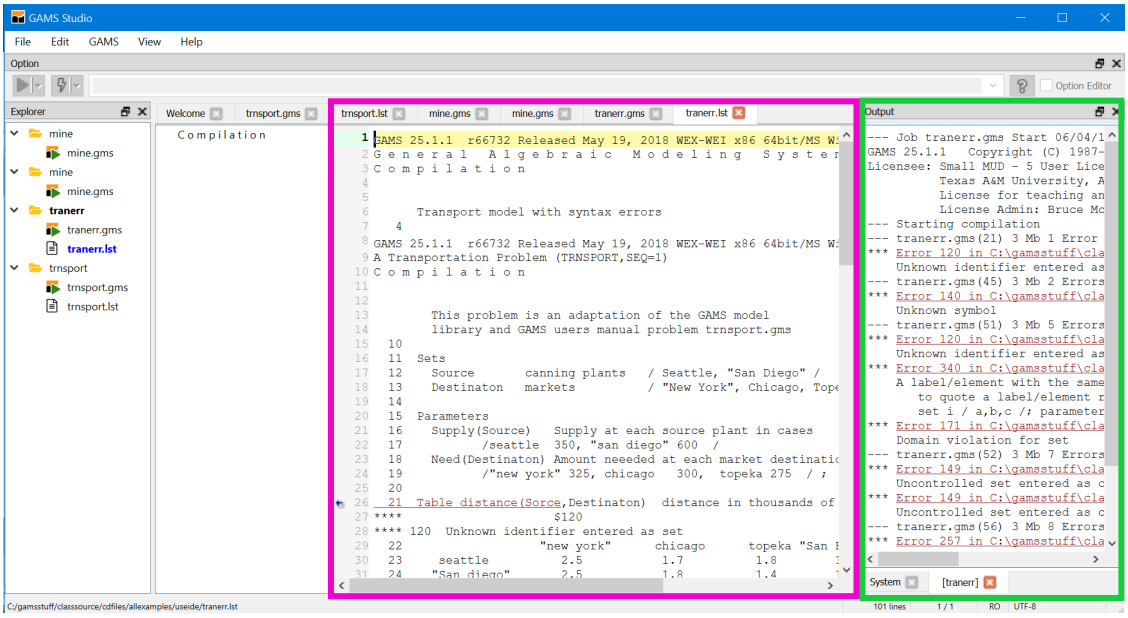
There as of now you get links to files recently used, and links to some documentation.

Also on the left hand side is an **Explorer** (green) window that shows recent files and allows you to open by double clicking. This remains whether you are in the welcome page or editing files unless it is not checked in the View dialogue.

4.20.5 Fixing Compilation Errors

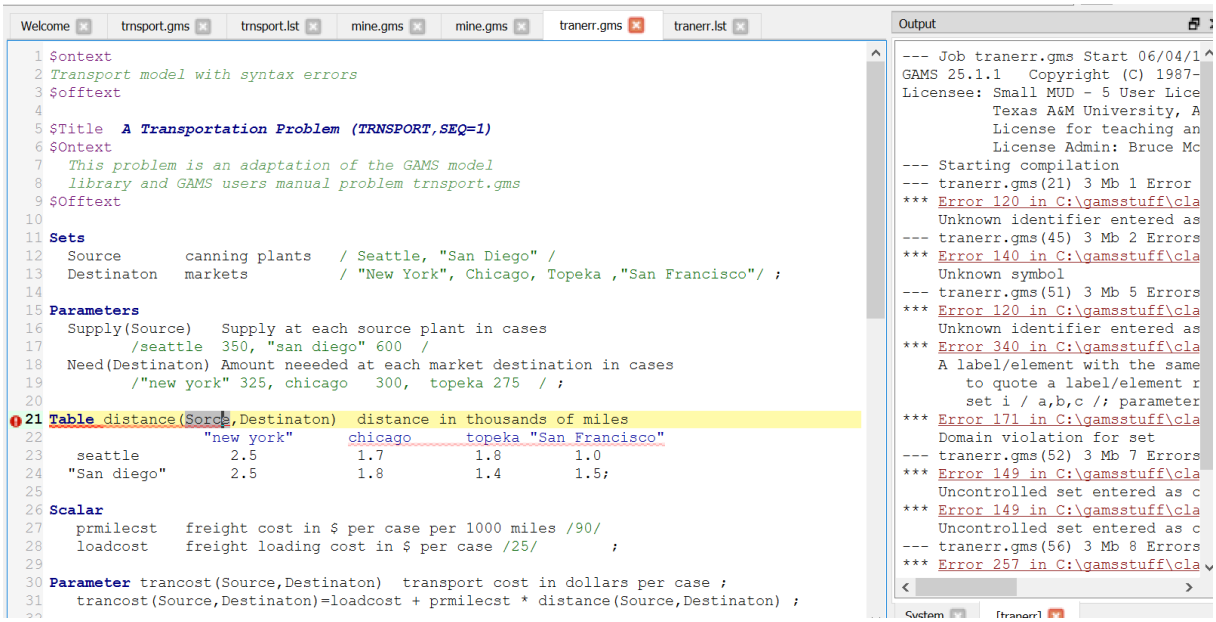
No one is perfect, errors always occur. The STUDIO can help you in finding and fixing those errors.

Let's look at an example with a small typo to illustrate (note: `source` vs. `sorce`). A run yields the windows below where the right part is the **process window** (green) positioned to show the first error and the center is the **lst file** (magenta) positioned to show the first error.



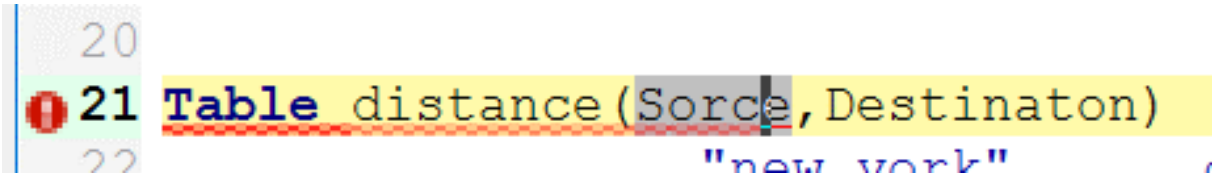
The red lines in the process window mark errors with error messages just below. To see where the errors occurred click on the very first red line which opens the source file or look at the LST file where the marking $ appears

A click on the red line **takes you to the place in the source where the error was made**. The tip here is **always start at the top of the process file when doing this**.

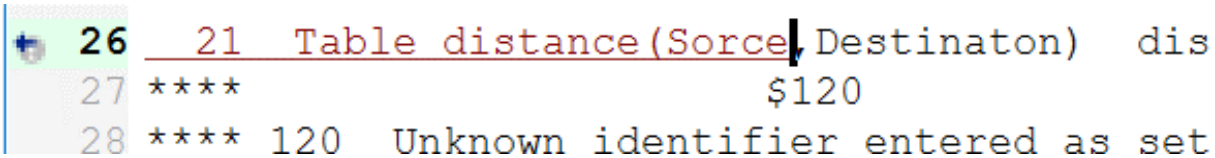


A double click on a black line takes you to the error location in the LST file.

Also in the source file window note the small red circle that appears on the line with the error



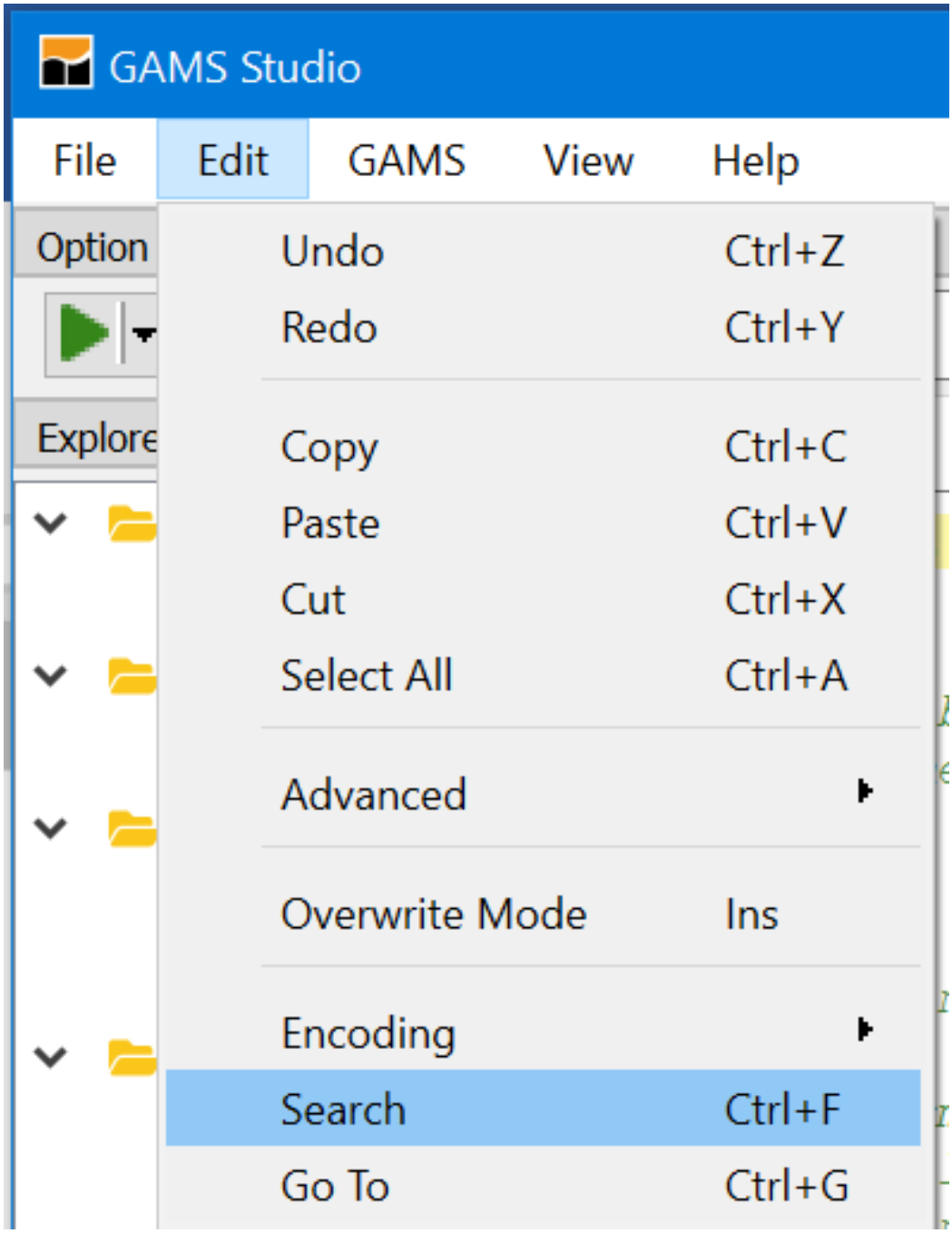
Clicking on that transfers you to the LST file where the error message appears.



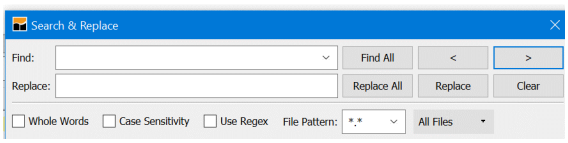
And there a small gray circle with an arrow in it that when clicked on will transfer you back to the source file

4.20.6 Ways to find and/or replace text strings

For finding you can use the search menu under the Edit tab or control F

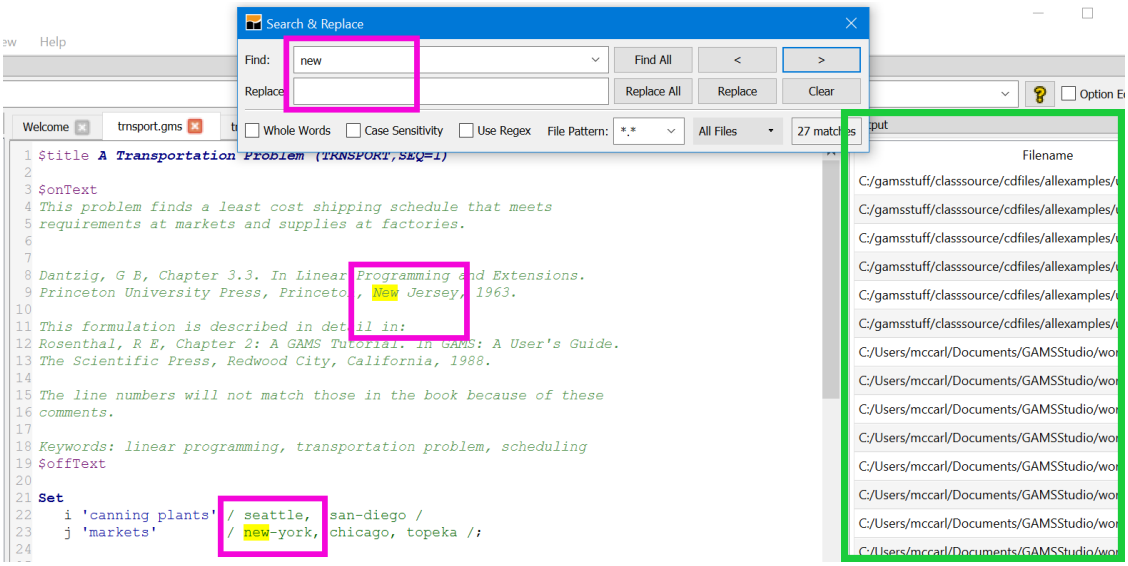


Subsequently you get



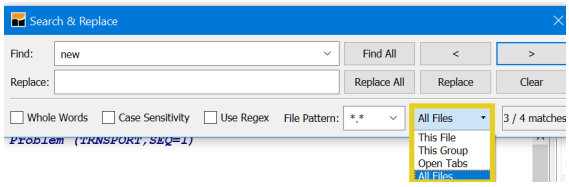
Which allows both search and replace. As of now this opens with the last option you used and may start with initially with an all files setting.

After the search you find



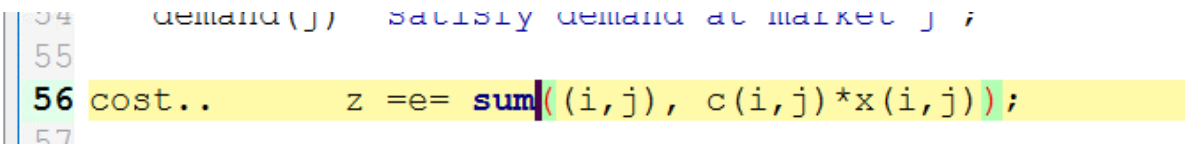
Where all occurrences of the **search term** in the current file are highlighted in yellow and to the right there is a **list of all files that contain that search term** (green) in all of the **directories corresponding to files opened in the explorer**.

The search dialogue is straight forward with arrow keys to jump to next (F3 also works for forward and shift F3 for backward) and the **all files can be manipulated as shown** (in yellow).

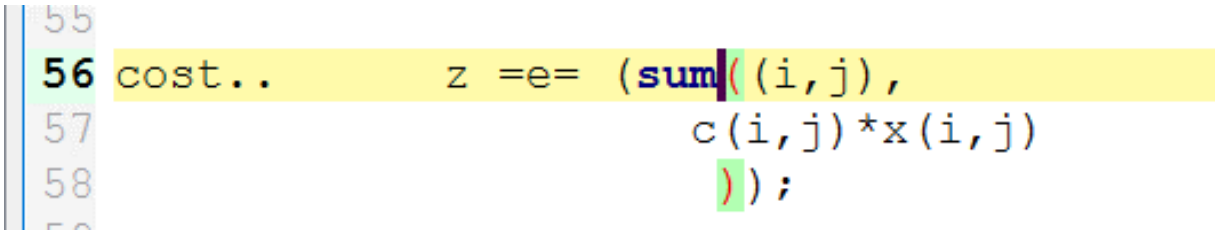


4.20.7 Matching Parentheses

The STUDIO provides you with a way of checking on how the **parentheses match up** in your GAMS code. This involves positioning your cursor just before an open parentheses or just after a close (Just after also works if there are not multiple parentheses). In either case the corresponding close or open is highlighted in **green**. Additionally positioning the cursor right before the beginning parentheses and tapping `Ctrl + b` will lead you to the matching ending parentheses whether it be 1, 100, or 1000+ lines away and vice versa.



When the expression in parentheses carries on the several lines the marking is in multiple lines and `Ctrl + b` will transfer position back and forth

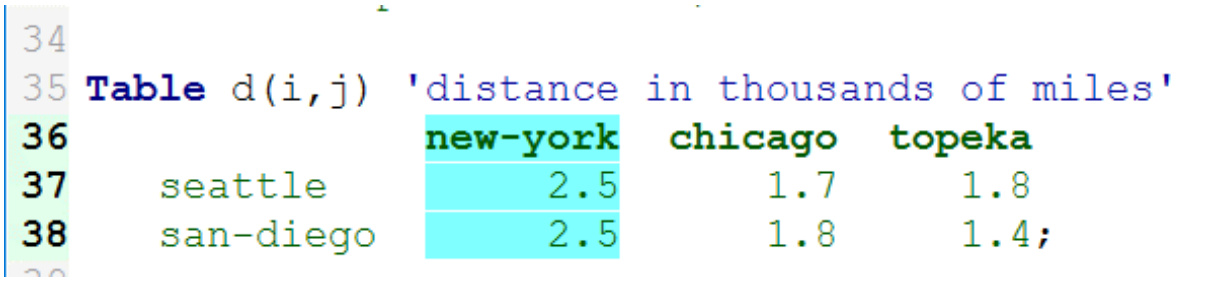


This feature will also match up { } or `[ ]` with a red marking if the types don't match.

4.20.8 Moving Blocks

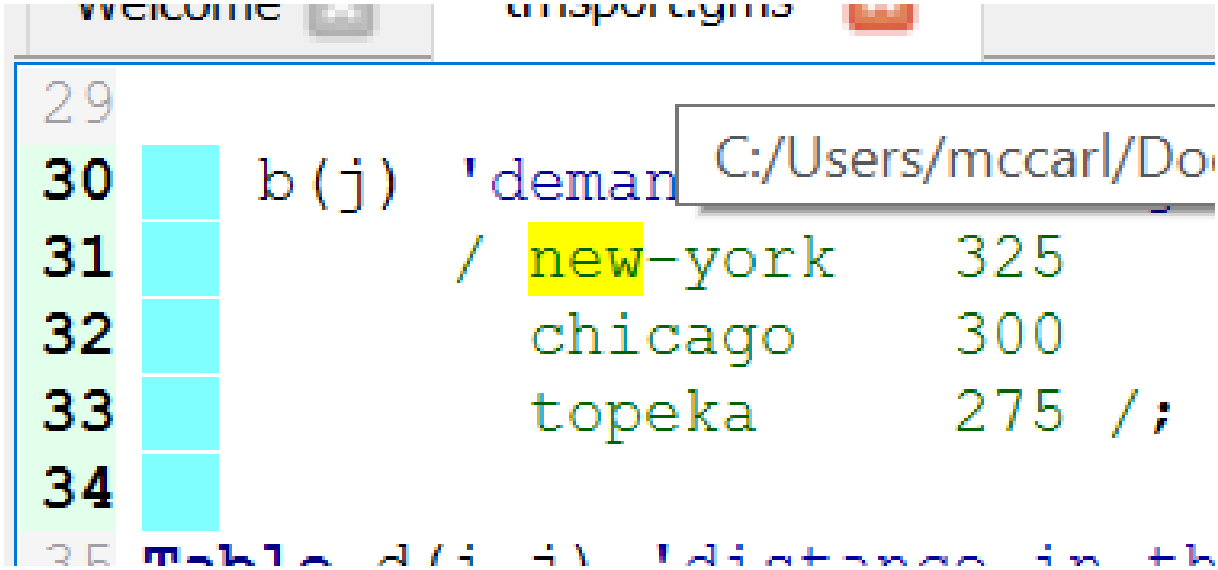
The STUDIO allows one to move text blocks through standard Cut and Paste Operations in two fashions.

1. One can identify a continuous block of text with the mouse or the keyboard (the latter involves putting the cursor at a beginning point then hold the shift key down and use the arrow keys).
2. One can identify a **column block of text** (blue) with the mouse or the keyboard (these involve holding **alt and shift down** then moving the mouse or the cursor with the arrow keys.



In turn copy, cut, and paste can be done with the Edit menu or with control c, x and v respectively as in normal windows. Control insert also pastes.

Also if one types with a column block highlighted than the character goes in all rows of the block and replaces what was there before



with an * typed becomes

```
29
30 *  b(j) 'demand at market j in cases'
31 *       / new-york    325
32 *         chicago     300
33 *         topeka      275 /;
34 *
```

4.20.9 Syntax Coloring

A feature in the STUDIO is syntax coloring. The STUDIO recognizes a subset of the GAMS syntax and reflects this in the display colors. Note in the display below that commands, explanatory text and set elements are differentially colored.

```
25 Parameter
26    a(i) 'capacity of plant i in cases'
27         / seattle    350
28           san-diego  600 /
29
30  b(j) 'demand at market j in cases'
31       / new-york   325
32         chicago    300
33         topeka     275 /;
34
35 Table d(i,j) 'distance in thousands of miles'
36                new-york  chicago  topeka
37    seattle          2.5      1.7     1.8
38    san-diego        2.5      1.8     1.4;
39
40 Scalar f 'freight in dollars per case per thousand miles' / 90 /;
41
42 Parameter c(i,j) 'transport cost in thousands of dollars per case';
43 c(i,j) = f*d(i,j)/1000;
44
45 Variable
46    x(i,j) 'shipment quantities in cases'
47    z      'total transportation costs in thousands of dollars';
48
49 Positive Variable x;
50
51 Equation
52    cost      'define objective function'
53    supply(i) 'observe supply limit at plant i'
54    demand(j) 'satisfy demand at market j';
55
56 cost..      z =e= [sum((i,j),
```

For now unlike in the IDE one cannot alter these syntax colors.

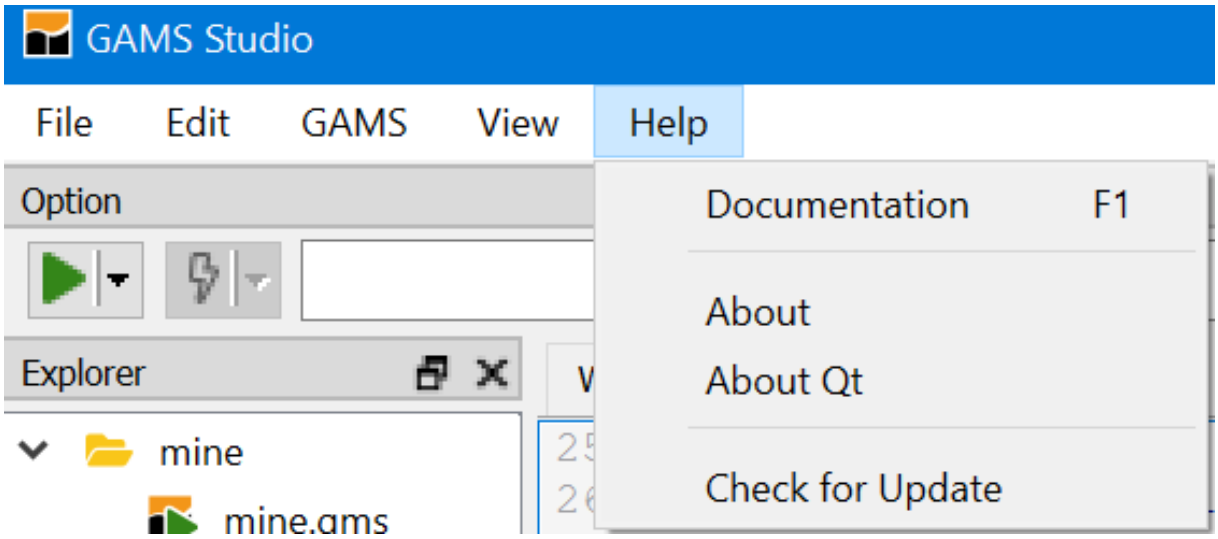
4.20.10 Showing where a symbol appears

Studio has a feature that identifies where a symbol is used. Taking the library file *agreste.gms* and positioning the cursor over a symbol like **c** (blue) then double clicking places a (gray) highlight over all visible instances of that symbol

```
$sTitle Model Definition
Variable
   xcrop(p,s) 'cropping activities                                (ha)'
   xliver(r)  'livestock activity defined on feed techniques (head)'
   xlive      'livestock production                            (head)'
   lswitch(s) 'land downgrading                                  (ha)'
   xprod(c)   'crop production                                  (ton)'
   cons(dr)   'on-farm consumption                              (ton)'
   sales(c)   'crop sales                                       (ton)'
   flab(tm)   'family labor                                (man-days)'
   tlab(tm)   'temporary labor                             (man-days)'
   plab       'permanent labor                              (workers)'
   rationr    'livestock ration requirements                     (cr)'
   pdev(ty)   'positive price deviations                         (cr)'
   ndev(ty)   'negative price deviations                         (cr)'
   yfarm      'farm income                                       (cr)'
   revenue    'from crop and livestock sales                     (cr)'
   cropcost   'accounting: cropping activities cost              (cr)'
   labcost    'accounting: labor costs - including family        (cr)'
   vetcost    'accounting: veterinary services cost              (cr)';

Positive Variable xcrop, xliver, lswitch, sales, cons, flab, tlab, plab,

Equation
   landb(s) 'land balance'
   lbal     'livestock balance                          ( )'
   rliv     'livestock ration requirements definition ( )'
   mbalc(c) 'material balance: crops                    ( )'
   dprod(c) 'crop production definition                 ( )'
   cond     'on farm consumption definition             ( )'
   labc(tm) 'labor supply-demand relation               ( )'
   ddev(ty) 'crop price deviation definition            ( )'
```

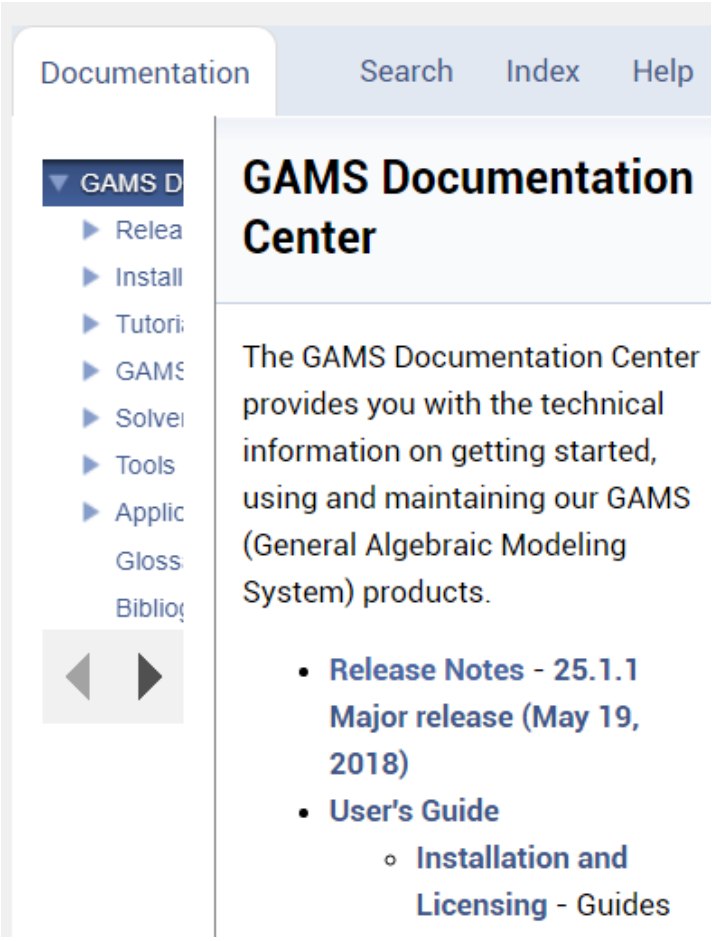
4.20.11 Accessing Documentation Via the Help

To access GAMS documentation choose help or F1



And if you choose documentation you get access through a window in lower right corner as shown. Therein you can navigate, search, and look at the index.

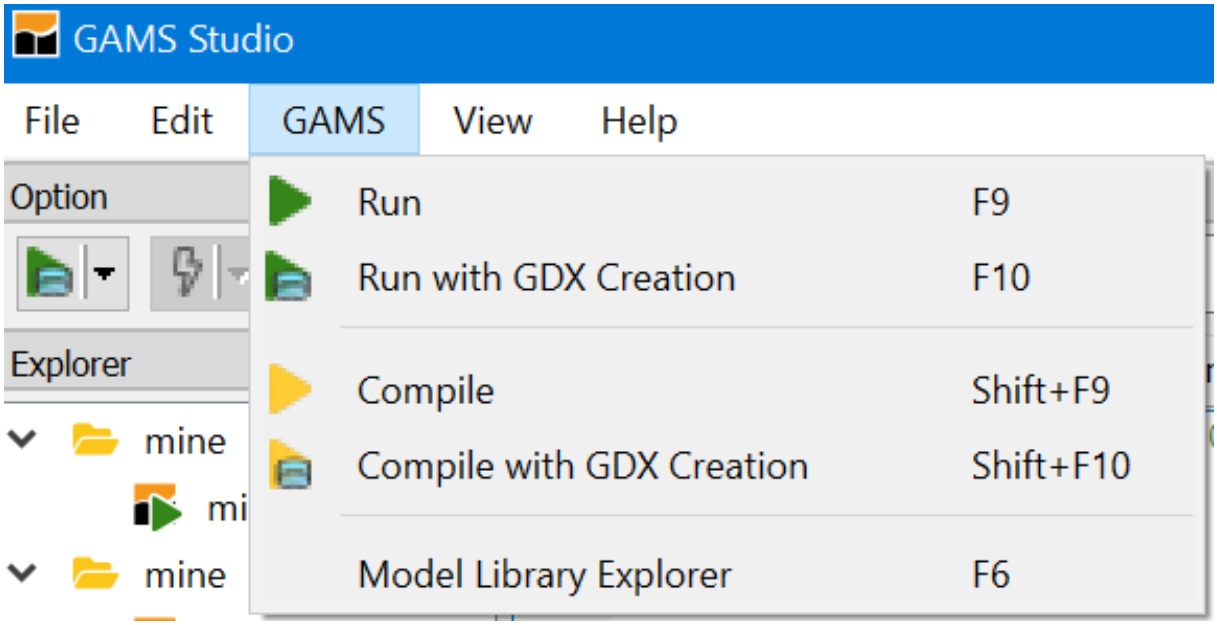
If your cursor is on a reserved word in the source file and you press F1 an index opens to all treatments of that word although in some strange places.



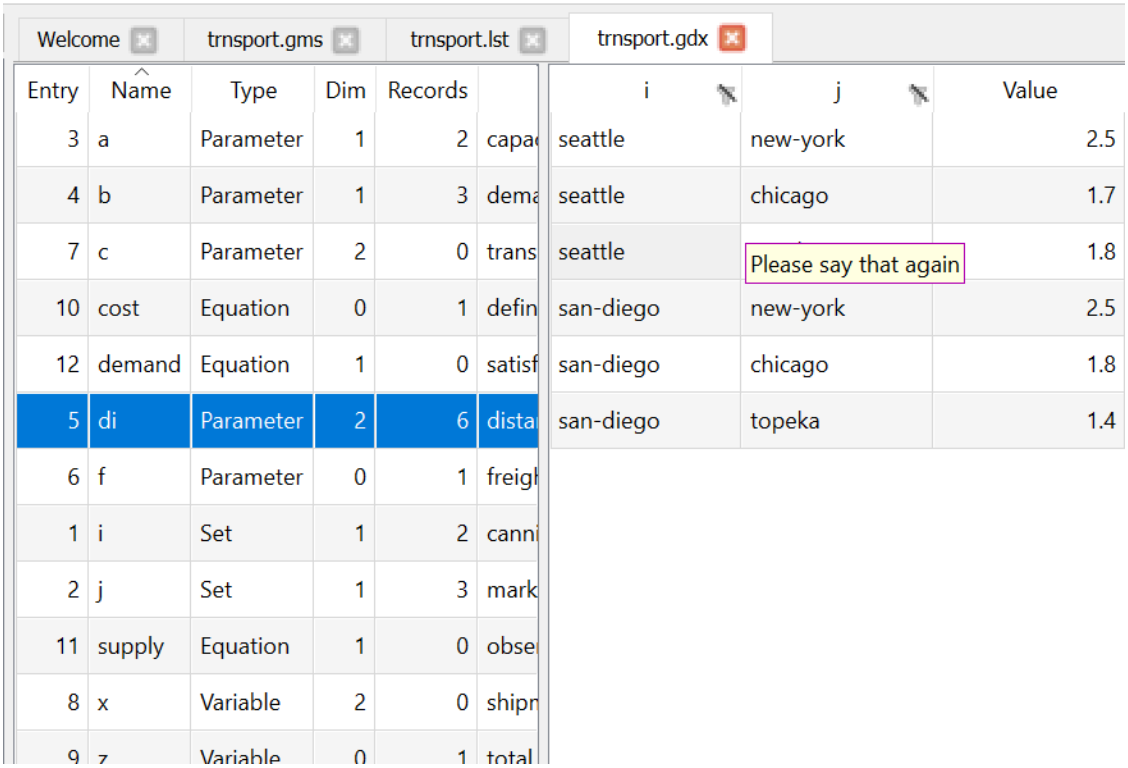
There is a document that explains features in the STUDIO and a `YouTube video` that can be accessed through the welcome page or in the general GAMS documentation under Tools.

4.20.12 Making GDX files

STUDIO can create and show GDX files. To create in STUDIO use the GAMS tab and choose a GDX creating choice. Alternatively use one of the other ways in the language as we will discuss elsewhere.



To browse a GDX file open it with the file open dialogue or click on green lines for ones created during a run in the process window and then for the transport example



Here in the left-hand panel one gets an alphabetical listing of the items in the GDX file and their characteristics. Then clicking on a left-hand panel item leads to the item contents being revealed in the right hand panel. As of now you can reorder columns and filter values then in table mode can move things in rows into columns and vice versa.

4.20.13 Examining GDX files

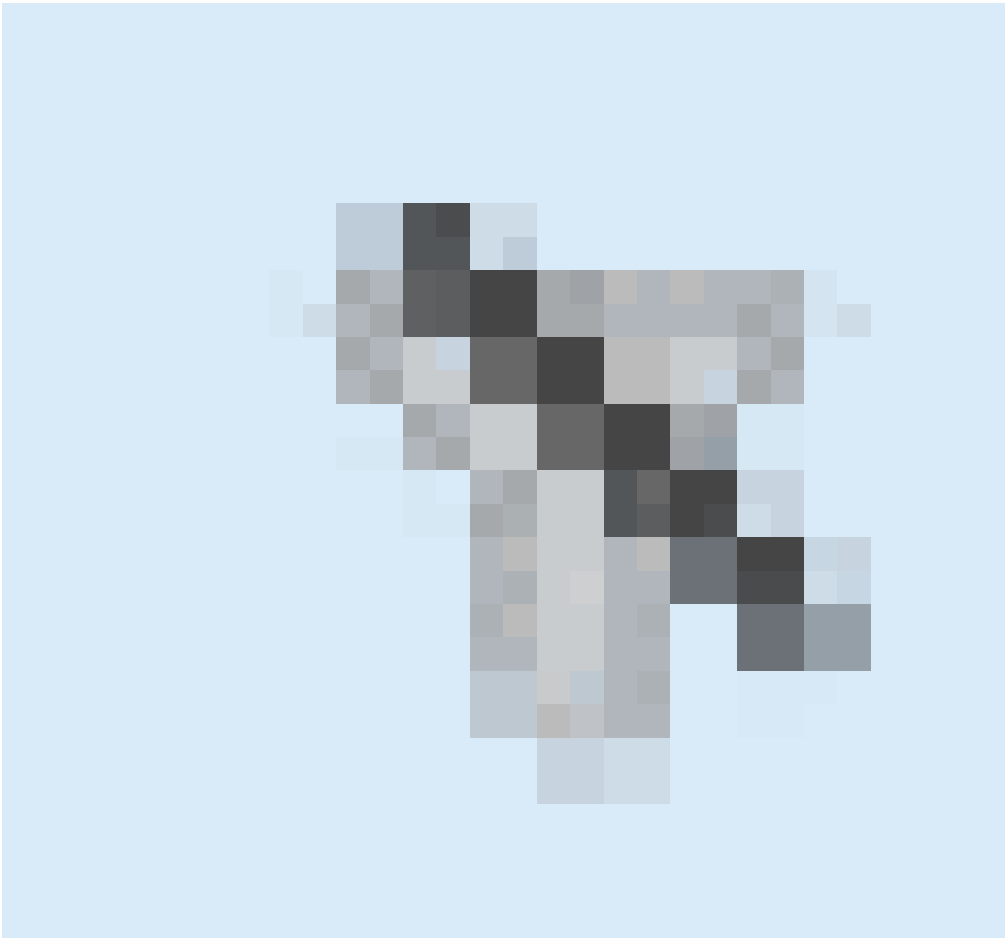
When you have opened a GDX file and selected an item then you get a window like that on the previous page

Entry	Name	Type	Dim	Records	Text	i	j	Value
3	a	Parameter	1	2	capacity of pla...	seattle	new-york	2.5
4	b	Parameter	1	3	demand at mar..	seattle	chicago	1.7
7	c	Parameter	2	0	transport cost i..	seattle	topeka	1.8
10	cost	Equation	0	1	define objectiv..	san-diego	new-york	2.5
12	demand	Equation	1	0	satisfy demand..	san-diego	chicago	1.8
5	di	Parameter	2	6	distance in tho..	san-diego	topeka	1.4

If in the right-hand panel you dragged **i to the right of value** (brown below) you get

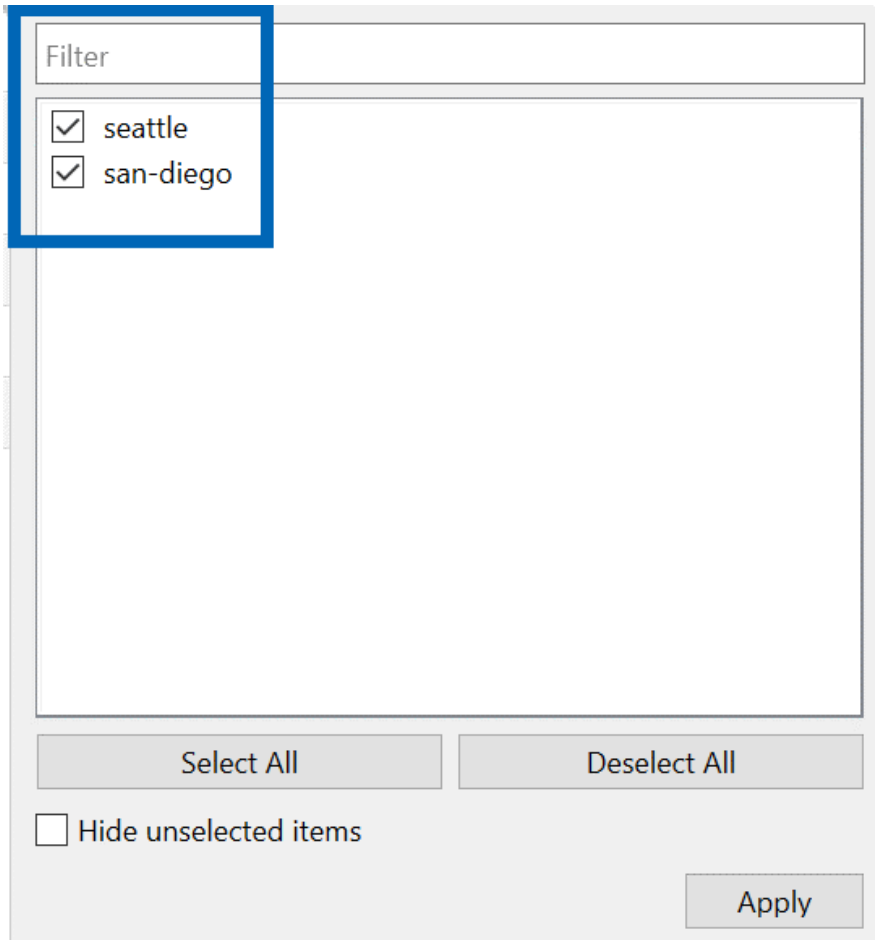
j	Value	i
new-york	2.5	seattle
chicago	1.7	seattle
topeka	1.8	seattle
new-york	2.5	san-diego
chicago	1.8	san-diego
topeka	1.4	san-diego

Also **by clicking on the header** you **cause the array to be sorted with a ^ marking the sort order** and **by clicking on**



you can **establish a filter** (green above and brown below) and **choose what elements to display** (blue below).

j	Value	i
chicago	1.7	seattle
chicago	1.8	san-diego
new-york	2.5	seattle
new-york	2.5	san-diego
topeka	1.8	seattle
topeka	1.4	san-diego



4.20.14 A difficulty you will have

When using and teaching both the IDE and STUDIO, I find that file locations sometimes gives me fits. I have a rule of thumb to avoid problems, but you will most likely inadvertently not follow it.

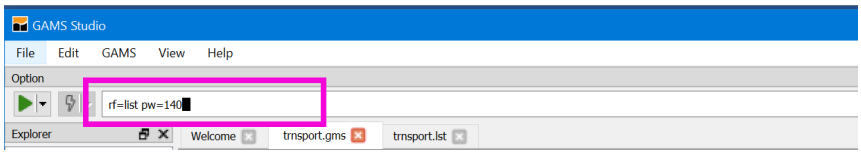
When working with a model with multiple files make sure they are located in the same directory location.

You do not have to follow this rule but deviations are the same as asking for trouble. When you are actively working on a file that you have opened from a directory then STUDIO will look for files in **the directory where active file is located**.

Also when you run a file which contains includes GAMS will look **in the directory where the active file is located**.

4.20.15 Command Line Parameters

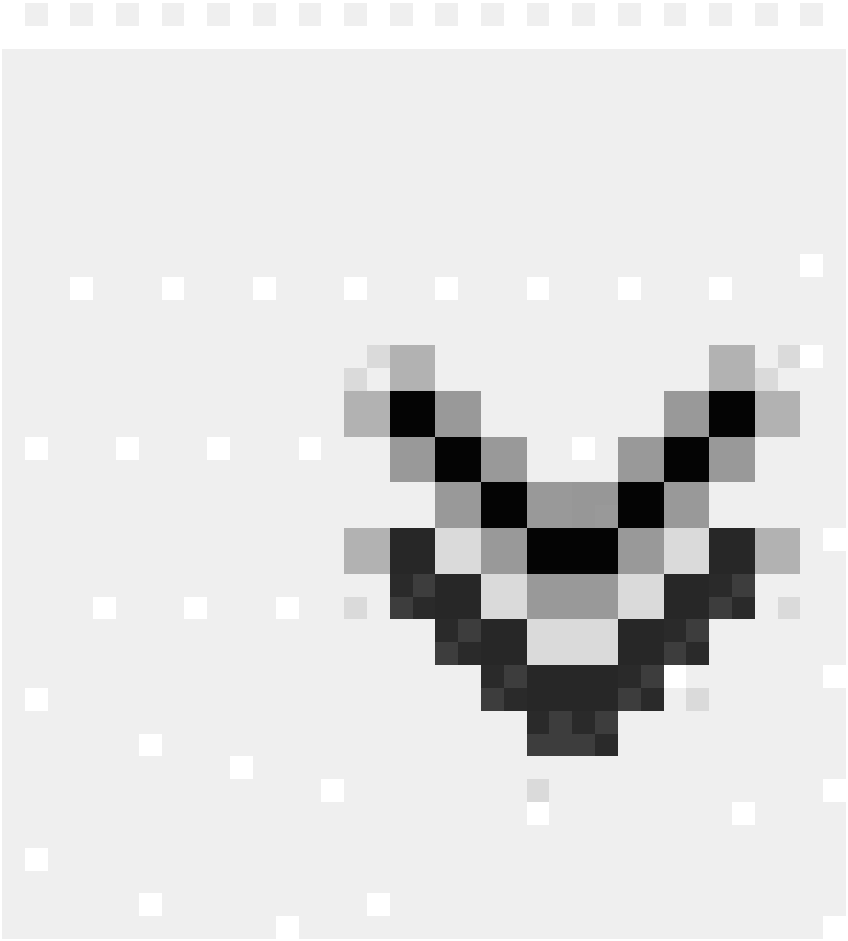
Experienced DOS or UNIX based GAMS users are used to having command line parameters associated with their GAMS execution commands. In STUDIO a **command line parameter entry box** (magenta) is available just to the right of the execute button. The parameters are associated with a file and STUDIO will remember these whenever the file is opened.



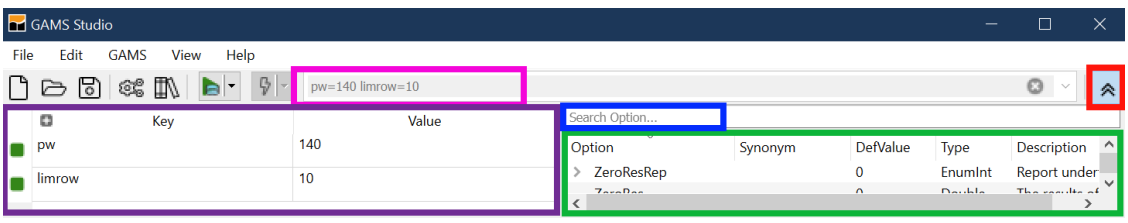
This is particularly useful for save, restart, limrow, limcol, pw and rf parameters as once they are defined they are associated with every subsequent use of the file.

As of now there is no way to provide command line parameters for use in every run (like a wide page width) as there was in the IDE.

GAMS provides assistance with command line parameters through the command line parameter editor. To activate it **check the**



symbol at the far right (red) up top. Then part of the screen becomes as below. In the left hand part **existing command line parameters** (purple) **in use** (magenta) **are listed along with their values**. On the right all possible **command line parameters are listed with definitions** (green) and a **search dialogue is provided** (blue).



The search dialogue looks for text in any of the fields, thus typing in **wid** brings up page width as does **pw** and **maxi** brings up those with maximum in the description.

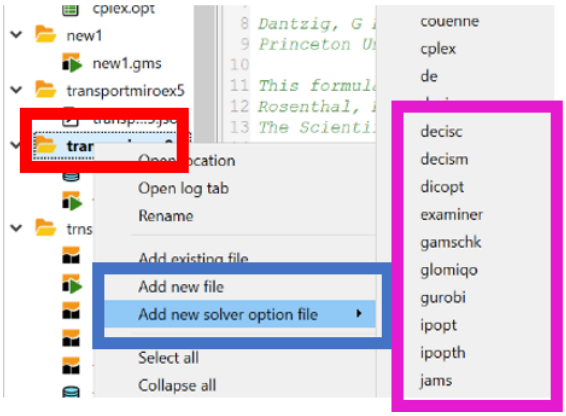
Note pressing F1 while the cursor is on a command line entry opens help to a place with a detailed description of that parameter.

4.20.16 Solver Option Files

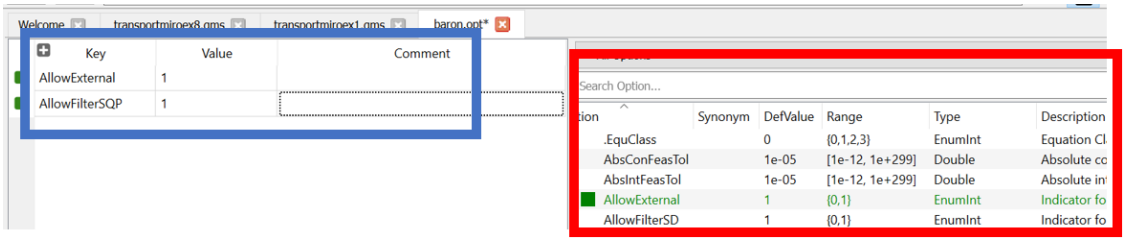
STUDIO allows you to edit or create a solver option file. When using the solver **Conopt** you create `conopt.opt` or with **Cplex** use `cplex.opt` (or `.op1` or `.o10` or `.111`) and more generally `solvername.opt`. Specifying such a file allows you to manipulate the way the solver works on the model. For example, you can alter scaling, tolerances, iteration limits, solver algorithm, and branch and bound strategy along with many other things.

Now in Studio the option file editor is accessed by either

- Right clicking on the **group name** (red) for a group where you want an option file then choosing **add new solver option file** (blue) and picking the **appropriate solver name** (magenta)

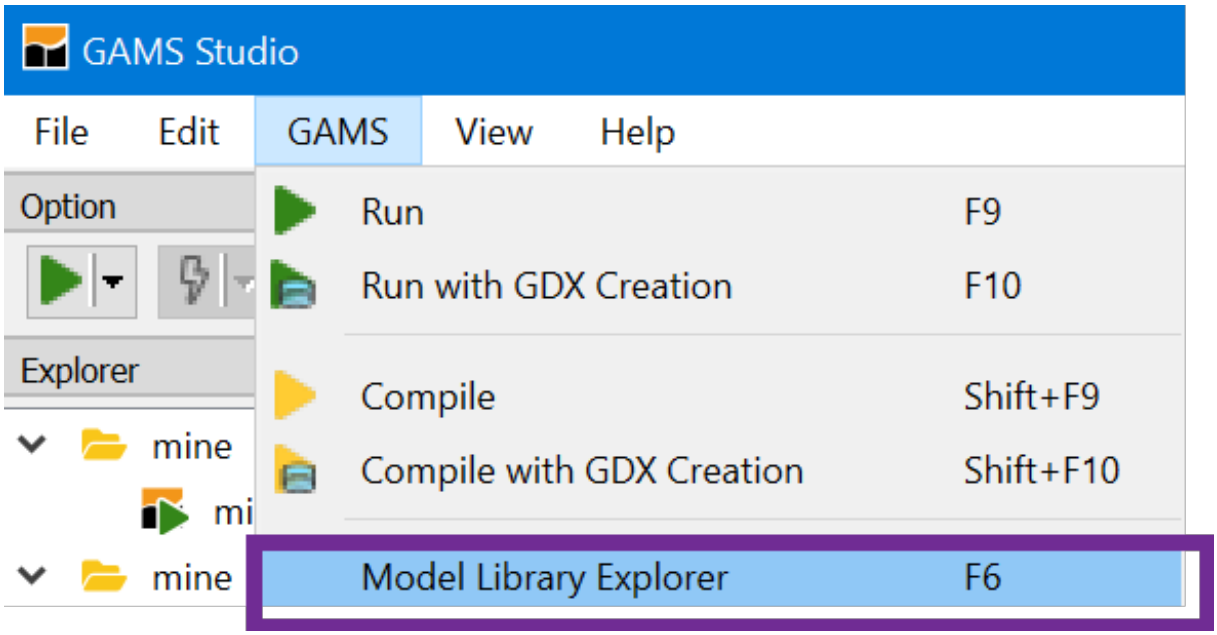


- Creating a file like `baron.opt` then when you bring that up in the editor the option file editor appears. In the editor you see **options in use** (blue), **possibilities and definitions** (red)

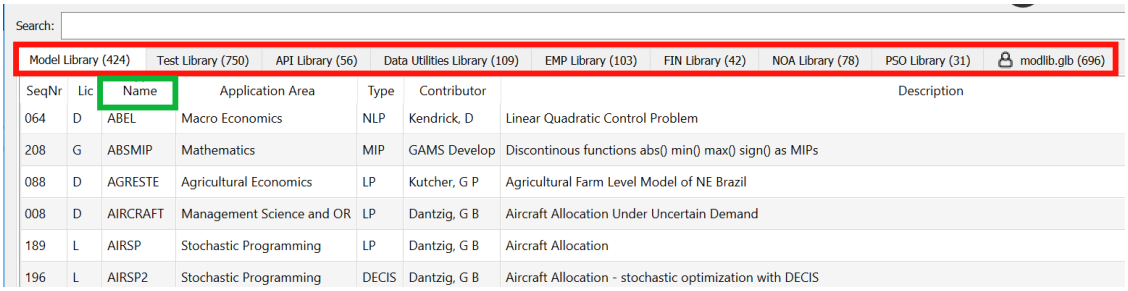


4.20.17 Using Libraries

STUDIO allows one to access libraries under the GAMS menu item and choosing the **Model Library Explorer** (purple).



In turn this opens a screen which identifies names of the libraries in tabs with a count of included files and, for the tab selected, the **library contents**. Furthermore across the top are tabs that when clicked on cause sorts. The picture below came after I clicked on the tab name with the model library active



SeqNr	Lic	Name	Application Area	Type	Contributor	Description
064	D	ABEL	Macro Economics	NLP	Kendrick, D	Linear Quadratic Control Problem
208	G	ABSMIP	Mathematics	MIP	GAMS Develop	Discontinous functions abs() min() max() sign() as MIPs
088	D	AGRESTE	Agricultural Economics	LP	Kutcher, G P	Agricultural Farm Level Model of NE Brazil
008	D	AIRCRAFT	Management Science and OR	LP	Dantzig, G B	Aircraft Allocation Under Uncertain Demand
189	L	AIRSP	Stochastic Programming	LP	Dantzig, G B	Aircraft Allocation
196	L	AIRSP2	Stochastic Programming	DECIS	Dantzig, G B	Aircraft Allocation - stochastic optimization with DECIS

Users can define their own libraries. They have to define what's called a GLB file and the formatting creation of this is discussed in my newsletter in the second issue: https://www.gams.com/mccarl/newsletter/news2.htm.

4.20.18 Using reference files - Listing and Unraveling Data items

GAMS has a command line parameter to create reference files (aka. `.ref` files) called reference (short: rf). These files give information on where things are defined and used.

Why? GAMS Modelers sometimes have to deal with complex implementations that

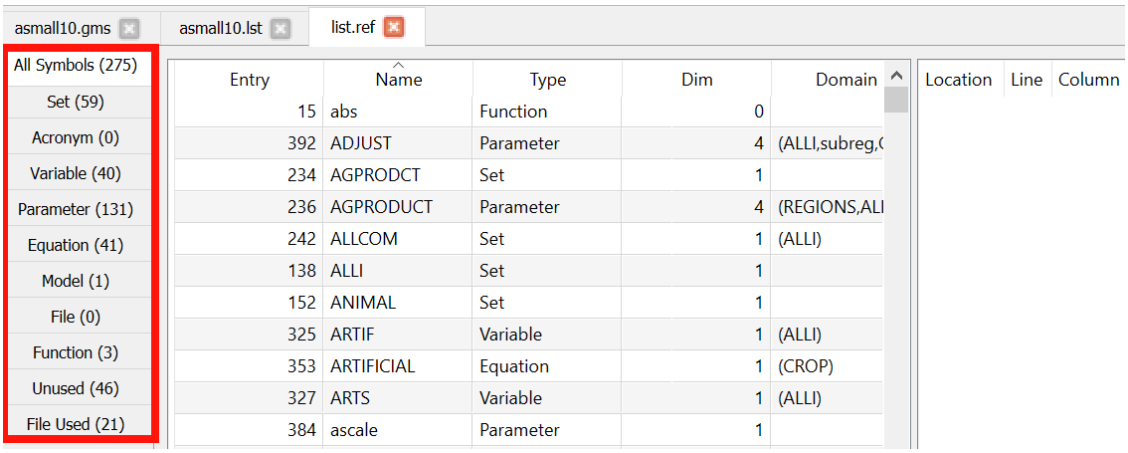
- Use include statements to incorporate numerous files.
- Have been developed by others.
- Have a complex structure with definitions and uses of items widely spread in a file or files.
- Contain items that are defined but never used.
- Were developed some time ago but are not extensively documented.

When faced with such cases one often asks

- Are there items defined in the program that are not used and if so what are they and where are they?
- In what files is an item defined, declared and used?

To resolve these questions some years ago I wrote a Fortran program called GAMSMAP but never made it widely available. Subsequently it was included into the GAMSIDE and then STUDIO.

When a `.ref` file is opened it creates a window as follows

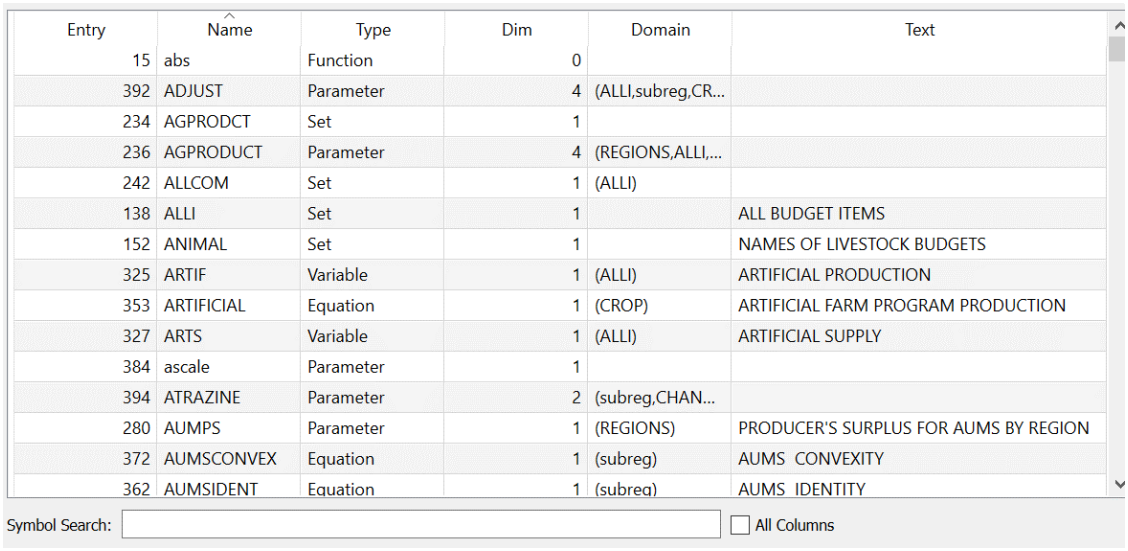


The **column on the left** (red) gives tags that can be clicked on and the key ones are

Column	Description
All Symbols	a list of items that are declared and a count of them
Set	a list of the sets that exist in the program and a count
Variable	a list of the variables in the program and a count
Parameter	a list of the parameters (items defined in scalar, parameter or table statements) and a count.
Equation	a list of the equations in the program and the names of files in which they appear
Model	a list of the models that exist in the program and the names of files in which they appear
Unused	a list of items that are declared or that are not used anywhere in the model
File used	a list of the files included into the model

4.20.18.1 All Symbols Tab

The all symbols tab causes the output to appear as follows:



Entry	Name	Type	Dim	Domain	Text
15	abs	Function	0		
392	ADJUST	Parameter	4	(ALLI,subreg,CR...	
234	AGPRODCT	Set	1		
236	AGPRODUCT	Parameter	4	(REGIONS,ALLI,...	
242	ALLCOM	Set	1	(ALLI)	
138	ALLI	Set	1		ALL BUDGET ITEMS
152	ANIMAL	Set	1		NAMES OF LIVESTOCK BUDGETS
325	ARTIF	Variable	1	(ALLI)	ARTIFICIAL PRODUCTION
353	ARTIFICIAL	Equation	1	(CROP)	ARTIFICIAL FARM PROGRAM PRODUCTION
327	ARTS	Variable	1	(ALLI)	ARTIFICIAL SUPPLY
384	ascale	Parameter	1		
394	ATRAZINE	Parameter	2	(subreg,CHAN...	
280	AUMPS	Parameter	1	(REGIONS)	PRODUCER'S SURPLUS FOR AUMS BY REGION
372	AUMSCONVEX	Equation	1	(subreg)	AUMS CONVEXITY
362	AUMSIDENT	Equation	1	(subreg)	AUMS IDENTITY

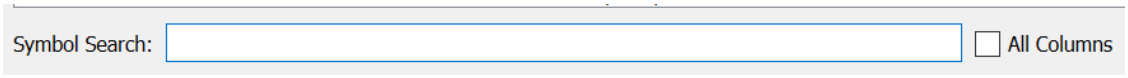
Symbol Search: All Columns

This shows all symbols used. The columns give

- the symbol name (Name),
- it's type(Type),
- its dimension (Dim),
- the sets over which it is dimensioned (Domain)
- the explanatory text used in it's declaration (Text).

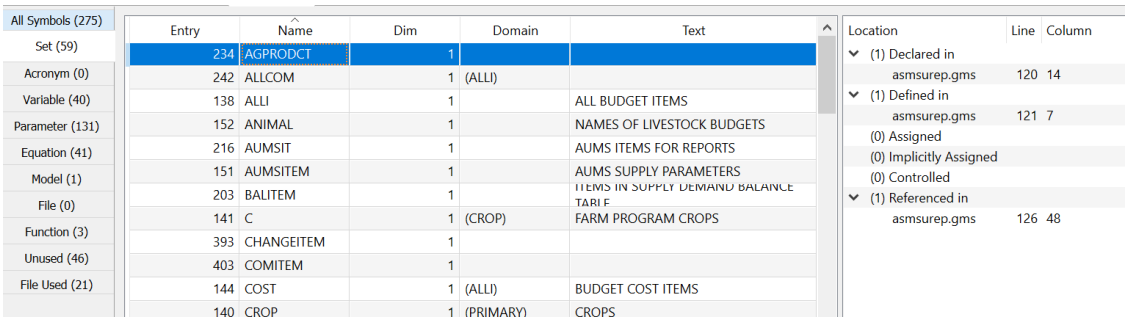
Things can be sorted by clicking on the column headings.

One can search for symbols using the box at the bottom



4.20.18.2 Sets, Parameters etc. Tabs

The display contains 5 lines that when activated give information for sets, parameters etc. The resultant display lists all items falling in a class (for all things that are sets or parameters etc.) and information on their characteristics and use. Specifically one gets output as follows (for the sets in this case)

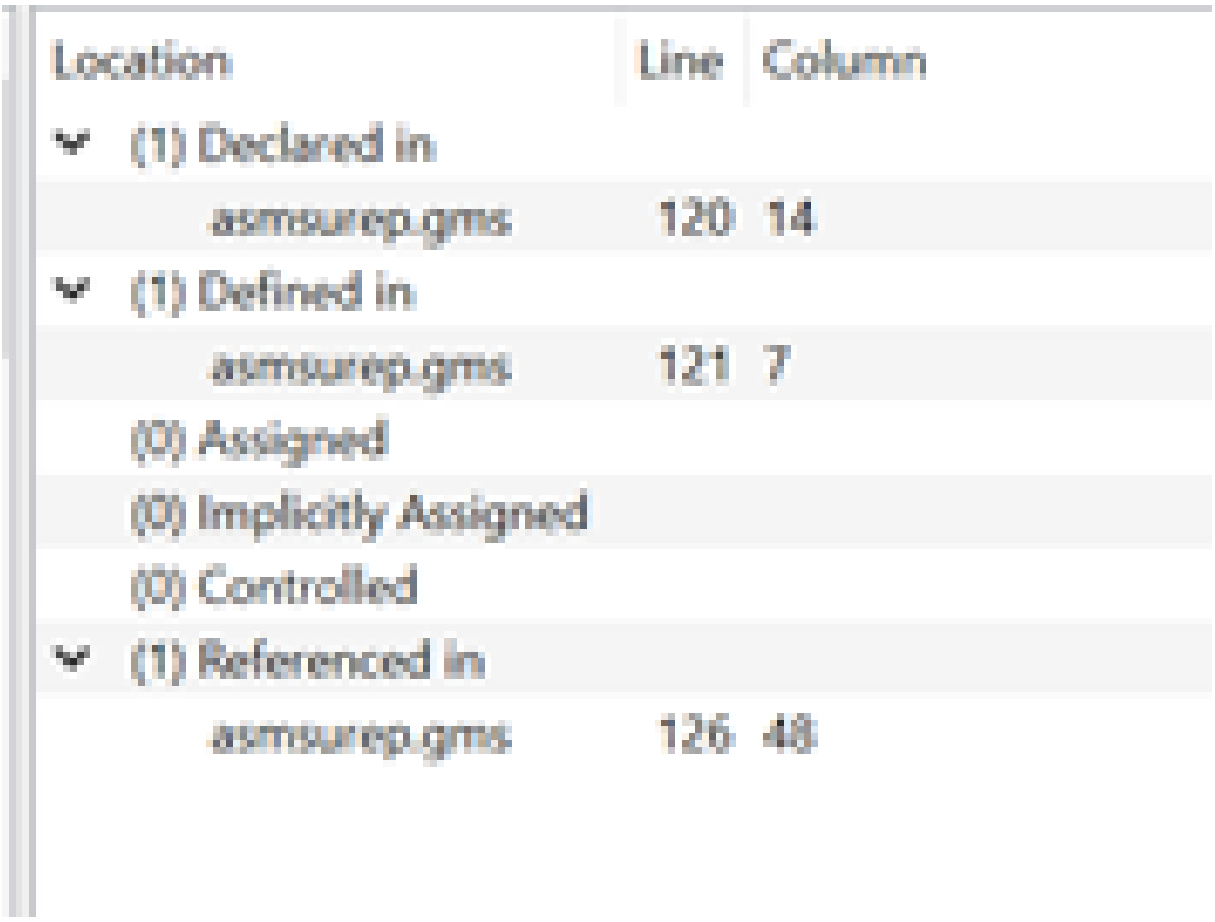


In this output once clicked on the entries tell the names of the files in which certain things happen relative to the identified items. The categories of things include the file name, line and column where items are declared, defined and used as discussed below.

Clicking on a line takes you to the file and place where the item appears.

Clicking on a column name causes it to be sorted.

After clicking on an item one gets



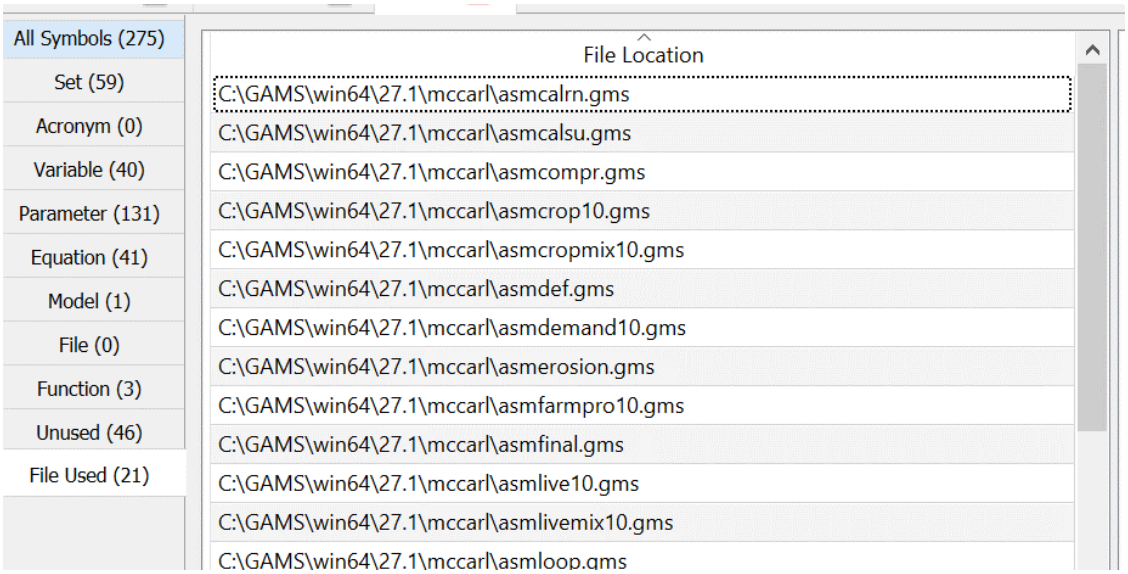
Entries in location columns identify files where items are

Column	Description
Declared	places where the named item is declared in a Set, Parameter, Table, Scalar, Variable, Equation, Acronym, File or Model command. This will be the first appearance.
Defined	places set elements or data are explicitly entered. For equations this tells where the .. specification begins.
Assigned	places where items appear on left hand side of an assignment statement
Implicitly-Assigned	places where an equation or variable has data put into it by the results of a solve statement.
Controlled	places where set is used in controlling a sum or defining an equation
Ref	places where item is on left hand side of assignment statement or in a model equation

Clicking on lines here opens the subject file in the specific place

4.20.18.3 File used Tab

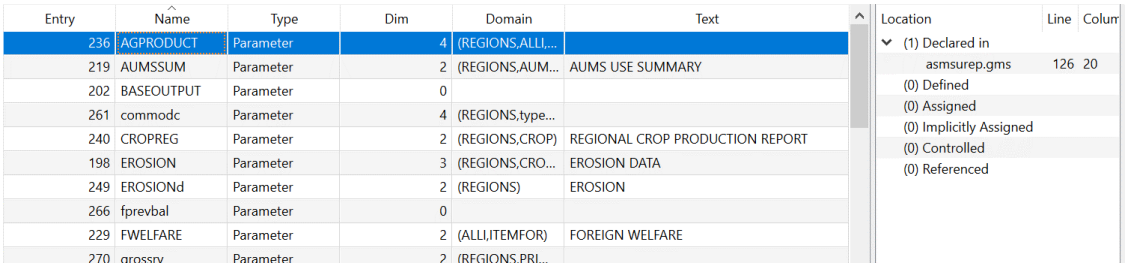
The file used tab causes the output to appear as follows:



This gives the names of the files included in the program with their full path references.

4.20.18.4 Unused Tab

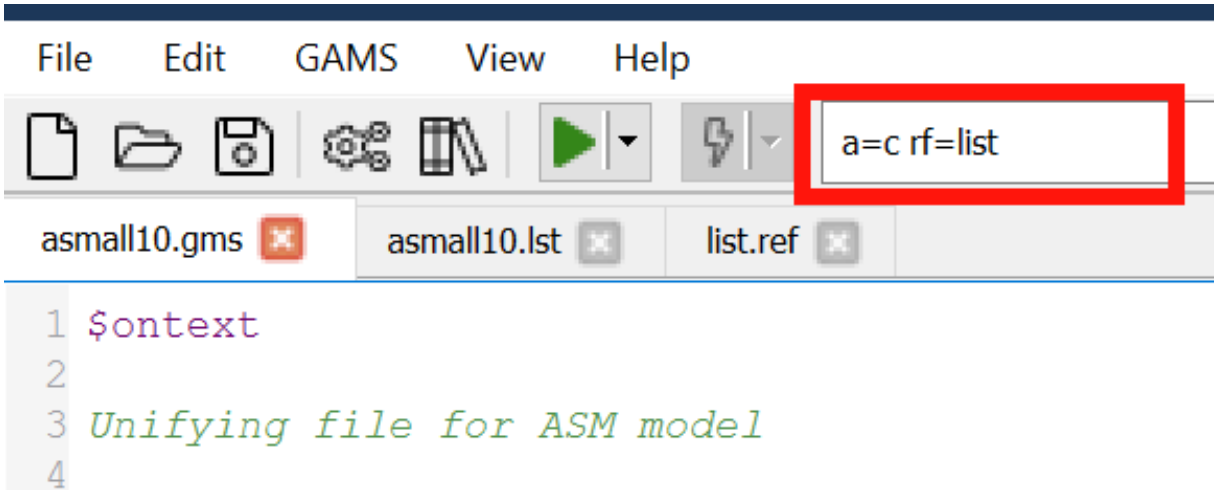
The unused tab identifies items that are declared (in set parameter etc. statements) but are never used on the right hand side of an assignment (=) statement or in a model equation:



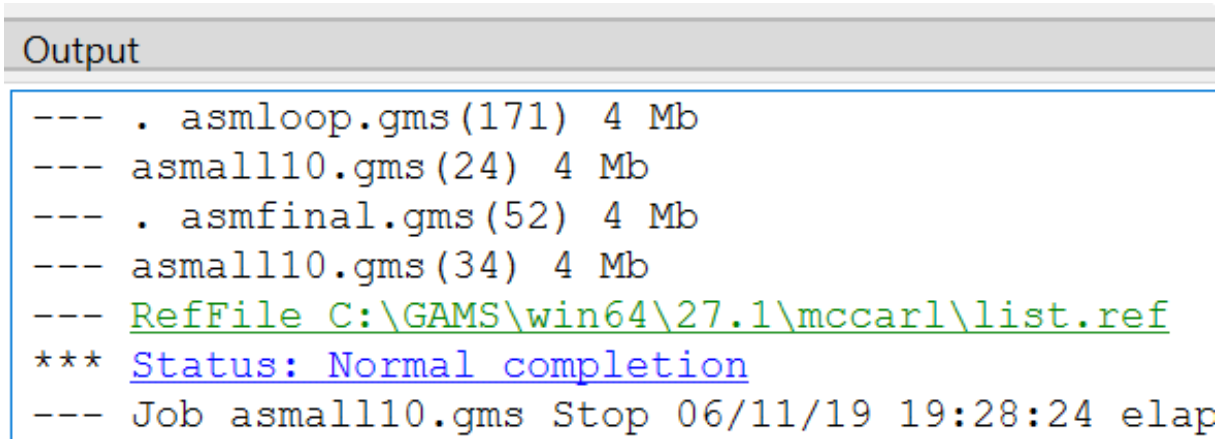
which shows the same information as in the All Symbols tab except when an item is clicked on one gets information on the files where it is declared.

4.20.18.5 Steps to use the Reference File Viewer

The Reference File Viewer will only work after a particular "reference file" has been created by a GAMS run. The file is generated by adding the `rf` option to the command line call entering a command in the command line box of Studio as follows (a=c cause compilation only)



Once the file has been run with the `rf` option the logfile is augmented with the (green) line identifying the `.ref` file name as below



Double clicking on this opens the Reference File Viewer window.

The run of GAMS with the `rf` option places the name of the `.ref` file in green in the GAMS log file and double clicking on that line causes the Reference File Viewer to be opened.

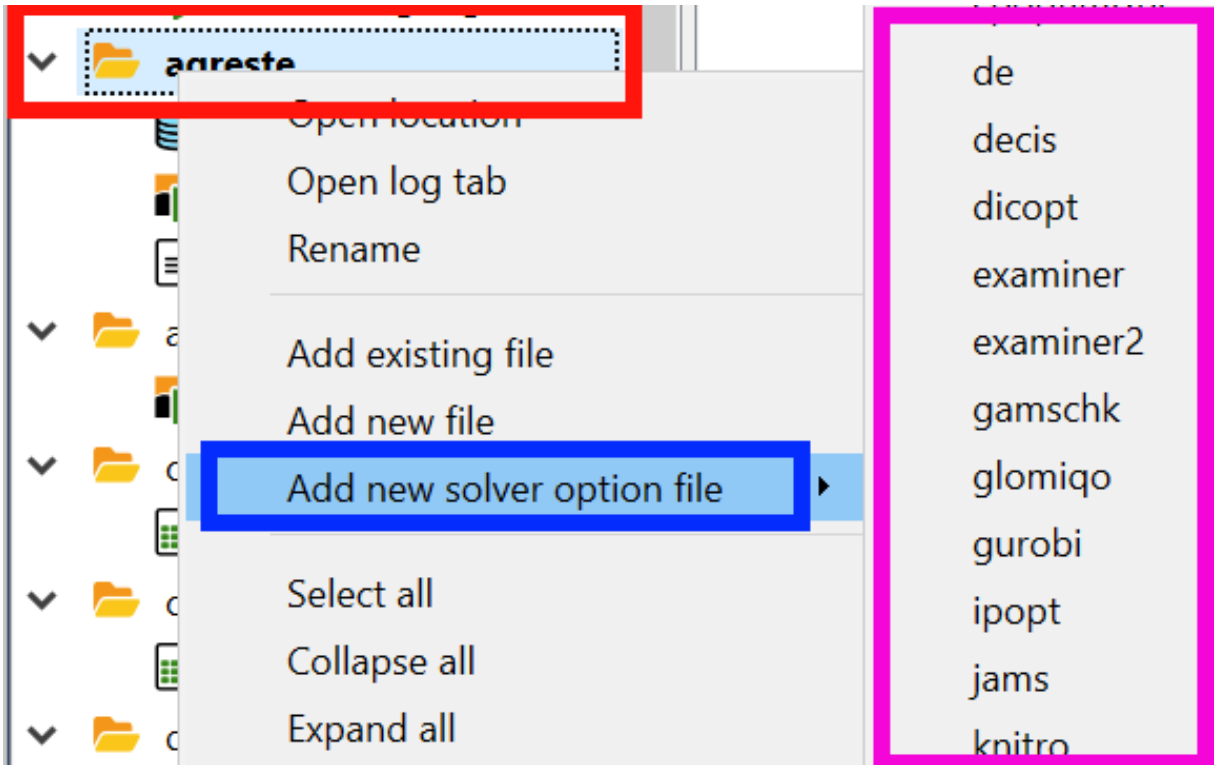
Several notes about the run

- The `rf=` command specifies the name of the file the Reference File Viewer will use. Generally, the Reference File Viewer expects it to have the extension `.ref`.
- The `.ref` file only covers the program components in a run and does not include any information from restart files. In general it is best to explicitly use all the files in one program without use of save and restart.
- It is often useful to just generate the reference file without any execution on behalf of the GAMS program. This is done by including the `a=c` option on the command line or in the command parameter IDE box.

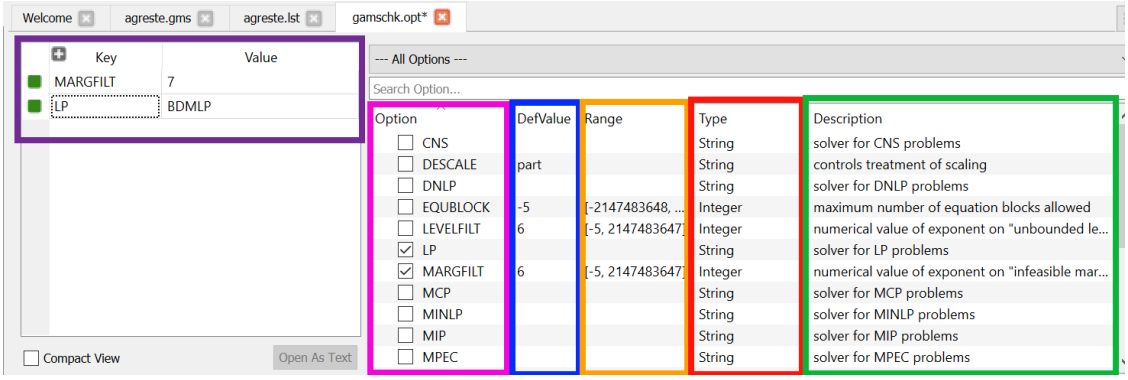
4.20.19 Editing Solver Option Files

Sometimes it is desirable to change solver specific options. This includes modifying things like branch and bound strategy, degree of scaling, barrier versus simplex solution approach, solution tolerances etc. The specific options allowed are described in the solver manuals. In addition Studio contains a solver option file editor that presents a solver specific menu of all possible options, default values and ones that have been altered. This is activated by either

- Opening or saving a file with solvername.opt (Like MINOS.opt)
- Right clicking on a **group file name** (red) in the Studio Explorer then selecting **add new solver option file** (blue) and subsequently selecting one of the many available **solver names** (magenta and only a few shown). This will create a file you then need to save.



Once it opens the general format of the solve option editor is as follows. Note the specific content depends on the solver chosen and in this case I use GAMSCHK.



Here in the right part we see a list of all **possible options** (magenta) for this solver with boxes checked for those that are in the option file. A brief description of **default values** (blue), **entry type** (red), if relevant **allowable settings** (yellow) and a **one sentence description of the option** (green). In the left part we see the **current entries** (purple) in the option file and **their settings** (purple).

Subsequently when this file is opened or its tab activated then this solver option editor appears.

Note, pressing F1 while on an option opens help to a place with a detailed description of that option.

4.20.20 When is it Not Worth Using?

There are costs and benefits of these approaches.

The STUDIO is much easier for simple models and can run across platforms.

The DOS/command line approach is generally better for models in customized environments.

A **development strategy** for more complex implementations

1. Use the STUDIO to get it right
2. Debug components of large models using save and restart
3. Then use script files or DOS/UNIX with batch files such as
   - `GAMS mymodel  -lo 0 -s ./t/save1`
   - `call myprogram.exe`
   - `GAMS moremod  -lo 0 -r ./t/save1`

4.20.21 What does it not do?

A number of IDE features are currently missing. Just to inventory (note the last three are pretty unimportant to me.)

- Ability to save output from the reference files
- Ability to change the file associations when clicking on them in Windows Explorer (you get more than I like at the moment and sometimes I want more)
- Project files (mixed blessing)
- Text file differencing (can use other programs)
- Spell checking

4.20.22 What does it not do so well?

Here are also some difficulties with it as of today

- Washed out colors – on my machine the syntax and error coloring is not very prominent.
- Excessive library choices and no way to put the ones you have no interest in somewhere out of the way. Plus any user defined libraries are at the end and there is no way to reorder.

4.21 Introduction

4.21.1 Summary

Today, algebraic modeling languages are widely accepted as the best way to represent and solve mathematical programming problems. Their main distinguishing features are the use of relational algebra and the ability to provide partial derivatives on multidimensional, very large and sparse structures. In this chapter we will briefly describe some of the origins of GAMS and provide background information that shaped early design decisions.

4.21.2 The Origins of GAMS

The initial Research and Development of GAMS was funded by the International Bank for Reconstruction and Development, usually referred to as The World Bank. Since 1987, further Research and Development has been funded by GAMS Development Corporation. GAMS was developed in close cooperation of mathematical economists who were and still are an important group of GAMS users. The synergy between economics, computer science and operations research was the most important success factor in the development of the system. Mathematical Programming and economics theory are closely intertwined. The Nobel Prize in Economics awarded to Leonid Kantorovich and Tjalling Koopmans in 1975 for their "contribution to the theory of optimal allocation of resources" was really a prize in mathematical programming. Other Nobel laureates like Kenneth Arrow in 1972, Wassily Leontief in 1973, and Harry Markowitz in 1990 are well known names in math programming. Another early example of this synergy is the use of LP in refining operations, which was started by Alan Manne, an economist, with his book on Scheduling of Petroleum Refinery Operations in 1956.

The origins of linear programming algorithms all go back to George Dantzig's early work in the 1940s and 1950s. Computing technology and algorithmic theory had developed at a rapid pace. Thirty years later, we could solve problems of practical size and complexity that allowed us to test of the economic theory on real life problems. The research agenda at the World Bank in the 1970s and 1980s created the perfect environment to bring different disciplines together to apply mathematical programming to research and operational questions in Economic Development.

4.21.3 Background and Motivation

From the very beginning, the driving force behind the development of the General Algebraic Modeling System (GAMS) has been the users of mathematical programming who believed in optimization as a the powerful and elegant framework for solving real life problems in the sciences and engineering. At the same time, these users were frustrated with the high cost, skill requirements, and overall low reliability of applying optimization tools. Most of our initiatives and support for new development came from the worlds of economics, finance, and chemical engineering. These disciplines find it natural to view and understand the world and its behavior as a mathematical program.

GAMS's impetus for development arose out of the frustrating experiences of a large economic modeling group at the World Bank. In hindsight, one may call it a historical accident that in the 1970s mathematical economists and statisticians were assembled to address problems of development. They used the best techniques available at the time to solve multisectoral economy-wide models and large simulation and optimization models in agriculture, steel, fertilizer, power, water use, and other sectors. Although the group produced impressive research, initial successes were difficult to reproduce outside their well functioning research environment. The existing techniques to construct, manipulate, and solve such models required several manual, time-consuming, and error-prone translations into the different, problem-specific representations required by each solution method. During seminar presentations, modelers had to defend the existing versions of their models, sometimes quite irrationally, because the time and money needed to make proposed changes were prohibitive. Their models just could not be moved to other environments,

because special programming knowledge was needed, and data formats and solution methods were not portable.

The idea of an algebraic approach to represent, manipulate, and solve largescale mathematical models fused old and new paradigms into a consistent and computationally tractable system. Using matrix generators (see appendix GAMS versus Fortran Matrix Generators) for linear programs taught us the importance of naming rows and columns in a consistent manner. The connection to the emerging relational data model became evident. Painful experience using traditional programming languages to manage those name spaces naturally lead one to think in terms of sets and tuples, and this led to the relational data model. Combining multidimensional algebraic notation with the relational data model was the obvious answer. Compiler writing techniques were by now widespread, and languages like GAMS could be implemented relatively quickly. However, translating this rigorous mathematical representation into the algorithm specific format required the computation of partial derivatives on very large systems. In the 1970s, TRW developed a system called PROSE that took the ideas of chemical engineers to compute point derivatives that were exact derivatives at a given point, and to embed them in a consistent, Fortran-style calculus modeling language. The resulting system allowed the user to use automatically generated exact first and second order derivatives. This was a pioneering system and an important demonstration of a concept. However, in our opinion PROSE had a number of shortcomings: it could not handle large systems, problem representation was tied to an array-type data structure that required address calculations, and the system did not provide access to state-of-the-art solution methods. From linear programming, we learned that exploitation of sparsity was the key to solve large problems. Thus, the final piece of the puzzle was the use of sparse data structures.

With all pieces in place, all we had to do was adopt the techniques to fit into one consistent framework and make it work for large problems

4.21.4 Design Goals and Changing Focus

The original and still valid goal is to improve the model builder's productivity, reduce costs, and improve reliability and overall credibility of the modeling process. To achieve this, we established the following key principles to guide the GAMS development:

- The problem representation is independent of the solution method.
- The data representation follows the relational data model.
- The problem and data representations are independent of computing platforms.
- The problem and data representations are independent of user interfaces.
- Optimization methods will fail, and systems have to be designed to be fail-safe.

Another way to express these principles is to think in terms of layers of representations and capabilities that have clearly defined interfaces and functions. The oldest and most basic layer is the solver layer or implementation of a specific algorithm. Above the solver is the model layer, expressed in an algebraic modeling language. The modeling layer translates the mathematical representation into a computational structure required by a specific solution method and provides various services such as function and derivative evaluations and error recovery. Above the modeling layer is the application or domain layer, which is highly context sensitive and has knowledge about the problem to be solved and the kind of user interacting with the system.

The representation of the model in GAMS is in a form that can be easily read by humans and by machines. This means that the GAMS program itself is the documentation of the model, and that the separate description required in the past (which was a burden to maintain, and which was seldom up-to-date) is no longer needed. Moreover, the design of GAMS incorporates the following features that specifically address the user's documentation needs:

- A GAMS model representation is concise, and makes full use of the elegance of the mathematical representation.
- All data transformations are specified concisely and algebraically. This means that all data can be entered in their most elemental form and that all transformations made in constructing the model and in reporting are available for inspection.
- Explanatory text can be made part of the definition of all symbols and is reproduced whenever associated values are displayed.
- All information needed to understand the model is in one document.

Of course some discipline is needed to take full advantage of these design features, but the aim is to make models more accessible, more understandable, more verifiable, and hence more credible.

It is instructive to put the development of modeling systems into some historic perspective and see how the focus and technical constraints have changed in the last 30 years. We can observe three major phases that shift the emphasis from computational issues to modeling issues and finally the application or the real problems. Each phase defined one of the main system layers discussed above. The dominant constraints in the first phase were the computational limits of our algorithms. Problem representation had to abide by algorithmic convenience, centralized expert groups managed large, expensive and long lasting projects and end users were effectively left out. The second phase has the model in focus. Applications are limited by modeling skill, project groups are much smaller and decentralized, the computational cost are low and the users are involved in the design of the application. Applications are designed to be independent of computing platforms and frequently operate in a client-server environment.

We believe that we are entering a third phase which has the application as its focus and the optimization model is just one of many analytic tools that help making better decisions. The users are often completely unaware of any optimization model or use a mental model that is different from the actual model to solved by optimization techniques. User interfaces are build with off-the-shelf components and frequently change to adjust to evolving environments and new computing technologies. As with databases, modeling components have a much longer life than user interfaces. We have observed cases where the model has remained basically unchanged over many years, whereas the computing environments and user interfaces have changed several times. The solvers used to solve the models have changed, the computing platforms have changed, the user interfaces have changed and the overall performance of the model has changed without any change in the model representation.

4.22 GAMS Programs

4.22.1 Introduction

This chapter provides a look at the structure of the GAMS language and its components. It should be emphasized again that GAMS is a programming language, and that programs must be written in the language to use it. A GAMS program is contained in a disk file, which is usually constructed with a text editor of choice (e.g. GAMS Studio). When GAMS is 'run', the file containing the program (the input file) is submitted to be processed. After this processing has finished, the results, which are in the output file(s), can be inspected. By default the GAMS log appears on the screen while GAMS runs, keeping the user informed about progress and error detection. It is the responsibility of the user to inspect the output file(s) carefully to see the results and to diagnose any errors.

4.22.2 The Structure of GAMS Programs

GAMS programs consist of one or more statements (sentences) that define data structures, initial values, data modifications, and symbolic relationships (equations). While there is no fixed order in which statements have to be arranged, the order in which data modifications are carried out is important. Symbols must be declared as to type before they are used, and must have values assigned before they can be referenced in assignment statements. Each statement is followed by a semicolon except the last statement, where a semicolon is optional.

Note

The semicolon at the end of a statement can be omitted if a new GAMS keyword follows. However, to improve readability of the code, it is recommended to use the semicolon at the end of a statement anyway.

4.22.2.1 Format of GAMS Input

GAMS input is free format. A statement may be placed anywhere on a line, multiple statements may appear on a line, or a statement may be continued over any number of lines as follows:

```
statement;
statement;
statement; statement; statement;
the words that you are now reading is an example of a very
long statement which is stretched over two lines;
```

Blanks and end-of-lines may generally be used freely between individual symbols or words. GAMS is not case sensitive. This means that lower and upper case letters may be mixed freely but are treated identically. Up to 80,000 characters may be placed on a line and completely blank lines may be inserted for easier reading.

Not all lines are a part of the GAMS language. Two special symbols, the asterisk `'*'` and the dollar symbol `'$'` may be used in the first position on a line to indicate a non-language input line. An asterisk in column one means that the line will not be processed, but treated as a comment. For more on comments, see section Comments. A dollar symbol in the first position indicates that compiler options or directives are contained in the rest of the line (see chapter Dollar Control Options for more information).

For example, multiple files may be used as input through the use of the $include facility. In short, the statement

```
$include file1
```

inserts the contents of the specified file (`file1` in this case) at the location of the call. A more complex versions of this is the option $batinclude. Both options are introduced and discussed in details in chapter Dollar Control Options.

4.22.2.2 Classification of GAMS Statements

Each statement in GAMS is classified into one of two groups:

1. Declaration and definition statements.
2. Execution statements.

A declaration statement describes the class of a symbol. Often initial values are provided in a declaration, then it may be called a definition. The specification of symbolic relationships for an equation is a definition. The declaration and definition statements are:

- acronym
- alias
- equation declaration
- equation definition
- file
- function
- model
- parameter
- scalar
- set
- table
- variable

Execution statements are instructions to carry out actions such as data transformation, model solution, and report generation. The execution statements are:

- abort
- assignment
- break
- continue
- display
- execute
- for
- if
- loop
- option
- put
- put_utility

- putclear
- putclose
- puthd
- putpage
- puttl
- repeat
- solve
- while

Note

While an assignment is an execution statement, it also defines the symbol on the left hand side of the assignment.

Although there is great freedom about the order in which statements may be placed in a GAMS program, certain arrangements are commonly used. The two most common are discussed next.

4.22.2.3 Organization of GAMS Programs

One common style of organizing GAMS statements places the data first, followed by the model and the solution statements.

Style 1:

Data:

Set declarations and definitions

Parameter declarations and definitions

Assignments

Displays

Model:

Variable declarations

Equation declarations

Equation definitions

Model definition(s)

Solution:

Solve(s)

Displays

In this style of organization, the sets are placed first. Then the data is specified with parameter, scalar, and table statements. Next, the model is defined with the variable declarations, equation declarations, equation definitions and one or more model statements. Finally, the model is / models are solved and the results are displayed. One can refer to the model **trnsport** as an example for this style.

A second style emphasizes the model by placing it before the data. This is a particularly useful order when the model is to be solved repeatedly with different data sets.

Style 2:

Model:

Set declarations

Parameter declarations

Variable declarations

Equation declaration

Equation definition

Model definition

Data:

Set definitions

Parameter definitions

Assignments

Displays

Solution:

Solve

Displays

Here, there is a separation between declaration and definition. For example, sets and parameters may be declared first with the following statements:

```
Set       c         "crops" ;
Parameter yield(c)  "crop yield" ;
```

Later they may be defined with the statements:

```
Set        c        / wheat, clover, beans / ;
Parameter  yield(c) / wheat     1.5
                      clover    6.5
                      beans     1.0  / ;
```

The first statement declares that the identifier `c` is a set and the later statement defines the elements in this set. Similarly, in the second statement `yield` is declared to be a parameter and later the corresponding data is given.

Note

Sets and parameters that are used in equations must be *declared* before the equations are specified. However, they may be *defined* after the equation specifications but before the specific equation is used in a solve statement. This gives GAMS programs substantial organizational flexibility.

4.22.3 Data Types and Definitions

Each symbol or identifier has exactly one of the following basic GAMS data types:

- acronyms
- equations
- files
- functions
- models
- parameters
- sets
- variables

Note

- `Scalars` and `tables` are not separate data types but convenient input formats for the data type `parameter`. For details see the overview Parameters, Scalars and Tables.
- GAMS uses shorthand symbols for each data type in the output. For details see the overview GAMS Data Types and their Shorthand Symbols.

Declarations have common characteristics. The following example has a typical structure:

```
Parameter    a(i,j)        "input-output matrix" ;
```

The structure is:

```
Keyword for data type - identifier (with index list) - explanatory text ;.
```

Note that the index list (or domain list) and the explanatory text are always optional characteristics. However, we recommend to specify the index list if the data type is defined over a domain; the advantages of this practice are outlined in section Domain Checking.

Note

Variables, sets, parameters and equations may be declared and defined over one or more indices or dimensions. Currently the maximum number of dimensions for all these data types is 20.

It is also recommend to add an explanatory text for reasons of clarity. For more on explanatory texts, see section Text below. Other examples for declarations follow:

```
Set         time     "time periods" ;
Model       turkey   "turkish fertilizer model" ;
Variables   x,y,z ;
```

Observe that in the last example a number of identifiers (separated by commas) is declared in one statement.

4.22.4 Language Items

Before proceeding with more language details, a few basic symbols need to be defined and the rules for recognizing and writing them in GAMS established. These basic symbols are often called lexical elements and form the building blocks of the language. They are:

- characters
- comments
- delimiters
- identifiers (idents)
- labels
- numbers
- reserved words and tokens
- text

Each of these items is discussed in detail in the following subsections.

Attention

As noted previously, GAMS is not case sensitive, so we may use any mix of lower and upper case.

4.22.4.1 Characters

A few characters are not allowed in a GAMS program, because they are illegal or ambiguous on some machines. Generally, all unprintable and control characters are illegal. The only place where any character is legal, is in an `$ontext-$offtext` block as illustrated in section Block Comments below. For completeness, the full set of legal characters are listed in Table 1. Note that most of the uncommon punctuation characters are not part of the language, but they may be used freely in the context of explanatory texts, comments, and labels (if quoted). Similarly, special language specific characters (e.g. ä, ß, à, é, ç, ț, ș) may also be used freely in explanatory texts, comments, and labels (if quoted).

Table 1: Legal Characters

Legal Characters	Description
A to Z	alphabet
a to z	alphabet
0 to 9	numerals
+	plus
-	minus
=	equals
<	less than
>	greater than
( )	parenthesis
[ ]	square brackets
{ }	braces
'	single quote
"	double quote

Legal Characters	Description
\	back slash
/	slash
,	comma
:	colon
;	semicolon
.	dot
?	question mark
!	exclamation mark
	space
_	underscore
&	ampersand
^	circumflex
#	pound sign
*	asterisk
%	percent
@	at
$	dollar

Note

Not every character listed above is allowed to be used in every place (for example, identifiers have certain limitations).

4.22.4.2 Reserved Words

GAMS, like other programming languages such as C and Java, uses reserved words (often also called keywords) that have predefined meanings. Users are in general not permitted to use these for their own definitions, neither as identifiers nor labels. The complete list of reserved words is given below. In addition, a small number of symbols constructed from non-alphanumeric characters have a meaning in GAMS.

Note

While it is not allowed to use reserved words as identifiers and labels in general, it is still possible (put not recommended) in certain cases which are explained in more detail further below.

- abort
- acronym[s]
- alias
- all
- and
- binary
- break
- card
- continue
- diag

- display
- do*
- else
- elseif
- endfor*
- endif*
- endloop*
- endwhile*
- eps
- equation[s]
- execute
- execute_load
- execute_loaddc
- execute_loadhandle
- execute_loadpoint
- execute_unload
- execute_unloaddi
- execute_unloadidx
- file[s]
- for
- free
- function[s]
- gdxLoad
- if
- inf
- integer
- logic
- loop
- model[s]
- na
- negative
- nonnegative
- no
- not
- option[s]
- or

- ord
- parameter[s]
- positive
- procedure[s] (deprecated)
- prod
- put
- put_utility/put_utilities
- putclear
- putclose
- putfmcl
- puthd
- putheader
- putpage
- puttitle
- puttl
- repeat
- sameas
- sand
- scalar[s]
- semicont
- semiint
- set[s]
- singleton
- smax
- smin
- solve
- sor
- sos1
- sos2
- sum
- system
- table[s]
- then∗
- undf
- until
- variable[s]

- while
- xor
- yes

Attention

Some of the keywords above can actually be used as an identifier (e.g. `sameas`). But if they get used as identifier, their built-in meaning as part of the GAMS language can not be accessed anymore.

Note

The words marked with $*$ in the list above are no reserved words by default. However, they get a special meaning if the dollar control option $onEnd is set.

The following list shows words which have a special meaning in GAMS (e.g. they are part of the solve statement), but can be used as identifiers anyway:

- eq
- eqv
- ge
- gt
- imp
- le
- lt
- maximizing
- minimizing
- ne
- scenario
- using

The reserved non-alphanumeric symbols are:

- ..
- =l=
- =g=
- =e=
- =n=
- =x=
- =c=
- =b=
- --
- ++
- **
- ->
- <=>

4.22.4.3 Identifiers

Identifiers are the names given to sets, parameters, variables, models, etc. GAMS requires an identifier to start with a letter followed by more letters or digits. The length of an identifier is currently limited to 63 characters. Identifiers may only contain alphanumeric characters (letters or numbers) or underscores (_). Examples of legal identifiers are:

```
a   a15   revenue   x0051
```

Note that the following identifiers are incorrect:

```
15   $casg      milk&meat
```

Attention

A name used for one data type cannot be reused for another.

4.22.4.4 Labels

Labels are set elements. They may be up to 63 characters long and may be used in quoted or unquoted form.

The unquoted form is simpler to use but places restrictions on characters allowed, in that any unquoted label must start with a letter or digit and can only be followed by letters, digits, underscores (_) or the sign characters + and -. Examples of valid unquoted labels are:

```
Phos-Acid      1986      1952-53   A
September      H2S04      Line-1
```

Quotes can be used to delimit labels. Quoted labels may begin with and include *any* legal character. Either single or double quotes may be used but the closing quote has to match the opening quote. A label quoted with double quotes may contain a single quote (and vice versa). Most experienced users avoid quoted labels because they can be tedious to enter and confusing to read. There are a couple of special circumstances though. If we want to make a label stand out, we could put asterisks in it and indent it. A more subtle example is that GAMS keywords may be used as labels if they are quoted. So labels like `parameter`, `put` or `while` may be used if they are quoted. Some examples of quoted labels follow:

```
'  *TOTAL*'   "MATCH"   '10%-INCR'   '12" / FOOT'   "line 1"
```

Note

- Labels do not have any numerical value. The label '1986' does not have the numerical value 1986 and the label '01' is different from the label '1'. One can access the numerical value of a label with the Set Attributes `.val` attribute.
- Leading blanks in a label are significant and preserved while trailing blanks are trimmed. So `' label1'` is different from `'label1'`, but `'label2 '` is identical to `'label2'`.
- Labels are, like the rest of the GAMS language, case insensitive. Note, however, that case insensitivity for labels applies only to ASCII characters. Labels with non-ASCII characters are case sensitive:

```
* This defines two different labels
Set i / "Ä", "ä" /;
display i;

* This gives a compilation error
Set j / "A", "a" /;
```

To summarize, set names are identifiers and set elements are labels. An overview of the rules for constructing identifiers and labels is given in the following table.

	Identifiers	Unquoted Labels	Quoted Labels
Number of characters	63	63	63
Must begin with	A letter	A letter or a number	Any character
Permitted special characters	Underscore (_)	Underscore (_) and the characters + and –	Any but the starting quote

Table 2: Rules for constructing identifiers and labels

4.22.4.5 Text

Identifiers and set elements may also be associated with a line of descriptive text. This text is more than a comment: it is retained by GAMS and is displayed whenever results are written for the identifier.

Text may be quoted or unquoted. Quoted text may contain any character except the quote character used. Single or double quotes may be used but they must match. Text has to fit on one line and cannot exceed 255 characters in length. Text used in unquoted form must follow a number of mild restrictions. Unquoted text cannot start with a reserved word, `'..'` or `'='` and must not include semicolons `';'`, commas `','`, or slashes `'/'`. End of lines terminate a text. These restrictions are a direct consequence of the GAMS syntax and are usually followed naturally by the user. Some examples are:

```
this is text
final product shipment (tpy)
"quoted text containing otherwise illegal characters ; /,"
'use single quotes to put a "double" quote in text'
```

4.22.4.6 Numbers

Numeric values are entered in a style similar to that used in other computer languages

Attention

- Blanks cannot be used in a number: GAMS treats a blank as a separator.
- The common distinction between real and integer data types does not exist in GAMS. If a number is used without a decimal point it is still stored as a real number.

In addition, GAMS uses an extended range arithmetic that contains special symbols for infinity (`INF`), negative infinity (`-INF`), undefined (`UNDF`), epsilon (`EPS`), and not available (`NA`). The user cannot enter `UNDF`; it is only produced by an operation that does not have a proper result, such as division by zero. All the other special symbols may be entered and used as if they were ordinary numbers. For more details, see section Extended Range Arithmetic.

The following example shows various legal ways of entering numbers:

```
0       156.70    -135       .095      1.
2e10    2e+10     15.e+10    .314e5    +1.7
0.0     .0        0.         INF       -INF
EPS     NA
```

The letter `e` denotes the well-known scientific notation allowing convenient representation of very large or small numbers. For example,

```
1e-5 = 1 * 10^{-5} = 0.00001;
3.56e6 = 3.56 * 10^6 = 3,560,000;
```

Note

- GAMS uses a smaller range of numbers than many computers are able to handle. This has been done to ensure that GAMS programs will behave in the same way on a wide variety of machines, including personal computers. GAMS will create an error if a number with an absolute value greater or equal to `1.0e+300` is used.
- A number may be entered with up to 16 significant digits. The dollar control option $offDigit can be used to control the behavior if this number is exceeded.

4.22.4.7 Delimiters

As mentioned before, statements are separated by a semicolon `';'`. However, if the next statement begins with a reserved word (often called keyword in succeeding chapters), then GAMS does not require that the semicolon is used.

The characters comma `','` and slash `'/'` are used as delimiters in data lists, to be introduced later. The comma terminates a data element (as does an end-of-line) and the slash terminates a data list.

4.22.4.8 Comments

A comment is an explanatory text that is not processed or retained by the computer. There are several ways to include comments in a GAMS program.

Blank Lines

The user may freely enter blank lines to set off certain sections and enhance readability. For example, in **trnsport** there are blank lines between the different parameters:

```
Sets
     i   "canning plants"   / seattle, san-diego /
     j   "markets"          / new-york, chicago, topeka / ;

Parameters

     a(i)  "capacity of plant i in cases"
       /    seattle     350
            san-diego   600  /

     b(j)  "demand at market j in cases"
       /    new-york    325
            chicago     300
            topeka      275  / ;
```

Single Line Comments

Users may insert a single line comment on any line by placing an asterisk * in column 1. The text that follows the asterisk is the comment and may contain GAMS reserved words, messages or any other content. It is completely ignored by the GAMS compiler. Note that several successive lines may be single line comments as in the example below.

The default asterisk * may be replaced by other symbols. GAMS provides the dollar control option $comment to customize the comment character (for more on dollar control options, see chapter Dollar Control Options). The new comment character cannot be used in column 1 as before, since now it has a special meaning. The change of comment character should be used with great care. An example can be seen here:

```
*normal comment
*next line is a deactivated GAMS statement
*  x=sum(I,z(i));
$comment !
!comment with new character
$comment *
*now we are back to how it should be
```

In the fourth line, the new comment character ! replaces the GAMS default * as the comment delimiter. Note that single line comments appear in the echo print of the GAMS output as numbered lines. For details see section The Echo Print of the Input File.

Block Comments

For longer comments special 'block' delimiters may be used that cause GAMS to ignore an entire section of the program. The dollar control option $ontext marks the beginning of the comment block and the option $offtext marks the end. Note that the $ symbol must be in the first character position. The example below illustrates the use of the block comment and also contains some useful information. For more on dollar control options, see chapter Dollar Control Options.

```
$ontext
Following a $ontext directive in column 1 all lines are
ignored by GAMS but printed on the output file until the matching $offtext is encountered, also
in column 1. This facility is often used to logically remove parts of programs
that are not used every time, such as statements producing voluminous reports.
Every $ontext must have a matching $offtext in the same file
$offtext
```

Note that block comments appear in the echo print without line numbers. For details see section The Echo Print of the Input File.

End-of-Line Comments

Comments may also be placed at the end of a line that contains GAMS code. The dollar control option $onEolCom activates end-of-line comments. The default symbol to indicate that the comment begins is a double exclamation mark !!. This symbol may be reset with the option $eolCom followed by the desired symbol which may be one character or a two-character sequence. The following example serves as illustration. For more on dollar control options, see chapter Dollar Control Options.

```
Scalar x /0/;
$onEolCom
x=x+1;        !! eol comment
x = x         !! eol comment in line of GAMS statement, where the GAMS statement continues  to the nex
   +1;
$eolCom &&
x=x+1;        && eol comment with new symbol
```

Note that the option to add end-of-line comments may be deactivated with the dollar control option $offEolCom. End-of-line comments appear in the echo print on the appropriate lines. For details see section The Echo Print of the Input File.

In-Line Comments

Comments may also appear *in* a line of GAMS code. The dollar control option $onInline activates in-line comments. By default, the in-line comment symbols are set to the two character pairs ' /*' and ' */', where ' /*' indicates that the in-line comment begins and ' */' indicates that the in-line comment ends. The comment may span lines till the end-of-comment characters are encountered. The in-line comment symbols may be reset with the option $inLineCom followed by the desired pair of characters. The following example serves as illustration. For more on dollar control options, see chapter Dollar Control Options.

```
Scalar x /0/;
$onInLine
x=x      /* in-line comment*/ +1;
x=x      /* in-line comment in line
            that continues to next line */
   +1;
$inLineCom /&  &/
x=x      /& in-line comment with new character &/ +1;
```

Note that the option to add in-line comments may be deactivated with the dollar control option $offInline. In-line comments appear in the echo print on the appropriate lines. For details see section The Echo Print of the Input File. Note that in-line comments may be allowed to be nested using the dollar control option $onNestCom.

Outside Margin Comments

GAMS provides the facility to define margins. The active code is within the margins, everything outside the set margins is ignored by the compiler and treated as comment. The dollar control option $onMargin activates margin marking and $offMargin deactivates it. The option $minCol is used to specify the first column, where GAMS code that is to be compiled may appear. Similarly, the option $maxCol is used to specify the last column for GAMS code. The following example shows how this works. For more on dollar control options, see chapter Dollar Control Options.

```
$ontext
         1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345678901234567890
$offtext

$onMargin minCol 20 maxCol 45
Now I have          Set i plant /US, UK/     This defines i
turned on the       Scalar x / 3.145 /       A scalar example.
margin marking.     Parameter a, b;          Define some
                                             parameters.
$offMargin
```

The text before column 20 and after column 45 is treated as a comment. Note that the full content of the lines is copied to the echo print, including everything outside the margins. For details see section The Echo Print of the Input File.

Hidden Comments

Finally, GAMS also allows hidden comments, that are not copied to the echo print of the GAMS output file. These comments contain information that is only relevant for the person manipulating the file. They are single line comments starting with the dollar control option $hidden. An example follows. For more on dollar control options, see chapter Dollar Control Options.

```
$hidden a comment I do not want in LST file
set a /a1,a2/;
set b /a2,c2/;
set c /a3,d3/;
```

4.22.5 Summary

This completes the discussion of the components of the GAMS language. Many unfamiliar terms used in this chapter are further explained in the Glossary.

4.23 Set Definition

4.23.1 Introduction

Sets are fundamental building blocks in any GAMS model. They allow the model to be succinctly stated and easily read. In this chapter we will introduce how sets are declared and initialized. More advanced set concepts, such as assignments to sets, and lag and lead operations are covered in the chapters Dynamic Sets and Sets as Sequences. The topics discussed in this chapter will be enough to provide a good start on most models. We will introduce simple sets, subsets, multi-dimensional sets, singleton sets and the universal set. The chapter will be concluded by a topic on domain checking, a very important feature of GAMS, and a section about Domain Defining Symbol Declarations.

4.23.2 Simple Sets

Using common mathematical notation, a set S that contains the elements a, b and c is written as:

$$S = \{a, b, c\}$$

Using GAMS notation, the same set is defined in the following way:

```
Set S / a, b, c /;
```

The set statement begins with the keyword `set`, `S` is the name of the set, and its members are `a`, `b`, and `c`. They are labels, but are often referred to as elements or members.

4.23.2.1 Defining a Simple Set: The Syntax

In general, the syntax for *simple* sets in GAMS is as follows:

```
set[s] set_name ["text"] [/element [text] {,element [text]} /]
     {,set_name ["text"] [/element [text] {,element [text]} /] } ;
```

`Set[s]` is the keyword that indicates that this is a set statement. `Set_name` is the internal name of the set in GAMS, it is an identifier. The optional explanatory text may be used to describe the set or a specific set element for future reference and to ease readability. The list of set elements is delimited by forward slashes. `Element` is the name of the set element(s). Note that each element in a set must be separated from other elements by a comma or by an end-of-line, and each element is separated from any associated text by a blank.

Consider the following example from the Egyptian fertilizer model **[FERTS]**, where the set of fertilizer nutrients could be written as

```
Set cq "nutrients" / N,  P2O5 /;
```

or as

```
Set cq "nutrients" / N
                     P2O5  /;
```

The order in which the set members are listed is usually not important. However, if the members represent, for example, time periods, then it may be useful to refer to the *next* or *previous* member. There are special operations to do this, and they are discussed in chapter Sets as Sequences: Ordered Sets. For now, it is enough to remember that the order in which set elements are specified is not relevant, unless and until some operation implying order is used. At that time, the rules change, and the set becomes what we will later call an *ordered set*.

Note

- The data statement, i.e. specification of set elements in forward slashes can be omitted. In such cases a set is declared without being defined.
- More than one set may be declared and defined in one set statement. Examples are given in subsection Declaring Multiple Sets below.

4.23.2.2 Illustrative Examples

Consider the following example based on the model [**SHALE**]:

```
Set cf "final products" / syncrude "refined crude (mil bbls)"
                          lpg      "liquefied petroleum gas (million bbls)"
                          ammonia  "ammonia (mil tons)"
                          coke     "coke (mil tons)"
                          sulfur   "sulfur (mil tons)" /;
```

The set statement is introduced with the keyword `set`, the name of the set is `cf` and the explanatory text `"final products"` describes the set. The set has five elements with explanatory texts that contain details of the units of measurement.

Usually sets are declared and defined once and then referenced in the model. There are two exceptions: the dollar control option onMulti allows adding more elements later, and dynamic sets. For details on dynamic sets, see chapter Dynamic Sets. The following code sightly varies the previous example to demonstrate the option `$onMulti:`

```
Set cf "final products"      / syncrude "refined crude (mil bbls)"
                               lpg      "liquefied petroleum gas (million bbls)"
                               ammonia  "ammonia (mil tons)" /;
$onmulti
Set cf "more final products" / coke     "coke (mil tons)"
                               sulfur   "sulfur (mil tons)" /;
```

After `$onmulti` additional elements are added to the set `cf`. Note that without the dollar control option `$onMulti` that would generate an error as per default a symbol can have at most one data statement.

4.23.2.3 Sequences as Set Elements

The asterisk `'*'` plays a special role in set definitions. It is used to relieve the tedium of typing a sequence of elements for a set, and to make intent clearer. For example, in a simulation model there might be ten annual time periods from 1991 to 2000. Instead of typing ten years, the elements of this set can be written as:

```
Set t "time" / 1991 * 2000 /;
```

This means that the set includes the ten elements `1991`, `1992`, ..., `2000`. GAMS builds up these label lists by looking at the differences between the two labels. If the only characters that differ are digits, with the number `L` formed by these digits in the left and `R` in the right, then a label is constructed for every integer in the sequence `L` to `R`. Any non-numeric differences or other inconsistencies cause errors.

The following example illustrates the most general form of the *'asterisked'* definition:

```
Set g1 / a1bc * a20bc /;
```

Note that this is not the same as:

```
Set g2 / a01bc * a20bc /;
```

Both sets have 20 members, but they have only 11 members in common.

Lists in decreasing order are also possible:

```
Set y "years in decreasing order" / 2000 * 1991 /;
```

As a last example, the following set definitions are both illegal because they are not consistent with the rule given above for making lists:

```
Set illegal1 / a1x1 * a9x9 /
    illegal2 / a1 * b9 /;
```

4.23.2.4 Declaring Multiple Sets

The keyword set does not need to be used for each set, rather only at the beginning of a group of sets. It is often convenient to put a group of set declarations (and definitions) together at the beginning of the program. When this is done the set keyword needs only be used once. Those who prefer to intermingle set declarations with other statements, have to use a new set statement for each additional group of sets. Note that the keywords `set` and `sets` are equivalent. The following example below shows how two sets can be declared together. Note that the semicolon is used only after the last set is declared.

```
Sets s "Sectors" / manuf, agri, services, government /
     r "Regions" / north, eastcoast, midwest, sunbelt / ;
```

4.23.2.5 Using Previously Defined Sets in Set Definitions

The following notation allows previously defined sets to be used in a new set definition:

```
Set i / i1 * i4 /
    j / j6 * j9 /
    k / #i, set.j /;
```

The set `k` contains all elements of the sets `i` and `j`. Note that the hash sign '#' followed by a set name is a shorthand for referring to all the elements in a set. The notation `set.set_name` works identically and is just a different way to refer to all elements in a previously defined set.

4.23.3 The Alias Statement: Multiple Names for a Set

Sometimes it is necessary to have more than one name for the same set. In input-output models for example, each commodity may be used in the production of all other commodities and it is necessary to have two names for the set of commodities to specify the problem without ambiguity. In the general equilibrium model [**ORANI**], the set of commodities `c` is written as

```
Set c "Commodities" / food, clothing /;
```

A second name for the set `c` is established with either of the following statements:

```
Alias (c, cp) ;
Alias (cp, c) ;
```

Here `cp` is the new set name that can be used instead of the original set name `c`.

Note

The newly introduced set name may be used as an alternative name for the original set; the associated set will always contain the same elements as the original set.

With the `alias` statement more than one new name may be introduced for the original set:

```
Alias (c,cp, cpp, cppp);
```

Here `cp`, `cpp`, `cppp` are all new names for the original set `c`.

Note

The order of the set names in the alias statement does not matter. The only restriction is that exactly one of the sets in the statement must be defined earlier. All the other sets are introduced by the alias statement.

Typical examples for the usage of aliases are problems where transportation costs between members of one set have to be modeled. The following code snippet is adapted from the Andean fertilizer model [**ANDEAN**]:

```
Set i "plant locations" / palmasola, pto-suarez, potosi, baranquill, cartagena /;
Alias(i,ip);

Table tran(i,i) "transport cost for interplant shipments (us$ per ton)"
              palmasola    pto-suarez     potosi  baranquill
 pto-suarez       87.22
 potosi           31.25         55.97
 baranquill       89.80        114.56      70.68
 cartagena        89.80        114.56      70.68        5.00
;
Parameter mui(i,ip) "transport cost: interplant shipments (us$ per ton)";
mui(i,ip) = (tran(i,ip) + tran(ip,i));
```

The alias statement introduces `ip` as another name for the set `i`. The table `tran` is two-dimensional and both indices are the set `i`. The data for the transport cost between the plants is given in this table; note that the transport costs are given only for one direction here, i.e. the costs from `pto-suarez` to `palmasola` are explicitly specified in the table while the costs in the opposite direction are not given at all. The parameter `mui` is also two-dimensional and both indices refer to the set `i`, but this time the alias `ip` is used in the second position. The parameter `mui` is defined with the assignment statement in the next line: `mui` contains the transport costs from one plant location to the other, in both directions. Note that if `mui` were defined without the alias, then all its entries would have been zero. For other examples where aliases are used, see sections The Universal Set and Finding Sets from Data below.

4.23.4 Subsets

It is often necessary to define sets whose members must all be members of some larger set. The syntax is:

```
set set_ident1(set_ident2) ;
```

Here `set` is the keyword indicating that this is a set statement, and `set_ident1` is a subset of the larger set `set_ident2`. The larger set is also called superset.

For instance, we may wish to define the sectors in an economic model following the style in [**CHENERY**].

```
Set i    "all sectors"        / light-ind, food+agr, heavy-ind, services /
    t(i) "traded sectors"     / light-ind, food+agr, heavy-ind /
    nt   "non-traded sectors" / services /;
```

Some types of economic activity, for example exporting and importing, may be logically restricted to a subset of all sectors. In order to model the trade balance we need to know which sectors are traded, and one obvious way is to list them explicitly, as in the definition of the set `t` above. The specification `t(i)` means that each member of the set `t` must also be a member of the set `i`. GAMS will enforce this relationship, which is called domain checking. Obviously, the order of declaration and definition is important: the membership of `i` must be known before `t` is defined, otherwise checking cannot be done.

Note

All elements of the subset must also be elements of the superset.

It is legal but unwise to define a subset without reference to the larger set, as is done above for the set `nt`. In this case domain checking cannot be performed: if `services` were misspelled no error would be marked, but the model may give incorrect results. Hence, it is recommended to use domain checking whenever possible. It catches errors and allows to write models that are conceptually cleaner because logical relationships are made explicit.

An alternative way to define elements of a subset is with assignments:

```
Set i    "all sectors"    / light-ind, food+agr, heavy-ind, services /
    t(i) "traded sectors" / light-ind,  heavy-ind /;
t('food+agr') = yes;
```

In the last line the element `food+agr` of the set `i` is assigned to the subset `t`. Assignments may also be used to *remove* an element from a subset:

```
t('light-ind') = no;
```

Note that `yes` and `no` are reserved words in GAMS. Note further that if a subset is assigned to, it then becomes a dynamic set. For more on assignments in GAMS in general, see section The Assignment Statement.

Attention

A subset can be used as a domain in the declaration of other sets, variables, parameters and in equations as long as it is no dynamic set.

This completes the discussion of sets in which the elements are simple. This is sufficient for many GAMS applications. However, there are a variety of problems for which it is useful to have sets that are defined in terms of two or more other sets.

4.23.5 Multi-Dimensional Sets

It is often necessary to provide mappings between elements of different sets. For this purpose, GAMS allows the use of multi-dimensional sets. For the current maximum number of permitted dimensions, see Dimensions. The next two subsections explain how to express one-to-one and many-to-many mappings between sets.

4.23.5.1 One-to-one Mapping

Consider a set whose elements are pairs: $A = \{(b, d),\ (a, c),\ (c, e)\}$. In this set there are three elements and each element consists of a pair of letters. This kind of set is useful in many types of modeling. For example, in the world aluminum model [**ALUM**] a port has to be associated with a nearby mining region

```
Set i       "mining regions"     / china, ghana, ee+ussr, s-leone /
    n       "ports"              / accra, freetown, leningrad, shanghai /
    in(i,n) "mines to ports map" / china  .shanghai
                                   ghana  .accra
                                   ee+ussr.leningrad
                                   s-leone.freetown /;
```

Here `i` is the set of mining regions, `n` is the set of ports and `in` is a two dimensional set that associates each port with a mining region. The dot between `china` and `shanghai` is used to create one such pair. Blanks may be used freely around the dot for readability. The set `in` has four elements, and each element consists of a region-port pair. The notation `(i,n)` after the set name `in` indicates that the first member of each pair must be a member of the set `i` of mining regions, and that the second must be in the set `n` of ports. GAMS will domain check the set elements to ensure that all members belong to the appropriate sets.

4.23.5.2 Many-to-Many Mapping

A many-to-many mapping is needed in certain cases. Consider the following sets:

```
Set  i         / a, b /
     j         / c, d, e /
     ij1(i,j)  / a.c, a.d /
     ij2(i,j)  / a.c, b.c /
     ij3(i,j)  / a.c, b.c, a.d, b.d /;
```

Here the set `ij1` presents a *one-to-many* mapping where one element of the set `i` maps onto many elements of the set `j`. The set `ij2` represents a *many-to-one* mapping where many elements of the set `i` map onto one element of the set `j`. The set `ij3` is the most general case: a *many-to-many* mapping where many elements of the set `i` map to many elements of the set `j`.

These sets may be written compactly as:

```
Set  i        / a, b /
     j        / c, d, e /
     ij1(i,j) / a.(c,d) /
     ij2(i,j) / (a,b).c /
     ij3(I,j) / (a,b).(c,d) /;
```

The parenthesis provides a list of elements that is expanded when creating pairs. Note that the dot '.', if used like above, acts as *product operator* and supports building the Cartesian product of sets.

Attention

> When complex sets like this are created, it is important to check that the desired set has been obtained. The checking can for example be done be done by using a display statement.

GAMS provides more notation to define multi-dimensional sets in a succinct way. As introduced above the hash sign `'#'` followed by a set name is a shorthand for referring to all the elements in a set. The matching operator `':'` may be used to map ordered sets. This operator is similar to the product operator `'.'`. However, in this case elements are matched pairwise by mapping elements with the same order number. The examples below demonstrate these concepts.

```
Set
i         / a, b /
j         / c, d, e /

ij4a(i,j) / a.#j /
ij4b(i,j) / a.c, a.d, a.e /

ij5a(i,j) / #i.#j /
ij5b(i,j) / a.c, a.d, a.e, b.c, b.d, b.e /

ij6a(i,j) / #i:#j /
ij6b(i,j) / a.c, b.d /;
```

Note that set names that differ only by the last letter denote identical sets. For example, set `ij4a` is identical to set `ij4b`. Observe that set `i` has two elements and set `j` has three elements, where `e` is the element with the highest order. Set `ij6a` is an ordered mapping of all elements of set `i` to all elements of set `j`. However, since there is a mismatch in the number of elements, element `e` is not mapped to.

These concepts may be generalized to sets with higher dimensions. Mathematically, these are called 3-tuples, 4-tuples, or more generally, n-tuples. Some examples for the compact representation of sets of n-tuples using combinations of dots, parentheses, and commas are shown in Table 1.

Compact Notation	Result
(a,b).c.d	a.c.d, b.c.d
(a,b).(c,d) .e	a.c.e, b.c.e, a.d.e, b.d.e
(a.1*3).c	(a.1, a.2, a.3).c or a.1.c, a.2.c, a.3.c
1*3. 1*3. 1*3	1.1.1, 1.1.2, 1.1.3, ..., 3.3.3

Table 1: Examples for compact representation of multi-dimensional sets

Note that the asterisk may also be used in conjunction with the dot. Recall that the elements of the list 1*4 are $\{1, 2, 3, 4\}$.

A powerful and very compact way to define multi-dimensional sets is with a special option that takes an identifier as value and carries out identifier operations like index matching using the matching operator ':'. The following example illustrates the method.

```
Set i / i1*i4 /
    j / j1*j5 /
    k / k1,k2 /
    h / h1*h3 /;

Set b(i,j,k), c(i,j,k,h);

Option b(i:j,k), c(b:h);
display b, c;
```

The set b is a three-dimensional set, the option statement specifies which permutations of the elements of i, j, and k are elements of b. The matching operator ':' is between i and j, so we must first match the elements of the sets i and j. That gives us the the first two positions. For the third position we cycle through all elements of the set k. This results in the following elements for the set b:

```
i1.j1.k1, i1.j1,k2, i2.j2.k1, ... , i4.j4.k2
```

The set c is a four-dimensional set. Note that the first three dimensions are identical to the domain of the set b. The option statement specifies that in the first three positions we will have elements of the set b and and these are matched with the elements of the set h which are in the fourth position. Now, the set h has only three elements, so only the first three elements of the set b are matched with the members of the set h. This results in the following set:

```
i1.j1.k1.h1, i1.j1.k2.h2, i2.j2.k1.h3
```

As recommended above, it is important to always check whether the multi-dimensional sets generated with compact statement like these are indeed the sets that were intended.

For more sophisticated examples of how to use the matching operator within an option statement please see section Index Matching .

4.23.5.3 The Table Format for Multi-Dimensional Sets

An alternative way to declare multi-dimensional sets is with tables. We show by example how tables may be used in the context of set definitions:

```
Set origins      / Berlin, Paris /
    destinations / London, Chicago, Budapest /
    linked_1(origins,destinations) "cities linked by railways"
                 / Berlin.London, Berlin.Budapest,
                   Paris .London, Paris .Budapest /;

Set Table linked_2(origins,destinations)
           London       Chicago     Budapest
Berlin       yes          no          yes
Paris        yes                      yes  ;
```

The set `linked_1` is a two-dimensional set that is defined with the dot notation introduced above. The set `linked_2` is the same set defined using the table notation: the keyword `set` is followed by the keyword `table` and the name of the set with its domain. The table itself consists of the elements of the first index in the first column, the elements of the second index in the first row, and the data in the grid positions. Note that the keyword `yes` indicates that a label combination is part of the two-dimensional set and the keyword `no` or a blank indicates that the label combination is not contained in the new set. Please see section Tables for detailed requirements for inputting data in the table format.

Alternatively, the multi-dimensional set may be declared first without any elements, and the elements are added later in a separate table statement:

```
Set   origins      / Berlin, Paris /
      destinations / London, Chicago, Budapest /
      linked_2(origins,destinations) "cities linked by railways";
Table linked_2(origins,destinations)
           London       Chicago     Budapest
Berlin       yes          no          yes
Paris        yes                      yes;
```

Instead of the keywords `yes` and `no` users may also use numbers to specify membership in the two-dimensional set: nonzero numeric entries mean that a label combination is part of the set and zero or a blank indicates that the label combination is not contained in the set.

4.23.5.4 Projection and Aggregation of Sets

In GAMS, projection and aggregation operations on sets can be performed in two different ways: with an option statement and with an assignment. For a detailed discussion, see section Projection and Aggregation of Sets and Parameters.

4.23.6 Singleton Sets

A *singleton set* in GAMS is a special set that has at most one element (zero elements are allowed as well). Like other sets, singleton sets may have a domain with several dimensions. For the current maximum number of permitted dimensions, see Dimensions. Singleton sets are declared and defined with the keyword `singleton` that acts as a modifier to the keyword `set:`

```
Set            i      / a, b, c /;
Singleton Set  j      / d       /
               k(i)   / b       /
               l(i,i) / b.c     /;
```

The sets `j`, `k` and `l` are declared as singleton sets, each of them has just one element. The set `k` is a subset of the set `i` and the set `l` is a two-dimensional set.

Note that a data statement for a singleton set with more than one element will create a compilation error:

```
   1  Singleton Set s / s1*s3 /;
****                           $844
   2  display s;

Error Messages
844  Singleton with more than one entry (see $onStrictSingleton)
```

It also possible to assign an element to a singleton set. In this case the singleton set is automatically cleared of the previous element first. For example, adding the following line to the code above will result in set `k` containing only element `a` after execution:

```
k('a') = yes;
```

The dollar control option offStrictSingleton may be used to allow sets that are declared as singleton sets to have more than one element in compile time definitions. However, in this case only the first listed element is a valid element of the set. Note that the value of zero for the command line parameter strictSingleton has the same effect for execution time definitions of singleton sets via assignment statements.

For more on dollar control options, see chapter Dollar Control Options. For more on GAMS command line parameters,see chapter The GAMS Call and Command Line Parameters. For more on compilation errors, see section Compilation Errors.

Singleton sets can be especially useful in assignment statements since they do not need to be controlled by a controlling index or an indexed operator like other sets. Consider the following examples:

```
Set            i      / a, b, c /;
Singleton Set  k(i)   / b /
               h(i)   / a /;
Parameter      n(i)   / a 2, b 3, c 5 /;
Scalar z1, z2;

z1 = n(k);
z2 = n(k) + 100*n(h);
```

The singleton sets `k` and `h` are both subsets of the set `i`. The parameter `n` is defined over the set `i`. The scalar `z1` is assigned a value of the parameter `n` without naming the respective label explicitly in the assignment. It is already specified in the definition of the singleton set `k`. The assignment statement for the scalar `z2` contains an expression where the singleton sets `k` and `h` are referenced without a controlling index or an indexed operation.

Note

Singleton sets cannot be used as domains.

4.23.7 The Universal Set: * as Set Identifier

GAMS provides the *universal set* denoted by '*' for cases where the user wishes not to specify an index but have only a placeholder for it. The following examples show two ways how the universal set is introduced in a model. We will discuss the advantages and disadvantages of using the universal set later. The first example is from the production and inventory model [**ROBERT**]:

```
Sets   r          "raw materials"  / scrap, new /;
Table  misc(*,r)  "other data"
           scrap  new
max-stock    400  275
storage-c     .5    2
res-value     15   25;
```

A `table` is an input format for the data type `parameter` and has at least two dimensions. For details see section Tables. In our example, the first index is the universal set '*' and the second index is the previously defined set `r`. Since the first index is the universal set any entry whatsoever is allowed in this position. In the second position elements of the set `r` must appear, they are domain checked, as usual.

The second example illustrates how the universal set is introduced in a model with an alias statement.

```
Alias (new_universe,*);
Set k(new_universe)    / Chicago / ;
```

The alias statement links the universal set with the set name `new_universe`. Set `k` is a subset of the universal set and `Chicago` is declared to be an element of `k`. Any item may be added freely to `k`.

The universal set is particularly useful for generating reports, since it allows the use of any labels without having to define special sets for them. For an example, see section Set Attributes below. For more on report writing, see chapter The Put Writing Facility.

Attention

> It is recommended to not use the universal set for data input, since there is no domain checking and thus typos will not be detected and data that the user intends to be in the model might actually not be part of it.

Observe that in GAMS a simple set is always regarded as a subset of the universal set. Thus the set definition

```
Set i / i1*i10 /;
```

is the same as

```
Set i(*)  / i1*i10 /;
```

GAMS follows the concept of a domain tree for domains in GAMS. It is assumed that a set and its subset are connected by an arc where the two sets are nodes. Now consider the following one dimensional subsets:

```
Set i, ii(i), j(i), jj(j), jjj(jj);
```

These subsets are connected with arcs to the set `i` and thus form a *domain tree* that is rooted in the universe node '*'. This particular domain tree may be represented as follows:

```
* - i - ii
      |
      - j - jj - jjj
```

Note that with the construct `i(jjj)` we may access `ii` iterating through the members of `jjj`. For an example, see domain tree in the loop statement.

Observe that the universal set is assumed to be *ordered* and operators for ordered sets such ord, lag and lead may be applied to any sets aliased with the universal set.

4.23.8 Set and Set Element Referencing

Sets or set elements are referenced in many contexts, including assignments, calculations, equation definitions and loops. Usually GAMS statements refer to the whole set or a single set element. In addition, GAMS provides several ways to refer to more than one, but not all elements of a set. In the following subsections we will show by example how this is done. GAMS also has set functions that specifically reference sets and are introduced in the chapter about logical conditions.

4.23.8.1 Referencing the Whole Set

Most commonly whole sets are referenced as in the following examples:

```
Set i / i1*i100 /;
Parameter k(i);
k(i) = 4;
Scalar z;
z = sum(i, k(i));
```

The parameter `k` is declared over the set `i`, in the assignment statement in the next line *all* elements of the set `i` are assigned the value 4. The scalar `z` is defined to be the sum of *all* values of the parameter `k(i)`.

4.23.8.2 Referencing a Single Element

Sometimes it is necessary to refer to specific set elements. This is done by using single or double quotes around the label(s). We may add the following line to the example above:

```
k('i77') = 15;
```

This statement changes the value of `k('i77')` to 15, all the other values of `k` remain 4.

4.23.8.3 Referencing a Part of a Set

There are multiple ways to restrict the domain to more than one element, e.g. subsets, conditionals and tuples. Suppose we want the parameter `k` from the example above to be assigned the value 10 for the first 8 elements of the set `i`. The following two lines of code illustrate how easily this may be accomplished with a subset:

```
Set j(i) / i1*i8 /;
k(j) = 10;
```

First we define the set `j` to be a subset of the set `i` with exactly the elements we are interested in. Then we assign the new value to the elements of this subset. The other values of the parameter `k` remain unchanged. For examples using conditionals and tuples, see sections Restricting the Domain: Conditionals and Restricting the Domain: Tuples respectively.

4.23.9 Set Attributes

A GAMS set element has several numbers attached to it. These values are called *attributes*; they may be recovered during execution. The attributes are listed in Table 3.

Set Attribute	Symbol	Description
Position	.pos	Element position in the current set (set does not have to be ordered), starting with 1.
Ord	.ord	Same as .pos but for ordered sets only.
Offset	.off	Element position in the current set minus 1. So .off = .pos - 1 (set does not have to be ordered).
Reverse	.rev	Reverse element position in the current set, so the value for the last element is 0, the value for the penultimate is 1, etc. (set does not have to be ordered)
Unique Element	.uel	Element position in the unique element list. For details see section Ordered and Unordered Sets
Length	.len	Length of the set element name (a count of the number of characters).
Text Length	.tlen	Length of the set element text (a count of the number of characters).
Value	.val	If a set element is a number, this attribute gives the value of the number. For extended range arithmetic symbols, the symbols are reproduced. If a set element is a string that is not a number, then this attribute is not defined and trying to use it results in an error.
Text Value	.tval	If a set element text is a number, this attribute gives the value of the number. For extended range arithmetic symbols, the symbols are reproduced. If a set element text is a string that is not a number, then this attribute is not defined and trying to use it results in an error.
First set element	.first	Returns 1 for the first set element, otherwise 0.
Last set element	.last	Returns 1 for the last set element, otherwise 0.

Table 3: Set Attributes

The attributes may be accessed with an assignment statement:

```
data(set_name) = set_name.attribute ;
```

Here data is a parameter, set_name is the name of the set and .attribute is one of the attributes listed above. The following example serves as illustration:

```
Set       id             "example set"            / Madison 'Wisconsin', tea-time '5', '-inf', '-7',
Parameter report(id,*)   "set attribute values";

report(id,'position')    = id.pos ;
report(id,'reverse')     = id.rev ;
report(id,'offset')      = id.off ;
report(id,'length')      = id.len ;
report(id,'textLength')  = id.tlen ;
report(id,'first')       = id.first;
report(id,'last')        = id.last ;

display report;
```

The parameter report is declared to have two dimensions with the set id in the first position and the universal set in the second position. In the following seven statements the values of report are defined for seven entries of the universal set. Note how the flexibility of the universal set is used here to make reporting easy. The display statement generates the output that follows.

```
----     11 PARAMETER report  set attribute values

            position     reverse      offset      length  textLength       first        last

Madison        1.000       4.000                   7.000       9.000       1.000
tea-time       2.000       3.000       1.000       8.000       1.000
-inf           3.000       2.000       2.000       4.000
-7             4.000       1.000       3.000       2.000
13.14          5.000                   4.000       5.000                                1.000
```

4.23.10 Finding Sets from Data

Sometimes it is desirable to find a set from the available data in order to use it later in the model. We will show by example how this may be accomplished using the alias statement, the universal set and conditionals. Suppose we have only the data related to the transportation model [**TRNSPORT**] and we want to identify the sets. We can tell from the data that there are two sets that we are interested in. First, we define these two sets as aliases of the universal set, which means that no elements are specified:

```
Alias(sources, places, *);
```

Then we enter the data that contain an indicator of which set elements are valid entries in the sets to be computed. We use the table format.

```
Table trandata (sources,places)    "data from spreadsheet"
                     Newyork     Chicago    totalsupply
      Seattle          2.5          1.7         350
      Sandiego         2.5          1.8         300
      totalneed        325           75                              ;
```

Next we define subsets that we will need in the calculations that follow:

```
Set  source(sources)       "sources in spreadsheet data"
     destination(places)   "destination in spreadsheet data";
```

Now we have everything that we need to do the calculation using the data on hand. In our case, a label qualifies as an element of the set `source` if it has an entry for `totalsupply` in the table above, and a label is an element of the set `destination` if it has an entry for `totalneed` in the table `trandata`:

```
source(sources)$(trandata(sources,"totalsupply"))    =  yes;
destination(places)$(trandata("totalneed", places )) =  yes;
```

These conditional assignments define the elements of the sets `source` and `destination`. From this point on these sets may be used in the model. However, note that the resulting sets are dynamic sets. Hence they cannot be used as domains in *declaration* statements of other sets, parameters, variables and equations. But they may be referenced and used in equation *definitions*.

Such computations may for example be useful if the user gets a data table from elsewhere and needs to specify the sets. Alternatively, if the data is available in gdx format, the dollar control option load provides functionality to project sets from data contained in a GDX file.

See also section Implicit Set Definition (or: Domain Defining Symbol Declarations).

4.23.11 Domain Checking

The GAMS compiler performs a check to ensure that each label quoted as a member of a set is indeed an element of the respective set, and each element defined in a subset is in fact a member of the superset. This screening for consistency is called *domain checking*. It is done whenever a domain is referenced, be it in set, variable, parameter or equation declarations and definitions, or in assignments. The following examples serve as illustration.

```
Set   i     "all cities"      / Lima, Toronto, Wuhan, Shanghai /
      as(i) "Asian cities"    / Wuhan, Shanhai, Calcutta /
      am    "American cities" / Lima, Toront /;
```

The set `as` is declared to be a subset of the set `i`, therefore domain checking will test every label for inconsistencies. It will catch two errors: there is a typo in `Shanhai` and `Calcutta` is not a member of the set `i`, so it cannot legally be a member of a subset.

```
   1  Set   i     "all cities"      / Lima, Toronto, Wuhan, Shanghai /
   2        as(i) "Asian cities"    / Wuhan, Shanhai, Calcutta /
****                                            $170      $170
**** 170  Domain violation for element
   3        am    "American cities" / Lima, Toront /;
```

The user can rectify the spelling error, and either delete `Calcutta` from the subset `as` or add it to the superset `i`. The following line will pass domain checking:

```
Set   as(i) "Asian cities"    / Wuhan, Shanghai /;
```

Note that `am` is not declared as a subset of the set `i` even though it apparently should contain cities contained in `i`. Hence, `am` cannot be domain checked and the typo in `Toront` will go undetected. This has consequences for the next line:

```
Parameter pam(am) "population in millions"  / Lima 8.9, Toronto 5.6 /;
```

In this parameter definition the domain of the parameter `pam` is the set `am`. GAMS will report an error here, since domain checking does not recognize the label `Toronto`. `Toront`, as specified in the definition of the set `am` above would be accepted.

A further example for domain checking concerns multi-dimensional domains where the user accidentally switches the positions of the indices:

```
Parameter  h(as,am) / Wuhan.Lima  10, Wuhan.Toronto 12, Shanghai.Lima 7/;
Parameter  d(as,am);
d(as,am) = 5*h(am,as) + 78;
```

Observe that we assume that the typo in the label `Toronto` has been rectified. The parameter `h` is defined over the domain `(as,am)`. However, in the assignment statement in the last line above, it is referenced with the domain `(am,as)`. This mistake is caught by domain checking and an error is reported.

As we have seen in the definition of the set `am` above, domain checking is not compulsory. If the following statement is entered, GAMS makes no assumptions about `rho` until further information is provided.

```
set       t   years         / 1988 * 1995 /;
Parameter rho discount rate ;
```

The modeler may later choose to domain check `rho` by continuing the definition with the following line:

```
Parameter rho(t)  / 1988 0.07, 1989*1994 0.10, 1995 0.09 /;
```

Alternatively, the modeler may choose not to domain check the paramter `rho`, as is shown in the deliberately nonsensical (but legal) statement that follows:

```
Parameter rho / 1988.January 0.07, strategy-1.cost 44, cat.capacity 99 /;
```

If a parameter is not domain-checked, the only restriction is that the dimensionality must be constant. Once the number of labels per data item has been established it is frozen; to refer to the parameter differently is an error.

Note

Domain checking is automatic; it is only suppressed in two cases:

1. The index is the universal set or a set aliased to the universal set, see the examples above.
2. The dollar control option $onWarning is used. It has the effect that warnings rather than errors are reported for domain violations.

We urge modelers to use domain checking whenever possible. It catches errors and allows users to write models that are conceptually cleaner because logical relationships are made explicit.

Note that the dollar control option $load is available in several variations to enable domain checking when loading data from a GDX file. For details, see $loadDC, $loadDCM and $loadDCR and chapter GAMS Data eXchange (GDX).

4.23.12 Implicit Set Definition (or: Domain Defining Symbol Declarations)

As seen above, sets can be defined through data statements in the declaration. Alternatively, sets can be defined implicitly through data statements of other symbols which use these sets as domains. This is illustrated in the following example, which is derived from the **[TRNSPORT]** model:

```
Set
   i 'canning plants'
   j 'markets';

Table d(i<,j<) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;

Display i,j;
```

Note the < signs in the domain list of the declaration of `d` (`d(i<,j<)`). These signal, that the set `i` will contain all elements which define the first dimension of symbol `d` and that the set `j` will contain all elements which define the second dimension of symbol `d`, respectively. So, this is the output of the `Display` statement at the end:

```
----     10 SET i  canning plants

seattle  ,    san-diego


----     10 SET j  markets

new-york,    chicago ,    topeka
```

This syntax is not limited to the table statement, but can be used with any symbol declaration. Also, one domain set can be defined through multiple symbols using the same domain, when using the dollar control option onMulti:

```
Set
   food
   fruits(food<)    / apple, orange       /
$onMulti
   vegetable(food<) / carrot, cauliflower /
   meat(food<)      / beef, pork          /;

Display food;
```

This is the output of the `Display` statement:

```
----      8 SET food

apple      ,    orange     ,    carrot     ,    cauliflower,    beef       ,    pork
```

Note

If the < sign is used to mark a declaration as "domain defining", this attribute is not limited to the data statement following this declaration, but also influences other ways to define data at compile time like the dollar control option load, as shown in the following example:

```
Set
   i 'canning plants'
   j 'markets';

Parameter d(i<,j<) 'distance in thousands of miles';

$gdxIn data.gdx
$load d
```

Attention

Only non-zero elements in a symbol will add elements to an implicitly defined set. This is illustrated in the following two examples.

```
Set
   i 'canning plants'
   j 'markets';

Table d(i<,j<) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5              1.8
   san-diego       2.5              1.4;

Display i,j;
```

Note the empty column for `chicago`. Since there is no data, `chicago` will not end up in the set `j`, which can be seen in the output of the `Display` statement:

```
----     10 SET j  markets

new-york,    topeka
```

Also, an explicit `0` in a data statement does not add elements to an an implicitly defined set (in contrast to an `eps`). This is shown in the following GAMS code and output:

```
Set
   j 'markets';

Parameter
   b(j<) 'demand at market j in cases'
        / new-york   325
          chicago      0
          topeka     eps /;

Display j;
```

```
----     10 SET j  markets

new-york,    topeka
```

See also section Finding Sets from Data.

4.23.13 Summary

In GAMS, a simple set consists of a set name and the elements of the set. Both the name and the elements may have associated text that explains the name or the elements in more detail. More complex sets have elements that are pairs or even n-tuples. These sets with pairs and n-tuples are ideal for establishing relationships between the elements in different sets. GAMS also uses a domain checking capability to help catch labeling inconsistencies and typographical errors made during the definition of related sets.

The discussion here has been limited to sets whose members are all specified as the set is being declared. For many models this is all the user needs to know about sets. Later we will discuss more complicated concepts, such as sets whose membership changes in different parts of the model (assignment to sets) and other set operations such as unions, complements and intersections.

4.24 Dynamic Sets

4.24.1 Introduction

Sets in general are covered in chapter Set Definition. In this chapter we introduce a special type of sets: *dynamic sets*. The sets that we discuss in detail in chapter Set Definition have their elements stated at compile time (e.g. enclosed in slashes at the set declaration or when loading a set from gdx via $load) and during execution time the membership is never changed. Therefore they are called *static sets*. In contrast, the elements of *dynamic sets* are not fixed, but may be added and removed during execution of the program. Dynamic sets are most often used as *controlling indices* in assignments or equation definitions and as the conditional set in a dollar-controlled indexed operation. We will first show how assignments are used to change set membership in dynamic sets. Then we will introduce set operations and the last part of this chapter covers dynamic sets in the context of dollar conditions.

4.24.2 Assigning Membership to Dynamic Sets

Dynamic Sets may be assigned to in a similar way as other data types. There are only two possible values: `yes` and `no`. Note that arithmetic operations *cannot* be performed on sets in the same way as on *value typed* identifiers (parameters, variables and equations subtypes). However, there are special set operations.

4.24.2.1 The Syntax

Like any other set, a dynamic set has to be declared before it may be used in the model. Often, a dynamic set is declared as subset of a static set. Dynamic sets in GAMS may also be multi-dimensional like static sets. The maximum number of permitted dimensions follows the rules of the basic Data Types and Definitions. For multi-dimensional dynamic sets the index sets can also be specified explicitly at declaration. That way dynamic sets are domain checked. Of course it is also possible to use dynamic sets that are not domain checked. This provides additional power and flexibility but also a lack of intelligibility and danger. Any label is legal as long as such a set's dimension, once established, is preserved.

In general, the syntax for assigning membership to dynamic sets in GAMS is:

```
set_name(index_list | label) = yes | no ;
```

`Set_name` is the internal name of the set in GAMS, `index_list` refers to the domain of the dynamic set and `label` is one specific element of the domain. An assignment statement may assign membership to the dynamic set either to the whole domain or to a subset of the domain or to one specific element. Note that, as usual, a label must appear in double or single quotes. `Yes` and `no` are keywords in GAMS. They are used to add members to or remove them from the dynamic set. Examples are given in the following subsections.

4.24.2.2 Illustrative Example

Throughout this chapter we will use examples adapted from the database model **[ZLOOF]** to illustrate the introduced concepts. Here we start with assignments of membership to dynamic sets.

```
Set  item            "all items"              / dish, ink, lipstick, pen, pencil, perfume /
     subitem1(item)  "first subset of item"   / pen, pencil /
     subitem2(item)  "second subset of item";

subitem1('ink')      = yes ;
subitem1('lipstick') = yes ;
subitem2(item)       = yes  ;
subitem2('perfume')  = no ;
display subitem1, subitem2;
```

Note that the sets `subitem1` and `subitem2` are declared like any other set. The two sets become dynamic as soon as they are assigned to a few lines later. They are also domain checked: the only members they will ever be able to have must also be members of the set `item`. The first assignment not only makes the set `subitem1` dynamic, it also has the effect that its superset `item` becomes a static set and from then on its membership is frozen. The first two assignments each add one new element to `subitem1`. Note that both are also elements of `item`, as required. The third assignment is an example of the familiar indexed assignment: `subitem2` is assigned all the members of `item`. The last assignment removes the label `'perfume'` from the dynamic set `subitem2`. The output generated by the display statement is shown below:

```
----      9 SET subitem1  first subset of item

ink     ,    lipstick,    pen     ,    pencil


----      9 SET subitem2  second subset of item

dish    ,    ink     ,    lipstick,    pen     ,    pencil
```

Note that even though the labels `'pen'` and `'pencil'` were declared to be members of the set `subitem1` *before* the assignment statements that added the labels `'ink'` and `'lipstick'` to the set, they appear in the listing above at the end. The reason is that elements are displayed in the internal order, which in this case is the order specified in the declaration of the set `item`.

Alternatively, the elements of the set `subitem2` could be assigned in the following way:

```
subitem2(item)     = no;
subitem2(subitem1) = yes;
subitem2('dish')   = yes;
```

The first statement removes all elements from the set `subitem2`. The second statement adds all elements of the set `subitem1`. Note that this assignment is permitted since the set `subitem1` is a proper subset of the set `item` (which is the domain of `subitem2`). The third statement adds one additional element.

4.24.2.3 Dynamic Sets with Multiple Indices

As mentioned earlier, dynamic sets may be multi-dimensional. The following lines continue the example above and illustrate assignments for multi-dimensional sets.

```
Sets sold(item)        "items sold" / pencil, pen /
     sup               "suppliers"  / bic, parker, waterman /
     supply(sold,sup) ;

supply('pencil','bic')  = yes ;
supply('pen',sup)       = yes ;
```

Note that `supply` is a two-dimensional dynamic set. It links sold items with their respective suppliers. Other examples with multi-dimensional dynamic sets are in subsections Dynamic Sets in Conditional Assignments and Conditional Indexed Operations with Dynamic Sets below.

All the mechanisms using asterisks and parenthesized lists that were introduced in section Multi-Dimensional Sets in chapter Set Definition are available for dynamic sets as well.

4.24.2.4 Equations Defined over the Domain of Dynamic Sets

Generally, dynamic sets are not permitted as domains in *declarations* of sets, variables, parameters and equations. However, they may be *referenced* and sometimes it is necessary to *define* an equation over a dynamic set.

Note

The trick is to *declare* the equation over the entire domain but *define* it over the dynamic set.

For example, defining an equation over a dynamic set can be necessary in models that will be solved for arbitrary groupings of regions simultaneously. We assume there are no explicit links between regions, but that we have a number of independent models with a common data definition and common logic. We illustrate with an artificial example, leaving out lots of details.

```
Set        allr            "all  regions"  /  N, S, W, E, N-E, S-W  /
           r(allr)         "region subset for particular solution"
           type            "set for various types of data" ;

Scalar     price                          /10/ ;
Parameter  revenue(allr);
Table      data(allr,type) "all other data ..." ;

Variables  activity1(allr) "first activity"
           activity2(allr) "second activity"
           revenue(allr)   "revenue"          ;

Equations  resource1(allr) "first resource constraint ..."
           prodbal1(allr)  "first production balance ..." ;

resource1(r)..  activity1(r)       =l=  data(r,'resource-1');
prodbal1(r)..   activity2(r)*price =e= revenue(r) ;
```

To repeat the important point: the equation is *declared* over the set `allr`, but *defined* over `r`, a subset. Note that the variables and data are *declared* over `allr` but *referenced* over `r`. Then the set `r` may be assigned arbitrary combinations of elements of the set `allr`, and the model may be solved any number of times for the chosen groupings of regions.

4.24.2.5 Assigning Membership to Singleton Sets

Singleton sets have only one element. Hence any assignment to a singleton set first clears or empties the set, no explicit action to clear the set is necessary. This is illustrated with the following example:

```
Set            i     "Static Set"            / a, b, c /
               ii(i) "Dynamic Set"           /    b    /;
Singleton Set  si(i) "Dynamic Singleton Set" /    b    /;

ii('c') = yes;
si('c') = yes;
display ii, si;
```

Note that both `ii` and `si` are subsets of the set `i`, but only `si` is declared as a singleton set. The assignment statements assign to both sets the element `'c'`. While `'c'` is *added* to the set `ii`, it *replaces* the original element in the singleton set `si`. The output from the display statement confirms this:

```
----      8 SET ii  Dynamic Set
b,    c

----      8 SET si  Dynamic Singleton Set
c
```

For more information on singleton sets in GAMS, see section Singleton Sets.

Attention

That an assignment to a `singleton set` first clears the set always, means that it is even cleared if there would be no change at all for a regular set:

```
Singleton Set s / 1 /;
s(s)$0 = yes;
display s;
```

Here is the output from the `display` statement in the listing file:

```
----      3 SET s

                                                    ( EMPTY )
```

The assignment behavior can be changed with the option and command line parameter strictSingleton which affects the behavior of a membership assignment to a Singleton Set. With `strictSingleton=0` GAMS does not complain about an assignment with more than one element on the right hand side but takes the first one. With `strictSingleton=1` (default), such an assignment raises an error. Consider the following example:

```
Set            i     "Static Set"            / a, b, c /
Singleton Set  si(i) "Dynamic Singleton Set";
si(i) = ord(i) > 1;
display si;
```

By default, the above code will trigger an error as an assignment to a singleton set with more than one element on the right hand side is forbidden:

```
*** Error at line 3: Multiple assignment to Singleton Set not allowed (see option strictSingleton)
```

However, with option (or command line parameter) `strictSingleton=0` GAMS does not complain about such an assignment with more than one element on the right hand side but takes the first one:

```
Set            i     "Static Set"            / a, b, c /
Singleton Set  si(i) "Dynamic Singleton Set";
option strictSingleton = 0;
si(i) = ord(i) > 1;
display si;
```

The output from the display statement confirms this:

```
----      5 SET si  Dynamic Singleton Set

b
```

4.24.3 Set Operations

GAMS provides symbols for arithmetic set operations that may be used with dynamic sets. An overview of the set operations in GAMS is given in Table 1. Examples and alternative formulations for each operation follow. Note that in the table below the set `i` is the static superset and the sets `j` and `k` are dynamic sets.

Set Operation	Operator	Description
Set Union	`j(i) + k(i)`	Returns a subset of `i` that contains all the elements of the sets `j` and `k`.
Set Intersection	`j(i) * k(i)`	Returns a subset of `i` that contains the elements of the set `j` that are also elements of the set `k`.
Set Complement	`not j(i)`	Returns a subset of `i` that contains all the elements of the set `i` that are *not* elements of the set `j`.
Set Difference	`j(i) - k(i)`	Returns a subset of `i` that contains all the elements of the set `j` that are *not* elements of the set `k`.

Table 1: Set Operations with Dynamic Sets

The following examples draw on the database model **[ZLOOF]** that we introduced above. Recall that the set `item` is the superset of the dynamic sets `subitem1` and `subitem2`. We add new dynamic sets for the results of the respective set operations. The following example illustrates that the dynamic set operations are equivalent to the following alternative ways of representation.

```
Sets union1(item), intersection1(item), complement1(item), difference1(item)
     union2(item), intersection2(item), complement2(item), difference2(item);

union1(item) = subitem1(item) + subitem2(item); display union1;
union2(subitem1) = yes; union2(subitem2) = yes; display union2;

intersection1(item) = subitem1(item) * subitem2(item);          display intersection1;
intersection2(item) = yes$(subitem1(item) and subitem2(item));  display intersection2;

complement1(item) = not subitem1(item);              display complement1;
complement2(item) = yes; complement2(subitem1) = no; display complement2;

difference1(item) = subitem2(item) - subitem1(item);                  display difference1;
difference2(item) = yes$(subitem2(item)); difference2(subitem1) = no; display difference2;
```

The display statements will show that the above assignment statements for each operation result in the same dynamic set like using the set operator. Observe that the alternative formulations for the set intersection and set difference involve conditional assignments. Conditional assignments in the context of dynamic sets are discussed in depth in the next section.

Note

> The indexed operation `sum` may be used for set unions. Similarly, the indexed operation `prod` may be used for set intersections. For examples see section Conditional Indexed Operations with Dynamic Sets below.

4.24.4 Using Dollar Controls with Dynamic Sets

The remainder of this chapter assumes familiarity with the dollar condition that is introduced in chapter Conditional Expressions, Assignments and Equations. All the dollar control machinery is available for use with dynamic sets. In fact, the full power of dynamic sets can be exploited using these dollar controls.

Recall that set membership of subsets and dynamic sets may be used as a logical condition; see subsection Logical Conditions: Set Membership and Set Functions for details. Set membership may also be a building block in complex logical conditions that are constructed using the logical operators not, and, or, xor, imp and eqv. Moreover, the set operations introduced in the previous section may also be used in logical conditions. Like other dollar conditions, dollar conditions with dynamic sets are used in the context of assignments, indexed operations and equations. We will discuss in detail each of these in the following subsections.

Apart from being part of logical conditions, dynamic sets may be assigned members with conditional assignments. Examples are given in the next subsection.

4.24.4.1 Dynamic Sets in Conditional Assignments

Dynamic sets may be used in two ways in conditional assignments: they may be the item on the left-hand side that is assigned to and they may be part of the logical condition. Below we present examples for both. The examples are again based on the database model [**ZLOOF**] that we introduced above.

```
Set  item            "all items"              / dish, ink, lipstick, pen, pencil, perfume /
     subitem1(item)  "first subset of item"   /       ink, lipstick, pen, pencil          /
     subitem2(item)  "second subset of item";

subitem2(item)$subitem1(item) = yes;
display subitem2;
```

The conditional assignment adds the members of dynamic set `subitem1` to the dynamic set `subitem2`. Thus `subitem2` will have the following elements:

```
----      6 SET subitem2  second subset of item

ink     ,    lipstick,    pen     ,    pencil
```

Note that instead of using `subitem1` in a dollar condition we could also write:

```
subitem2(subitem1) = yes;
```

In the next example of a conditional assignment, a dynamic set features in the logical condition on the right-hand side. The first statement clears the set `subitem2` of any previously assigned members and the second statement assigns all members of `subitem1` to `subitem2`. The following conditional assignment will have the same result:

```
subitem2(item) = no;
subitem2(item) = yes$subitem1(item);
```

The logical condition in this assignment is `subitem1(item)`. It is satisfied for all members of the set `subitem1`. Hence the statement assigns all elements of the domain `item` that are members of the set `subitem1` to the dynamic set `subitem2`. Note that in this assignment the dollar operator is on the right. In the section Dollar on the Right we show that conditional assignments with the dollar operator on the right-hand side imply an `if-then-else` structure where the `else` case is automatically zero. Unlike parameters, dynamic sets cannot be assigned the value of zero, they are assigned the value `no` instead. Therefore a more explicit formulation of the conditional assignment above would be:

```
subitem2(item) = no;
subitem2(item) = yes$subitem1(item) + no$(not subitem1(item));
```

For more on sets in logical conditions, see section Logical Conditions: Set Membership and Set Functions. For more on conditional assignments, see section Conditional Assignments.

4.24.4.2 Conditional Indexed Operations with Dynamic Sets

Indexed operations in GAMS are introduced in section Indexed Operations. They may be controlled by dollar conditions as discussed in section Conditional Indexed Operations. The domain of conditional indexed operations is often restricted by a set, called the conditional set. Dynamic sets may be used as conditional sets or they may be assigned to with a statement that features a conditional indexed operation on the right-hand side. We will illustrate both cases with examples.

Suppose we have a set of origins, a set of destinations and a table specifying the flight distance between them:

```
Set   i           "origins"       / Chicago, Philadelphia /
      j           "destinations"  / Vancouver, Bogota, Dublin, Rio, Marrakech / ;

Table d(i,j)      "distance (miles)"
                  Vancouver   Bogota   Dublin    Rio    Marrakech
    Chicago          1777      2691     3709     5202      4352
    Philadelphia     2438      2419     3306     4695      3757          ;
```

We wish to find the longest distance that we can travel given that we have a limit of 3500 miles.

```
Set   can_do(i,j) "connections with less than 3500 miles";
can_do(i,j)$(d(i,j) < 3500) = yes;
display can_do;
Scalar maxd  "longest distance possible"
maxd = smax((i,j)$can_do(i,j), d(i,j));
display maxd;
```

The dynamic set `can_do` contains all connections that are less than 3500 miles. The scalar `maxd` is defined by a conditional assignment where the indexed operation `smax` scans all entries of the parameter `d` whose label combinations are members of the set `can_do` and chooses the largest value. The output generated by the display statements is shown below:

```
----     11 SET can_do  connections with less than 3500 miles

               Vancouver      Bogota      Dublin

Chicago              YES         YES
Philadelphia         YES         YES         YES


----     15 PARAMETER maxd                =     3306.000  longest distance possible
```

There is a shorter alternative formulation for this assignment; see subsection Filtering through Dynamic Sets below for details.

Finally, we also wish to know which flight connection is linked to the longest possible distance. Consider the following two lines:

```
Singleton set maxc(i,j)  "maximum distance connection";
maxc(i,j) = yes$can_do(i,j) and (d(i,j) = maxd);
```

The dynamic singleton set is assigned the member of the dynamic set `can_do` whose distance equals the maximum distance.

The full power of indexed operators becomes apparent with multi-dimensional dynamic sets. As earlier in this chapter, we illustrate with fragments of code adapted from the relational database model [**ZLOOF**].

```
Set  dep         "departments"  / cosmetics, hardware, houshold, stationary, toy /
     sup         "suppliers"    / bic, dupont, parker, revlon /
     item        "items sold"   / dish, ink, lipstick, pen, pencil, perfume /

sales(dep,item)  "departments and items sold"  /
                 cosmetics.  (lipstick,perfume)
                 hardware.    ink
                 houshold.   (dish,pen)
                 stationary. (dish,ink,pen,pencil)
                 toy.        (ink,pen,pencil)         /

supply(item,sup) "items and suppliers"   /
                 dish.(bic,dupont)   , ink.(bic,parker)    , lipstick.revlon
                 pen.(parker,revlon) , pencil.(bic,parker) , perfume.revlon  /

Set g03(dep)    "departments selling items supplied by Parker";

g03(dep) = sum(item$supply(item,'parker'), sales(dep,item));
display g03;
```

The assignment above is used to create the set of departments that sell items supplied by `'parker'`. Note that the set `g03` is a subset of the set `dep`. Its members are specified by assignment, hence it is a dynamic set. Note that the assignment is made to a set, therefore the indexed operator `sum` refers to a set union (and not to an addition as would be the case if the assignment were made to a parameter). The indexed operation is controlled by the two-dimensional set `supply` with the label `'parker'` in the second index position. This logical condition is `TRUE` for all members of the set `supply` where the second index is `'parker'`. Hence the summation is over all items sold, provided that the supplier is `'parker'`. Given the declaration of the set `supply`, this means `'ink'`, `'pen'` and `'pencil'`. The associated departments are thus all departments except for `'cosmetics'`:

```
----     19 SET g03  departments selling items supplied by Parker

hardware  ,    houshold  ,    stationary,    toy
```

Now suppose we are interested in the departments that are selling *only* items supplied by `'parker'`. We introduce a new dynamic set `g11` and the following assignment adds the desired departments:

```
Set g11(dep) "departments only selling items supplied by parker";
g11(dep) = prod(sales(dep,item), supply(item,"parker"));
display g11;
```

Note that the indexed operation `prod` refers to set intersections in the context of assignments to dynamic sets. From all departments linked with items only those are included where *all* items sold are supplied by `'parker'`. This means that departments that additionally sell items that are not supplied by `'parker'` are excluded. Hence, only `'hardware'` and `'toy'` are added to `g11`.

4.24.4.3 Conditional Equations with Dynamic Sets

Recall that dollar conditions in the context of equations may restrict the domain of the equation and they may also feature in the algebraic formulation of the equation; see section Conditional Equations for more information. In both instances dynamic sets may be used as part of the logical condition. Dollar conditions with dynamic sets in the algebra of equations are similar to conditional assignments with dynamic sets; see section Dynamic Sets in Conditional Assignments above. The example that follows illustrates the use of a dynamic set to restrict the domain of definition of an equation. In section Equations Defined over the Domain of Dynamic Sets above we had the following equation definition:

```
prodbal1(r)..            activity2(r)*price    =e= revenue(r)    ;
```

Recall that `r` is a dynamic set and a subset of the set `allr`. Hence this equation may be rewritten in the following way:

```
prodbal1(allr)$r(allr).. activity2(allr)*price =e= revenue(allr) ;
```

Note that both formulations achieve the same result: restricting the domain of definition to those elements that belong to the dynamic set `r`. While in the second formulation the condition is specified explicitly, in the first formulation the domain is *filtered* through the dynamic set `r`. This is the topic of the next subsection.

4.24.4.4 Filtering through Dynamic Sets

The filtering process is introduced and explained in section Filtering Sets in chapter Conditional Expressions, Assignments a
In certain circumstances it is an alternative to the dollar condition to restrict the domain of equations, sets, variables, parameters and indexed operations. We already saw an example for restricting the domain of definition of an equation in the previous subsection. The next example refers to restricting the domain in an indexed operation. In section Conditional Indexed Operations with Dynamic Sets we had the following assignment:

```
maxd = smax((i,j)$can_do(i,j), d(i,j));
```

Recall that `maxd` is a scalar, `i` and `j` are sets, `can_do` is a dynamic set and `d` is a two-dimensional parameter. Note that the conditional set is the dynamic set `can_do`. The assignment may be rewritten in the following way:

```
maxd = smax(can_do(i,j), d(i,j));
```

Here the indexed operation is filtered through the dynamic set `can_do`, a dollar condition is not necessary.

4.25 Sets as Sequences: Ordered Sets

4.25.1 Introduction

Sets are introduced in chapter Set Definition. There we state that in general, sets in GAMS are regarded as an unordered collection of labels. However, in some contexts, say, multi-period planning models, some sets need to be treated as if they were sequences. In this chapter we will establish the notion of *ordered sets* and we will cover their special features and the associated operations.

Examples where ordered sets are needed include economic models that explicitly represent conditions in different time periods that are linked, location problems where the formulation may require a representation of contiguous areas, as in a grid representation of a city, scheduling problems and programs that model stocks of capital with equations of the form '*stocks at the end of period n are equal to stocks at the end of period* $n-1$ *plus net gains during period n*'.

Note

Models involving sequences of time periods are often called *dynamic models*, because they describe how conditions change over time. This use of the word *dynamic* unfortunately has a different meaning from that used in connection with dynamic sets, but this is unavoidable.

4.25.2 Ordered and Unordered Sets

Certain one-dimensional sets may be treated as if they were a sequence. Those sets need to be ordered and static. A one-dimensional set is ordered if the definition or initialization of the elements in the set corresponds to the order of the labels in the GAMS Entry order.

Note

- The GAMS entry order is the order in which the individual labels first appear in the GAMS program, either explicitly or as a result of using the shorthand asterisk notation.
- For the sake of simplicity, sets that are static and ordered are often just referred to as *ordered sets*.

GAMS maintains a *unique element* list where all labels that are used as elements in one or more sets are listed. The order of the elements in any one set is the same as the order of those elements in the unique element list. This means that the order of a set may not be what it appears to be if some of the labels were used in an earlier definition. The internal GAMS order of the labels can be made visible with the dollar control option $onUELlist. This directive generates a map that is shown in the compilation output of the listing file. For details on the listing file and GAMS output in general, see chapter GAMS Output. A good rule of thumb is that if the user wants a set to be ordered and the labels in the set have not been used already, then they will be ordered.

In the example below we show ordered and unordered sets and the map showing the order. The input is:

```
$onUELlist
Set    t1  / 1987, 1988, 1989, 1990, 1991 /
       t2  / 1983, 1984, 1985, 1986, 1987 /
       t3  / 1987, 1989, 1991, 1983, 1985 / ;
```

Note that the label "1987" is the first label seen by GAMS. It appears again as the last label in the initialization list for the set t2. This means that the set t2 is *not* ordered and any attempt to use t2 in a context implying order, such as the operations introduced later in this chapter, will cause error messages. Observe that the set t3 is ordered, as all the members of t3 have appeared in the GAMS program before, and in the same order that they are listed in the definition of t3.

The unique element listing below shows the entry order (the important one) and the sorted order, obtained by sorting the labels into dictionary order. The single digits on the left are the sequence numbers of the first label on that line.

```
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
Unique Element Listing

Unique Elements in Entry Order
    1   1987       1988       1989       1990       1991       1983
    7   1984       1985       1986

Unique Elements in Sorted Order
    1   1983       1984       1985       1986       1987       1988
    7   1989       1990       1991
```

A set can always be made ordered by moving its declaration closer to the beginning of the program. With these restrictions in mind, we will continue in the next section with the operations that may be used with ordered sets.

4.25.3 Sorting a Set

Note that besides the entry order of unique elements there is also a sorted (alphabetical) order. To obtain the sorted order (of an ordered or unordered set) the special predefined two-dimensional set SortedUELs(*,*) can be used. Consider the following example where set j is reported in an alphabetically sorted format:

```
set j / c, a, b, 1, 2, 11 /;
display j;
alias(*,u);
file ordered /ordered.txt/;
loop(SortedUels(u,j), put ordered j.tl:0 '  ');
putclose ordered;
```

The output generated by the display statement follows:

```
----      2 SET j

c ,    a ,    b ,    1 ,    2 ,    11
```

Note that, as expected, the display statement shows the elements of set j in the entry order *not* in alphabetical order. However, the elements are listed in alphabetical order in the file ordered.txt. Note furthermore, that alphabetical sorting leads to an order where e.g. 11 precedes 2.

```
1  11  2  a  b  c
```

In the example above u is aliased with the Universal Set. For an introduction to writing external files with put, see chapter The Put Writing Facility.

4.25.4 Ord and Card Operators

As stated in section Labels in chapter GAMS Programs, labels in GAMS do not have a numerical value. The examples used were that the label '1986' does not have a numerical value of 1986 and the label '01' is different from the label '1'. This section introduces two operators - ord and card - that return integer values when applied to sets. While the integer values returned do not represent the numerical value of the label, they can be used for the same purpose.

Note

GAMS provides some string manipulation capability by extending the card and ord operators to also work on strings.

4.25.4.1 The Ord Operator

The operator ord returns the relative position of a member in a set.

Attention

By default the operator ord may be used only with one-dimensional sets that are static and ordered.

Some examples show the usage.

```
Set       t      "time periods"  / 1985*1995 / ;
Parameter val(t) ;
val(t)  =  ord(t);
```

Note that as a result of the statements above, the value of val('1985') will be 1, val('1986') will be 2 and so on.

A common use of ord is in setting up vectors that represent quantities growing in some analytically specified way. For example, suppose a country has 56 million people in the base period and the population is growing at the rate of 1.5 percent per year. Then the population in succeeding years can be calculated as follows:

```
population(t)  =  56*(1.015**(ord(t) - 1)) ;
```

It is often useful to simulate general matrix operations in GAMS. The first index of a two dimensional parameter can conveniently represent the rows, the second the columns and order is necessary. The example below shows how to set the upper triangle of a matrix equal to the row index plus the column index, and the diagonal and lower triangle to zero.

```
Set       i       "row and column labels"        / x1*x10 /;
alias (i,j);
Parameter a(i,j)  "a general square matrix";
a(i,j) $ (ord(i) < ord(j)) = ord(i) + ord(j);
```

Note that in the assignment statement the logical condition `(ord(i) < ord(j))` restricts the assignment to the entries of the upper triangle. For more information on logical conditions and conditional assignments in GAMS, see sections Logical Conditions and Conditional Assignments respectively.

Note that the strict requirement that the set needs to be ordered for the operator `ord` to be used may be relaxed with a dollar control option called offOrder. Consider the following lines where we revisit the example from the previous section.

```
$offOrder
Set       t1    / 1987, 1988, 1989, 1990, 1991 /
          t2    / 1983, 1984, 1985, 1986, 1987 /;
Parameter p(t2);
p(t2) = ord(t2);
display p;
```

Note that, as we explained above, the set `t2` is *not* ordered. Therefore using `ord(t2)` somewhere in the model will usually cause the program to be terminated with an error message. However, with the dollar control option `offOrder` active, the set `t2` is treated as if it were ordered and the display statement will generate the following output:

```
----      6 PARAMETER p
1987 1.000,    1983 2.000,    1984 3.000,    1985 4.000,    1986 5.000
```

While this may be useful in some circumstances, the option comes with a price, since the system will not be able to diagnose odd and incorrect formulations and data sets. The strict requirement that the set needs to be ordered for the `ord` operator can be turned on again via onOrder.

4.25.4.2 The Card Operator

The operator `card` returns the number of elements in a set.

Note

The operator `card` may be used with any set: static and dynamic sets, ordered and unordered sets.

The following example illustrates its use:

```
Set    t  "time periods"  / 1985*1995 /;
Scalar s;
s  =  card(t);
Display s;
```

Note that `s` will be assigned the value 11 since the set `t` has 11 elements.

A common use of `card` is to specify some condition only for the final period, for example to fix a variable. Consider the following artificial example:

```
c.fx(t)$(ord(t) = card(t)) = demand(t);
```

Note that the logical condition `(ord(t) = card(t))` restricts the assignment to the last element of the set `t:` no assignment is made for other members of `t`. The advantage of this way of fixing the variable `c` is that the membership of `t` can be changed safely and this statement will always fix `c` for the last element. For more information on logical conditions and conditional assignments in GAMS, see sections Logical Conditions and Conditional Assignments respectively.

4.25.5 Lag and Lead Operators

Lag and lead operators are used to relate the *current* member of an ordered set to the *previous* or *next* member of the set. GAMS provides two forms of lag and lead operators (linear and circular), they are summarized in Table 1. Note that in the table below `t` is a member of an ordered set and `n` is a positive integer.

Operation	Symbol	Description
Linear Lag	`t - n`	Refers to the element of the ordered set whose relative position in the set is `ord(t)-n`.
Linear Lead	`t + n`	Refers to the element of the ordered set whose relative position in the set is `ord(t)+n`.
Circular Lag	`t -- n`	Same as `t - n`, only here the first element of the set is assumed to be preceded by the last element of the set, thus forming a circle of elements.
Circular Lead	`t ++ n`	Same as `t + n`, only here the last element of the set is assumed to be followed by the first element of the set, thus forming a circle of elements.

Table 1: Linear and Circular Lag and Lead Operators

Note that the only difference between *linear* and *circular* lag and lead operators is how endpoints are treated. *Linear* operators assume that there are no members preceding the first and following the last member of the ordered set. This assumption may result in elements of the set being referenced that actually do not exist. Therefore the user must think carefully about the treatment of endpoints: models with linear lag and lead operators will need special exception handling logic to deal with them. The next two sections will describe how this issue is handled in GAMS in the context in which these operators are typically used: assignments and equations. Linear lag and lead operators are useful for modeling time periods that do not repeat, like a set of years (say 1990 to 1997).

Circular lag and lead operators assume that the first and last members of the set are adjacent, so as to form a circular sequence of members. This means that `'first--1'` is a reference to `'last'` and `'last++2'` is the same as `'first++1'`. All references and assignments are defined. The assumption of circularity is useful for modeling time periods that repeat, such as months of the year or hours in the day. It is quite natural to think of January as the month following December. Agricultural farm budget models and workforce scheduling models are examples of applications where circular leads occur naturally.

Note

- GAMS is able to distinguish the linear lag and lead operators `'-'` and `'+'` from arithmetic operators by context. To avoid ambiguity, GAMS does not allow to mix lag and lead operators with arithmetic operators. For example, `i+1+1` is not allowed, but writing `i+(1+1)` would work.
- Observe that the lag (or lead) value `n` does not have to be an explicit constant: it can be an arbitrary expression, provided that it evaluates to an integer. If it does not, error messages will be produced. A negative result causes a switch in sense (from lag to lead, for example).

Note that if lag and lead operators are used with an *unordered* set, the program will terminate with an error message. The strict requirement that the set be ordered may be relaxed with the dollar control option offOrder. If the directive `$offOrder` is added, in the lines that follow unordered sets are treated as if they were ordered and therefore lag and lead operators may be used with them. While this may be advantageous in some circumstances, the option comes with a price, since the system will not be able to diagnose odd and incorrect formulations and data sets. The strict requirement that the set needs to be ordered for the use of lag and lead operators can be turned on again via onOrder.

The next two subsections will give some illustrative examples on the use of lag and lead operators in assignment statements and in equations respectively.

4.25.5.1 Lags and Leads in Assignments

Lag and lead operators may be used in assignment statements. The use of a lag or lead operator on the right-hand side of an assignment is called a *reference*, while their use on the left-hand side is called an *assignment* and involves the definition of a domain of the assignment. The concepts behind reference and assignment are equally valid for the linear and circular forms of the lag and lead operators. However, the importance of the distinction between reference and assignment is not pronounced for circular lag and lead operators, because non-existent elements are not referred to in this case.

Note

A reference to a non-existent element causes the default value zero to be used, whereas an attempt to assign to a non-existent element results in no assignment being made.

We will first illustrate *linear* lag and lead operators for reference and assignment. Then we will turn to the *circular* form of the operators.

Linear Lag and Lead Operators in Assignments - Reference

Consider the following example, where the lag operator is used on the right-hand side of an assignment statement:

```
Set       t             "time sequence"  / y-1987*y-1991 /;
Parameter a(t), b(t);
a(t)  =  1986 + ord(t);
b(t)  =  -1;
b(t)  =  a(t-1);
option decimals = 0;
display a, b;
```

Note that the option statement suppresses the decimal places from the display statement.

```
----      7 PARAMETER a
y-1987 1987,    y-1988 1988,    y-1989 1989,    y-1990 1990,    y-1991 1991

----      7 PARAMETER b
y-1988 1987,    y-1989 1988,    y-1990 1989,    y-1991 1990
```

Note that, as expected, the values for the parameter `a` are 1987, 1988 up to 1991 corresponding to the labels `"y-1987"`, `"y-1988"` and so on. Observe that the parameter `b` is initialized to −1 so that the result of the next assignment can be seen clearly. Note that in the last assignment the lag operator `'-'` is used on the right-hand side, resulting in the values for `b` to equal the values for `a` from the *previous* period. If there is no previous period, as with the first element, `"y-1987"`, the value zero is assigned, replacing the previous value of −1 (values of zero for parameters are not displayed).

Linear Lag and Lead Operators in Assignments - Assignment

Consider the following variation of the previous example. Here the lead operator is used on the left-hand side of an assignment statement:

```
Set       t            "time sequence"  / y-1987*y-1991 / ;
Parameter a(t), c(t) ;
a(t)    =  1986 + ord(t) ;
c(t)    =  -1;
c(t+2)  =  a(t) ;
option decimals = 0;
display a, c ;
```

Note that, as before, the option statement suppresses the decimal places from the display statement.

```
----      7 PARAMETER a
y-1987 1987,    y-1988 1988,    y-1989 1989,    y-1990 1990,    y-1991 1991

----      7 PARAMETER c
y-1987   -1,    y-1988   -1,    y-1989 1987,    y-1990 1988,    y-1991 1989
```

The assignment to `a` is explained in the previous subsection. Note that the parameter `c` is initialized to −1. The assignment to `c` involving the lead operator on the left-hand side needs special attention. It is best to spell out step by step how this assignment is made. For each member of `t` in sequence, find the member of `c` associated with `t+2`. If it exists, replace its value with the value of `a(t)`. If not (as with labels `"y-1990"` and `"y-1991"`) make no assignment. The first member of the set `t` is `"y-1987"`, therefore the first assignment is made to `c("y-1989")` which takes the value of `a("y-1987")`, that is 1987. No assignments at all are made to `c("y-1987")` and `c("y-1988")`: these two retain their previous values of −1.

Circular Lag and Lead Operators in Assignments

The following example illustrates the use of circular lag and lead operators in assignment statements.

```
Set       s          "seasons" / spring, summer, autumn, winter /;
Parameter val(s)                / spring 10, summer 15, autumn 12, winter 8 /
          lagval(s)
          leadval(s);
lagval(s)     = -1 ;
lagval(s)     = val(s--2) ;
leadval(s)    = -1 ;
leadval(s++1) = val(s) ;
option decimals = 0;
display val, lagval, leadval;
```

Note that, as before, the option statement suppresses the decimal places from the display statement. The results are shown below.

```
-----     10 PARAMETER val

spring 10,    summer 15,    autumn 12,    winter  8


----     10 PARAMETER lagval

spring 12,    summer  8,    autumn 10,    winter 15


----     10 PARAMETER leadval

spring  8,    summer 10,    autumn 15,    winter 12
```

In the example parameter `lagval` is used for reference while `leadval` is used for assignment. Notice that the case of circular lag and lead operators does not refer to any non-existent elements. The difference between reference and assignment is therefore not important. Note that the following two statements from the example above,

```
lagval(s)     = val(s--2) ;
leadval(s++1) = val(s) ;
```

are equivalent to

```
lagval(s++2)  = val(s) ;
leadval(s)    = val(s--1) ;
```

The use of reference and assignment have been reversed with no difference in effect.

4.25.5.2 Lags and Leads in Equations

The principles established in the previous section follow quite naturally into equation definitions. A lag or lead to the left of the `'..'` symbol is a modification of the domain of definition of the equation. The linear form may cause one or more individual equations to be suppressed. A lag or lead operation in the body of an equation (to the right of the `'..'` symbol) is a reference. If the associated label is not defined, the term vanishes.

Note

> All lag and lead operands must be *exogenous*. For more information, see section Functions in Equation Definitions.

In the next two subsections we will provide examples illustrating the use of the *linear* form of the lag and lead operators in equations to modify the domain of definition and for reference respectively. In the last subsection we will turn to *circular* lag and lead operators in equations.

Linear Lag and Lead Operators in Equations - Domain Control

Consider the following simple artificial multi-period example. We specify a complete model and encourage users to solve it and further explore it.

```
Sets      t           / t1*t5 /
          tfirst(t);
Parameter i(t)        / #t 1 /;
Scalar    k0          / 3.00 /;

tfirst(t)  =  yes$(ord(t) = 1);

Variables k(t), z;
k.fx(tfirst) = k0;

Equations kk(t), dummy;
dummy..      z =e= 0;
kk(t+1)..    k(t+1) =e=  k(t) + i(t);

model m /all/;
option limrow = 10;
solve m using lp min z;
```

Note that the equation `kk` is declared over the set `t`, but it is defined over the domain `(t+1)`. Therefore the first equation that will be generated is the following:

```
k('t2')  =e=  k('t1') + i('t1');
```

Note that the value of the variable `k('t1')` is fixed at the value of scalar `k0`. Observe that for the last element of `t`, the term `k(t+1)` is not defined and therefore the equation will not be generated. If lag or lead operators are used in the domain of definition of an equation, the equation listing can be a useful tool to verify whether the equations that have been generated are those that were intended:

```
kk(t2)..  - k(t1) + k(t2) =E= 1 ; (LHS = 0, INFES = 1 ****)
kk(t3)..  - k(t2) + k(t3) =E= 1 ; (LHS = 0, INFES = 1 ****)
kk(t4)..  - k(t3) + k(t4) =E= 1 ; (LHS = 0, INFES = 1 ****)
kk(t5)..  - k(t4) + k(t5) =E= 1 ; (LHS = 0, INFES = 1 ****)
```

To summarize, the lead operator in the domain of definition has restricted the number of constraints generated so that there are no references to non-existent variables.

For a more realistic model that illustrates the usage of linear lag operators in equations, see for example the optimal economic growth model [**RAMSEY**].

Linear Lag and Lead Operators in Equations - Reference

In the previous subsection we showed how to write the equation `kk` using the lead operator for domain control in combination with fixing the variable `k(tfirst)` to `k0`. An alternative formulation could neglect the fixing of `k(tfirst)` and use a lag operator and a dollar condition in the *body* of the equation while the *domain* of definition is unrestricted:

```
kk(t)..  k(t) =e= k(t-1) + i(t-1) + k0$tfirst(t);
```

Note that for the first element of the set `t` the terms `k(t-1)` and `i(t-1)` are not defined and therefore vanish. Without the conditional term, the resulting equation would be:

```
k('t1') =e= 0;
```

However, this would lead to different results as `k('t1')` would not be set to the value of `k0` anymore. Therefore the conditional expression `k0$tfirst(t)` is added. Observe that in this formulation equations are generated for *all* time periods, no equation is suppressed.

In general, the choice between using lag and lead operators as reference like in the last example or in domain control is often a matter of taste.

Circular Lag and Lead Operators in Equations

In the case of circular lag and lead operators, the difference between their use in domain control and as reference is not important because it does not lead to any equations or terms being suppressed. Consider the following artificial example.

```
Set      s        "seasons" / spring, summer, autumn, winter /;

Variable produ(s) "amount of goods produced in each season"
         avail(s) "amount of goods available in each season"
         sold(s)  "amount of goods sold in each season";

Equation matbal(s);

matbal(s)..  avail(s++1) =e= avail(s) + produ(s) - sold(s);
```

In this example four individual equations are generated. They are listed below.

```
avail('summer')  =e=  avail('spring') + produ('spring') - sold('spring');
avail('autumn')  =e=  avail('summer') + produ('summer') - sold('summer');
avail('winter')  =e=  avail('autumn') + produ('autumn') - sold('autumn');
avail('spring')  =e=  avail('winter') + produ('winter') - sold('winter');
```

Note that for the last element of the set `s` the term `avail(s++1)` is evaluated to `avail('spring')`. This term is well defined and therefore it does not vanish. Similarly, using the circular lead operator in the domain of definition like in the following line will result in the same four equations being generated as above and no equation being suppressed.

```
matbal(s++1)..  avail(s++1) =e= avail(s) + produ(s) - sold(s);
```

4.25.6 Summary

This chapter introduced the concept of ordered sets. All the features in GAMS that deal with this issue including the `ord` and `card` operators, as well as the linear and circular forms of the lag and lead operators were described in detail.

4.26 Data Manipulations with Parameters

4.26.1 Introduction

In this chapter we explicitly cover only *parameter* manipulation, including all aspects of data handling. Much of this material is relevant elsewhere (e.g. to sets), but for specifics related to assignment to sets, to conditions, and to assignments within flow control constructs such as the `loop` statement, see chapters Dynamic Sets, Conditional Expressions, Assignments and Equations and Programming Flow Control Features respectively.

Once initialized, data often requires manipulation in order to bring it into a form most suitable for use in a model or an application. The assignment statement is the way to do this. All the possible components of the assignment statement except conditions are introduced and discussed in this chapter.

4.26.2 The Assignment Statement

The assignment statement is the fundamental data manipulation statement in GAMS. It may be used to define or alter values associated with sets, parameters, variables or equations.

A simple assignment is written in the style associated with many other computer languages. GAMS uses the traditional symbols for addition `(+)`, subtraction `(-)`, multiplication `(*)` and division `(/)`. We will use them in the examples that follow, and give more details in section Expressions.

4.26.2.1 Scalar Assignments

Consider the following artificial sequence:

```
scalar x / 1.5 /;
x = 1.2;
x = x + 2;
```

The scalar `x` is initialized to be 1.5. The second statement changes the value to 1.2, and the third changes it to 3.2. The second and third statements are assignments: each replaces the current value of `x` with a new one.

Note that the same symbol can be used on the left and right of the `=` sign. The new value is not available until the calculation is complete, and the operation gives the expected result.

Note

> An assignment cannot start with a reserved word. A semicolon is therefore required as a delimiter before all assignments.

Note that extended range identifiers (e.g. `INF`) and acronyms may also be used in assignment statements. For specific details, see sections Extended Range Arithmetic and Acronym Usage respectively.

4.26.2.2 Indexed Assignments

The GAMS syntax for performing indexed assignments is extremely powerful. This operation offers what may be thought of as simultaneous or parallel assignments and provides a concise way of specifying large amounts of data. Consider the mathematical statement $DJ_d = 2.75 \cdot DA_d$ for all elements of d. This means that for every member of the set d, a value is assigned to DJ. This can be written in GAMS as follows:

```
dj(d) = 2.75*da(d) ;
```

This assignment is known technically as an *indexed assignment* and set `d` as the controlling index or controlling set.

Attention

> The index set(s) on the left hand side of an indexed assignment are referred to synonymously as the controlling indices, controlling sets, or controlling domain of the assignment.

The extension to two or more controlling indices should be obvious. There will be an assignment made for each label combination that can be constructed using the indices inside the parentheses. Consider the following example of an assignment to all 100 data elements of the parameter `a`.

```
sets  row            / r-1*r-10 /
      col            / c-1*c-10 /
      sro(row)       / r-7*r-10 / ;
parameters  r(row)   /r-1*r-7 4, r-8*r-10 5/
            c(col)   /c-1*c-5 3, c-6*c-10 2/;
parameters  a(row,col);
a(row,col)  =  13.2 + r(row)*c(col)  ;
```

The calculation in the last statement is carried out for each of the 100 unique two-label combinations that can be formed from the elements of `row` and `col`. An explicit formulation of the first of these assignments follows:

```
a('r-1','c-1') = 13.2 + r('r-1')*c('c-1');
```

Note that for indexed assignments a copy of the symbols on the right hand side is installed before the assignment is carried out. That means it does not work "in-place" or recursively. Consider the following example where we compute the first ten Fibonacci numbers and store them in parameter `f` using a loop. The example also illustrates how such a recursive calculation does not work with a parallel assignment statement for parameter `g`.

```
set i / i1*i10 /
parameter
    f(i) / i1 1 /
    g(i) / i1 1 /;
loop(i$(ord(i)>=2),
  f(i) = f(i-2) + f(i-1);
);
g(i)$(ord(i)>=2) = g(i-2) + g(i-1)
display f,g;
```

The display statement results in the following output.

```
----      9 PARAMETER f

i1   1.000,    i2   1.000,    i3   2.000,    i4   3.000,    i5   5.000
i6   8.000,    i7  13.000,    i8  21.000,    i9  34.000,    i10 55.000


----      9 PARAMETER g

i1 1.000,    i2 1.000,    i3 1.000
```

4.26.2.3 Restricting the Domain in Assignments

Sometimes it is necessary to make assignments over selected elements of a set instead of over the entire domain. There are several ways to accomplish this: using explicit labels, subsets, conditionals and tuples.

Restricting the Domain: Explicit Labels

The strongest restriction of the domain is assigning a value to just one element. Labels may be used explicitly in the context of assignments to accomplish this. The following example illustrates:

```
a('r-7','c-4') = -2.36 ;
```

This statement assigns a constant value to just one element of the parameter `a`. All other elements of `a` remain unchanged. Labels must be quoted when used in this way.

Restricting the Domain: Subsets

In general, wherever a set name may occur in an indexed assignment, a subset may be used instead.

Consider the following example:

```
a(sro,'col-10') = 2.44 -33*r(sro) ;
```

Since the set `sro` was declared as a subset of the set `row`, we can use `sro` as a controlling index in the assignment above to make the assignment only for the elements of `sro`.

Restricting the Domain: Conditionals

Conditional assignments are discussed in detail in section Conditional Assignments in chapter Conditional Expressions, Assignments and Equations. For details on the types of logical conditions, see section Logical Conditions. Here we present a simple example to illustrate:

```
a(row,col)$[a(row,col) >= 100] = INF ;
```

This assignment has the following effect: all elements of the parameter `a` whose value was at least 100 are assigned the value INF, while all other elements of `a` remain unchanged.

Restricting the Domain: Tuples

Tuples or multi-dimensional sets are introduced in section Many-to-Many Mapping. In this simple example we show how they may be used to restrict the domain. The example builds on the first example in this section. We repeat the whole code here for clarity.

```
sets  row            / r-1*r-10 /
      col            / c-1*c-10 /
      sro(row)       / r-7*r-10 / ;
set tuple(row,col)   /r-1.c-1, r-1.c-10, r-10.c-1, r-10.c-10/;
parameters  r(row)   /r-1*r-7 4, r-8*r-10 5/
            c(col)   /c-1*c-5 3, c-6*c-10 2/;
parameters  a(row,col), b(row,row);
a(row,col)  =  13.2 + r(row)*c(col)  ;
a(tuple(row,col)) = 7 + r(row)*c(col)  ;
a(tuple) = 0.25 * a(tuple)  ;
```

Note that we have introduced the new set `tuple`. It is two-dimensional and contains just four elements. As before, the parameter `a` is first assigned values for all its 100 elements. We then change some of these values using the set `tuple` as domain. The values of the elements of the parameter `a` that are not elements of the set `tuple` remain unchanged.

For a more elaborate example involving tuples, see section Filtering Sets in Assignments.

4.26.2.4 Issues with Controlling Indices

Attention

> The number of controlling indices on the left of the `=` sign should be at least as large as the number of indices on the right. There should be no index on the right-hand side of the assignment that is not present on the left unless it is operated on by an indexed operator. For more on indexed operators, see section Indexed Operations.

Consider the following statement:

```
a(row,'col-2') = 22 - c(col) ;
```

GAMS will flag this statement as an error since `col` is an index on the right-hand side of the equation but not on the left.

Note that there would be no error here if `col` were a `singleton set`. Since there is only one element in a singleton set, the intent and behavior is well-defined even when `col` is not under control.

Attention

> Each set is counted only once to determine the number of controlling indices. If the intent is for a set to appear independently more than once within the controlling domain, the second and subsequent occurrences of the set should be `aliases` of the original set, so that the number of controlling indices is equal to the number of indices. For details on aliases, see section The Alias Statement: Multiple Names for a Set.

Consider the following statement as an illustration:

```
b(row,row) = 7.7 - r(row) ;
```

This statement has only one controlling index, namely `row`. One element (on the diagonal of `b`) is assigned for each element of `row`, for a total of 10 assigned values. None of the off-diagonal elements of `b` will be changed!

If the intent is to assign values to each element of `b`, this can be done by introducing an alias `rowp` for `row` and using this alias in the second index position. There will then be two controlling indices and GAMS will make assignments over all 100 values of the full Cartesian product. The following example illustrates this method:

```
alias(row,rowp) ;
b(row,rowp) = 7.7 - (r(row) + r(rowp))/2 ;
```

4.26.3 Expressions

An expression is an arbitrarily complicated specification for a calculation, with parentheses nested as needed for clarity and intent. In this section the discussion of parameter assignments will continue by showing in more detail the expressions that may be used on the right of the = sign. We will cover standard arithmetic operations and indexed operations here, and functions and extended range arithmetic in the next two sections.

4.26.3.1 Standard Arithmetic Operations

The standard arithmetic symbols and operations and their order of precedence are given in Table 1. Note that 1 denotes the highest order of precedence and 3 denotes the lowest order of precedence. Parentheses can be used to override the default precedence order in the usual way. Operators (including exponentiation) on the same level are evaluated from left to right.

Operation	Symbol	Order of Precedence
Exponentiation	**	1
Multiplication	*	2
Division	/	2
Addition	+	3
Subtraction	-	3

Table 1: Standard Arithmetic Operations

Note that the full GAMS operator precedence hierarchy also includes logical operators; it is given in section Mixed Logical Conditions. Note further that the symbols for addition, subtraction and multiplication

have a different meaning if they are used in the context of sets. For details see sections Set Operations and Lag and Lead Operators.

Attention

The operation `x**y` is equivalent to the function rPower(x,y) and is calculated internally as $e^{y \times \log(x)}$. This operation is not defined if x is negative; an error will result in this case. If the possibility of negative values for x is to be admitted *and* the exponent is known to be an integer, then the function power(x,n) may be used.

Like many GAMS intrinsic functions, the operation `x**y` is not defined for all possible input values. When evaluating nonlinear functions and operators GAMS guards against evaluating at or very near singular points and in such cases signals an error or returns an appropriate function value.

Note

As usual, operations within parentheses or brackets are evaluated before other numerical calculations, where the innermost parentheses are resolved first. Any of the pairs `()`, `[]` and {} are allowed.

Consider for example:

```
x = 5 + 4*3**2;
```

For clarity, this could have been written as follows:

```
x = 5 + (4*[3**2]) ;
```

In both cases the result is 41.

Note

It is often better to use parentheses than to rely on the order of precedence of operators, since this prevents errors and clarifies intentions.

Note that expressions may be freely continued over many lines: an end of line is permissible at any point where a blank may be used. Blanks may be used for readability around identifiers, parentheses and operation symbols. Blanks are not allowed within identifiers or numbers, and are significant inside the quotes used to delimit labels.

4.26.3.2 Indexed Operations

In addition to the simple operations in Table 1, GAMS also provides the following six indexed operations.

Operation	Keyword
Summation	`sum`
Product	`prod`
Minimum value	`smin`
Maximum value	`smax`
Conjunction	`sand`
Disjunction	`sor`

Table 2: Indexed Operations

These six operations are performed over one or more controlling indices. The syntax in GAMS for these operations is:

```
indexed_op( (index_list), expression);
```

`Indexed_op` is one of the four keywords for indexed operations, `index_list` is the list of the controlling indices and `expression` is the expression to be evaluated. If there is only one controlling index, the parentheses around it may be removed. Consider the following simple example adapted from [**ANDEAN**]:

```
sets i   plants  / cartagena, callao, moron /
     m   product / nitr-acid, sulf-acid, amm-sulf /;

parameter capacity(i,m) capacity in tons per day
          totcap(m)     total capacity by process ;

totcap(m) = sum(i, capacity(i,m));
```

The index over which the summation is done, `i`, is separated from the reserved word `sum` by a left parenthesis and from the data term `capacity(i,m)` by a comma. The set `i` is called the controlling index for this operation. The scope of the control is the pair of parentheses `()` that start immediately after the sum. Note that using normal mathematical representation the last line could be written as: $totC_m = \sum_i C_{im}$.

It is also possible to sum simultaneously over the domain of two or more sets as in the first assignment that follows. The second assignment demonstrates the use of a less trivial expression than an identifier within the indexed operation.

```
count = sum((i,j), a(i,j)) ;
emp = sum(t, l(t)*m(t)) ;
```

The equivalent mathematical forms are:

$$count = \sum_i \sum_j A_{ij} \quad \text{and} \quad emp = \sum_t L_t M_t.$$

Note that the following alternative notation may be used for the first assignment above:

```
count = sum(i, sum(j, a(i,j)));
```

In the context of sets the `sum` operator may be used to compute the number of elements in a set and for projections. For details see section Projection and Aggregation of Sets.

The `smin` and `smax` operations are used to find the largest and smallest values over the domain of the index set or sets. The index for the `smin` and `smax` operators is specified in the same manner as in the index for the `sum` operator. In the following example we want to find the largest capacity:

```
lrgunit = smax((i,m),capacity(i,m));
```

Attention

The indexed operations `smin` and `smax` may be used in equation definitions only if the corresponding model is of type DNLP. For more on GAMS model types, see GAMS Model Types.

Note

- In the context of assignment statements, the attributes of variables and equations (e.g. `x.up(i,j)`) may be used in indexed operations just as scalars and parameters are used. For more on variable and equations attributes, see sections Variable Attributes and Equation Attributes respectively.
- In the context of equation definitions, scalars, parameters and variables may appear freely in indexed operations. For more on equation definitions, see section Defining Equations.

Sometimes it is necessary to restrict the domain of indexed operations. This may be done with the same techniques as in indexed assignments, see section Restricting the Domain in Assignments for details. See also section Conditional Indexed Operations for more details on conditions in the context of indexed operations.

We now turn to two additional capabilities that are available in GAMS to add power and flexibility to expression calculations: functions and extended range arithmetic.

4.26.4 Functions

Functions play an important role in the GAMS language, especially for nonlinear models. Like other programming languages, GAMS provides a number of built-in or intrinsic functions. GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and quite sophisticated or specialized functions. There is a trade-off between satisfying these requests and avoiding a complexity not needed by most users. The GAMS Function Library Facility provides the means for managing this trade-off, see subsection Extrinsic Functions below.

4.26.4.1 Intrinsic Functions

GAMS provides many functions, ranging from commonly used standard functions like exponentiation, logarithms, and trigonometric functions to utility functions for controlling and querying the running GAMS job or process. The complete list of available functions is given in the following tables: Mathematical Functions, String Manipulation Functions, Logical Functions, Time and Calendar Functions, and GAMS Utility and Performance Functions. For details specific to using these functions in equations, see the section on Expressions in Equation Definitions.

Some of the tables that follow contain an endogenous classification column "End. Classif." that specifies in which models the function may legally appear. In order of least to most restrictive, the choices are *DNLP*, *NLP*, *any*, *none*. See section Classification of Models for details on model types in GAMS. Note well: functions classified as *any* are only permitted with exogenous (constant) arguments.

Functions are typically used in assignment statements and equations. In these cases, they are only evaluated at execution time. Some functions can also be used at compile time, e.g. to compute the factorial or square root of a scalar. Some of the tables below contain a column "Compile Time" indicating which functions can be used at compile time.

A word on the notation in the tables below: for function arguments, lower case indicates that an endogenous variable is allowed. For details on endogenous variables, see section Functions in Equation Definitions. Upper case function arguments indicate that a constant is required. Arguments in square brackets may be omitted: the default values used in such cases are specified in the function description provided.

Mathematical Functions

Mathematical functions may be used as expressions in assignment statements and in equation definitions. We start with some simple examples to illustrate. A list with all mathematical functions available in GAMS is given in Table 3.

Exp(x)

```
a = exp(t);
b = exp(t+2);
```

The GAMS function `exp(x)` returns the exponential e^x of its argument. The assignments above set $a = e^t$ and $b = e^{t+2}$ respectively.

Log(x)

```
z(j) = log(y(j));
```

The GAMS function `log(x)` returns the natural logarithm of its argument. The assignment above evaluates the logarithm once for each element of the controlling domain `j`.

Max(x1,x2,x3,...)

```
x = max(y+2, t, t*t);
```

The function `max` returns the maximum of a set of expressions or terms. In the assignment above, `x` will be assigned the maximum of $y + 2$, t, and $t \cdot t$.

Round(x[,DECPL])

The function `round` rounds its argument `x` to the specified number `DECPL` of places, where positive values of `DECPL` indicating rounding to the right of the decimal point. If the argument `DECPL` is not specified, it defaults to zero, and the function rounds its argument to the nearest integer value. For example,

```
x = round(q);
y = round(q,d);
z = round(12.432,2);
h = round(515.5,-1);
```

In the first assignment `q` is rounded to the nearest integer value, while in the second `q` is rounded to the number of decimals specified by `d`. The result of the third assignment is 12.43, while the final assignment results in a value of 520.

Table 3: ***Mathematical Functions***

Function	Description	End. Classif.	Compile Time
`abs(x)`	**Absolute value** of the argument x.	DNLP	yes

Function	Description	End. Classif.	Compile Time
`arccos(x)`	**Inverse cosine** of the argument x, where x is a real number between -1 and 1 and the output is in radians, see MathWorld	NLP	no
`arcsin(x)`	**Inverse sine** of the argument x, where x is a real number between -1 and 1 and the output is in radians, see MathWorld	NLP	no
`arctan(x)`	**Inverse tangent** of the argument x, where x is a real number and the output is in radians, see MathWorld	NLP	no
`arctan2(y,x)`	**Four-quadrant arctan function** yielding $\arctan(\frac{y}{x})$ which is the angle the vector (x, y) makes with (1,0) in radians.	NLP	no
`beta(x,y)`	**Beta function**: $B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$, see MathWorld	DNLP	no
`betaReg(x,y,z)`	**Regularized beta function**, see MathWorld	NLP	no
`binomial(n,k)`	(Generalized) **Binomial coefficient** for $n > -1$, $-1 < k < n + 1$.	NLP	no
`ceil(x)`	**Ceiling**: returns the smallest integer greater than or equal to x.	DNLP	yes
`centropy(x,y[,Z])`	**Cross-entropy**: $\begin{cases} x \cdot \ln(\frac{x+Z}{y+Z}), & \text{if } x > 0, \\ 0, & \text{if } x = 0, \end{cases}$ where $y > 0$, $Z \geq 0$. Default $Z = 10^{-20}$.	NLP	no
`cos(x)`	**Cosine** of the argument x, where x must be in radians, see MathWorld	NLP	yes
`cosh(x)`	**Hyperbolic Cosine** of x, where x must be in radians, see MathWorld	NLP	no
`cvPower(X,y)`	**Real power**: $\begin{cases} X^y, & \text{if } X > 0, \\ 0, & \text{if } X = 0 \text{ (also for } y \leq 0!). \end{cases}$	NLP	no
`div(dividend,divisor)`	Returns $\frac{\text{dividend}}{\text{divisor}}$, but undefined for divisor $= 0$.	NLP	no
`div0(dividend,divisor)`	Returns $\frac{\text{dividend}}{\text{divisor}}$, but 10^{299} for divisor $= 0$.	NLP	no
`eDist(x1[,x2,x3,...])`	**Euclidean or L-2 Norm**: $\sqrt{x_1^2 + x_2^2 + \ldots}$.	NLP	no
`entropy(x)`	**Entropy**: $\begin{cases} -x \cdot \ln(x), & \text{if } x > 0, \\ 0, & \text{if } x = 0. \end{cases}$	NLP	no

Function	Description	End. Classif.	Compile Time
`errorf(x)`	**Integral of the standard normal distribution** from negative infinity to x: $\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{\frac{-t^2}{2}} dt$.	NLP	no
`execSeed`	Reads or resets the **seed** for the random number generator (RNG). Note that the state of the RNG cannot typically be captured in one seed value; in such cases "reading" the seed involves harvesting a seed from the RNG, resetting the RNG with this seed, and returning the seed.	none	no
`exp(x)`	**Exponential** of x: e^x, see MathWorld	NLP	yes
`fact(N)`	**Factorial** of N, where $N \geq 0$ is an integer. $0! = 1$.	any	yes
`floor(x)`	**Floor**: greatest integer less than or equal to x.	DNLP	yes
`frac(x)`	Returns the **fractional part** of x, s.t. `x = trunc(x) + frac(x)`	DNLP	yes
`gamma(x)`	**Gamma function** : $\Gamma(x) = \int_0^{\infty} t^{x-1} e^{-t} dt$, see MathWorld	NLP	no
`gammaReg(x,a)`	**Lower Incomplete Regularized Gamma function**, see `GammaRegularized[a,0,x]` in MathWorld	NLP	no
`log(x)`	**Natural logarithm**: logarithm base e, see MathWorld	NLP	yes
`logBeta(x,y)`	**Log Beta function**: $\log(B(x,y))$.	NLP	no
`logGamma(x)`	**Log Gamma function** as discussed in MathWorld	NLP	no
`logit(x)`	**Logit Transformation**: $\log(x/(1-x))$, see MathWorld	NLP	yes
`log10(x)`	**Common logarithm**: logarithm base 10, see MathWorld	NLP	yes
`log2(x)`	**Binary logarithm**: logarithm base 2, see MathWorld	NLP	yes
`lseMax(x1[,x2,x3,...])`	Smoothed **Max** via the **L**ogarithm of the **S**um of **E**xponentials: $\ln(e^{x_1} + e^{x_2} + ...)$, see Wikipedia	NLP	no
`lseMaxSc(T,x1[,x2,x3,...])`	**Sc**aled smoothed **Max** via the **L**ogarithm of the **S**um of **E**xponentials: `lseMaxSc(T,x) = lseMax(Tx)/T`, see Wikipedia	NLP	no

Function	Description	End. Classif.	Compile Time
`lseMin(x1[,x2,x3,...])`	Smoothed **Min** via the **L**ogarithm of the **S**um of **E**xponentials: $-\ln(e^{-x_1}+e^{-x_2}+...)$, see Wikipedia	NLP	no
`lseMinSc(T,x1[,x2,x3,...])`	**Sc**aled smoothed **Min** via the **L**ogarithm of the **S**um of **E**xponentials: `lseMinSc(T,x) = lseMin(Tx)/T`, see Wikipedia	NLP	no
`max(x1,x2,x3,...)`	**Maximum** value of the arguments, where the number of arguments may vary.	DNLP	yes
`min(x1,x2,x3,...)`	**Minimum** value of the arguments, where the number of arguments may vary.	DNLP	yes
`mod(x,y)`	**Remainder** of x divided by y.	DNLP	yes
`ncpCM(x,y,Z)`	**Chen-Mangasarian smoothing**: $x - Z \cdot \ln(1 + e^{\frac{x-y}{Z}})$.	NLP	no
`ncpF(x,y[,Z])`	**Fisher-Burmeister smoothing**: $\sqrt{(x^2+y^2+2\cdot Z)}-x-y$, $Z \geq 0$. Default $Z=0$.	NLP	no
`ncpVUpow(r,s[,`μ`])`	**NCP Veelken-Ulbrich** (smoothed min(r,s)): $\begin{cases}\frac{1}{2}(r+s-\lvert r-s\rvert), & \text{if } \lvert r-s\rvert \geq \mu \geq 0,\\ \frac{1}{2}(r+s-\frac{\mu}{8}\cdot(-(\frac{r-s}{\mu})^4+6(\frac{r-s}{\mu})^2+3)), & \text{otherwise.}\end{cases}$ Default $\mu=0$.	NLP	no
`ncpVUsin(r,s[,`μ`])`	**NCP Veelken-Ulbrich** (smoothed min(r,s)): $\begin{cases}\frac{1}{2}(r+s-\lvert r-s\rvert), & \text{if } \lvert r-s\rvert \geq \mu \geq 0,\\ \frac{1}{2}(r+s-(\frac{2\mu}{\pi}\sin(\frac{\pi}{2\mu}+\frac{3\pi}{2})+\mu)), & \text{otherwise.}\end{cases}$ Default $\mu=0$.	NLP	no
`normal(MEAN,STDDEV)`	Generate a **random number** from the **normal distribution** with mean `MEAN` and standard deviation `STDDEV`, see MathWorld	none	no
`pi`	Value of $\pi =$ **3.141593**....	any	yes
`poly(x,A0,A1,A2[,A3,...])`	Returns $p(x)$, where the **polynomial** $p(x) = A_0 + A_1x + A_2x^2 + A_3x^3 + ... + A_{20}x^{20}$. By default $A_3, ..., A_{20} = 0$.	NLP	no
`power(x,Y)`	Returns x^Y, where Y must be an integer. $0^Y = \begin{cases}0, & \text{if } Y>0,\\ 1, & \text{if } Y=0,\\ \text{undefined}, & \text{if } Y<0.\end{cases}$	NLP	no

Function	Description	End. Classif.	Compile Time
`randBinomial(N,P)`	Generate a **random number** from the **binomial distribution**, where N is the number of trials and P the probability of success for each trial, see MathWorld	none	no
`randLinear(LOW,SLOPE,HIGH)`	Generate a **random number** between `LOW` and `HIGH` with **linear distribution**, `SLOPE` must be greater than $\frac{2}{\text{HIGH}-\text{LOW}}$	none	no
`randTriangle(LOW,MID,HIGH)`	Generate a **random number** between `LOW` and `HIGH` with **triangular distribution**, `MID` is the most probable number, see MathWorld	none	no
`round(x[,DECPL])`	**Round** x to `DECPL` decimal places. Default `DECPL=0`	DNLP	yes
`rPower(x,y)`	Returns $\begin{cases} x^y, & \text{if } x > 0, \\ 0, & \text{if } x = 0, y > 0, \\ 1, & \text{if } x = y = 0. \end{cases}$ This function is equivalent to the operation `x**y`, see Standard Arithmetic Operations.	NLP	no
`sigmoid(x)`	**Sigmoid:** $\frac{1}{1+e^{-x}}$, see MathWorld	NLP	no
`sign(x)`	**Sign** of x: $\begin{cases} 1, & \text{if } x > 0, \\ 0, & \text{if } x = 0, \\ -1, & \text{if } x < 0. \end{cases}$	DNLP	yes
`signPower(x,Y)`	**Signed power**: $\text{sign}(x)\lvert x\rvert^Y$ for $Y > 0$.	NLP	no
`sin(x)`	**Sine** of the argument x, where x must be in radians, see MathWorld	NLP	yes
`sinh(x)`	**Hyperbolic sine** of x, where x must be in radians, see MathWorld	NLP	no
`slexp(x[,S])`	**Smooth (linear) exponential**: $\begin{cases} e^x, & \text{if } x \leq S, \\ e^x \cdot (1 + (x - S)), & \text{otherwise,} \end{cases}$ where $S \leq 150$. Default $S = 150$.	NLP	no
`sllog10(x[,S])`	**Smooth (linear) logarithm base 10**: $\begin{cases} \log_{10}(x), & \text{if } x \geq S, \\ \frac{1}{\ln(10)} \cdot (\ln S + \frac{x-S}{S}), & \text{otherwise,} \end{cases}$ where $S \geq 10^{-150}$. Default $S = 10^{-150}$.	NLP	no

Function	Description	End. Classif.	Compile Time
`slrec(x[,S])`	**Smooth (linear) reciprocal**: $\begin{cases}\frac{1}{x}, & \text{if } x \geq S,\\ \frac{1}{S} - \frac{x-S}{S^2}, & \text{otherwise,}\end{cases}$ where $S \geq 10^{-10}$. Default $S = 10^{-10}$.	NLP	no
`sqexp(x[,S])`	**Smooth (quadratic) exponential**: $\begin{cases}e^x, & \text{if } x \leq S,\\ e^x \cdot (1 + (x-S) + \frac{(x-S)^2}{2}), & \text{otherwise,}\end{cases}$ where $S \leq 150$. Default $S = 150$.	NLP	no
`sqlog10(x[,S])`	**Smooth (quadratic) logarithm base 10**: $\begin{cases}\log_{10}(x), & \text{if } x \geq S,\\ \frac{1}{\ln(10)} \cdot (\ln S + \frac{x-S}{S} - \frac{(x-S)^2}{2S^2}), & \text{otherwise,}\end{cases}$ where $S \geq 10^{-150}$. Default $S = 10^{-150}$.	NLP	no
`sqrec(x[,S])`	**Smooth (quadratic) reciprocal**: $\begin{cases}\frac{1}{x}, & \text{if } x \geq S,\\ \frac{1}{S} - \frac{x-S}{S^2} + \frac{(x-S)^2}{S^3}, & \text{otherwise,}\end{cases}$ where $S \geq 10^{-10}$. Default $S = 10^{-10}$.	NLP	no
`sqr(x)`	**Square** of argument x.	NLP	yes
`sqrt(x)`	**Square root** of x, see MathWorld	NLP	yes
`tan(x)`	**Tangent** of the argument x, where x must be in radians, see MathWorld	NLP	yes
`tanh(x)`	**Hyperbolic tangent** of x, where x must be in radians, see MathWorld	NLP	no
`trunc(x)`	**Truncation:** returns the integer part of x, truncating towards zero.	DNLP	yes
`uniform(LOW,HIGH)`	Generates a **random number** from the **uniform distribution** between `LOW` and `HIGH`, see MathWorld	none	no
`uniformInt(LOW,HIGH)`	Generates an **integer random number** from the **discrete uniform distribution** whose outcomes are the integers between `LOW` and `HIGH`, inclusive, see MathWorld	none	no
`vcPower(x,Y)`	Returns x^Y for $x \geq 0$. $0^0 = 1$.	NLP	no

String Manipulation Functions

GAMS provides some string manipulation capability by extending the card and ord operators to work on strings as well as sets. In Table 4 the extended behavior is described. In this context, the functions take

strings and *places* as arguments, and the numeric *places* argument must be a constant. This behavior only applies to execution-time usage of these functions.

Table 4: ***String Manipulation Functions***

Function	Description
`card(STRING)`	Returns the **number of characters** in the `string`.
`ord(STRING[,PLACE])`	Returns the **ASCII code number of a character** in a position in a `string`. The optional `place` entry defaults to 1 if omitted and identifies the character position within the `string` to be examined (1 for the first, 2 for the second, etc.)

There are four types or sources of strings in this context. A string may be a string literal, i.e. a concatenation of letters and blanks as in `"drink it"` in the example that follows. It may be the symbol text (aka the explanatory text) associated with any symbol. Or it may be the labels or text associated with the elements of a set. The following table gives an overview:

Table 5: ***String Types***

Notation	Description	Comments
`characters`	A concatenation of characters and blanks: all legal GAMS characters are allowed, see section Characters for details.	The characters must be surrounded by double or single quotes.
`symbol_name.ts`	The string is the explanatory text associated with a symbol name.	If the explanatory text is missing, the value of `card` is 0.
`set_name.tl`	The string is the label for a set element.	This string type may be used only when `set_name` is part of the controlling domain.
`set_name.te`	The string is the explanatory text associated with a set element.	This string type may be used only when `set_name` is part of the controlling domain. If the explanatory text is missing, the value of `card` is 0.

The string types are illustrated in the following example. Note that the directive in the first line activates the end-of-line comment option, see eolCom.

```
$oneolcom
variable z 'any symbol can have explanatory text';
set teas "teas available to order"  / black "English Breakfast",  green, peppermint /;
scalar p;

p = card("drink it");       !! result: p=8
p = card(z.ts);             !! result: p=36
p = card(teas.ts);          !! result: p=23

loop{teas,
    p = card(teas.tl)       !! length of set element label from the set "teas": "teas" is the controll
    display "length of set element label", p;
    p = card(teas.te)       !! length of set element explanatory text: "teas" is the controlling set
    display "length of set element explanatory text", p;
};
```

Note that the strings `teas.tl` and `teas.te` are used in the context of a `loop` statement (see section The Loop Statement). This is a typical usage pattern.

Logical Functions

Logical functions may be used as expressions in assignment statements as in the following example.

```
x = ifthen(tt=2, 3, 4+y);
```

Here $x = 3$ if $tt = 2$, otherwise $x = 4 + y$.

The logical functions available in GAMS are given in Table 6. Note that logical functions may also be used in conditions and logic equations, see sections Logical Conditions and Logic Equations respectively. Most of the logical functions can also be indicated using the familiar operator notation, e.g. (x and y), (x >= y), etc. In such cases, the operator notation is allowed at compile time. The last column in Table 6 indicates if a logical function can be used at compile time (yes) or if only its alternative notation is available (alt). Further, note that the inputs and outputs of these functions are often logical/Boolean values, so GAMS does implicit conversions as necessary. As expected, false becomes 0 and true becomes 1 when converting logical values to numeric, and 0 becomes false and nonzero becomes true when numeric values are converted to logicals. For details on behavior when the inputs are special values, see sections Extended Range Arithmetic and Acronym Usage, but note that EPS, +INF, -INF, and acronyms become true when converted to logicals.

Table 6: ***Logical Functions***

Function	Alternative Notation 1	Alternative Notation 2	Description	Return Values	End. Classif.	Compile Time
bool_and(x,y)	x and y		**Boolean** AND	Returns true iff both x and y are true	DNLP	alt
bool_eqv(x,y)	x eqv y	x <=> y	**Boolean equivalence**	Returns false iff *exactly* one argument is false	DNLP	alt
bool_imp(x,y)	x imp y	x -> y	**Boolean implication**	Returns true iff x is false or y is true	DNLP	alt
bool_not(x)	not x		**Boolean** NOT	Returns true iff x is false	DNLP	alt
bool_or(x,y)	x or y		**Boolean** OR	Returns true iff x is true or y is true	DNLP	alt
bool_xor(x,y)	x xor y		**Boolean** XOR	Returns true iff *exactly* one argument is false	DNLP	alt

Function	Alternative Notation 1	Alternative Notation 2	Description	Return Values	End. Classif.	Compile Time
`ifThen(cond,iftrue,else)`			**Condition**	If the logical condition (first argument) is true, the function returns `iftrue`, else it returns `else`. See example above.	DNLP	yes
`rel_eq(x,y)`	`x eq y`	`x = y`	**Numeric Relation: Equal**	Returns true iff $x = y$	DNLP	alt
`rel_ge(x,y)`	`x ge y`	`x >= y`	**Numeric Relation: Greater Equal**	Returns true iff $x \geq y$	DNLP	alt
`rel_gt(x,y)`	`x gt y`	`x > y`	**Numeric Relation: Greater Than**	Returns true iff $x > y$	DNLP	alt
`rel_le(x,y)`	`x le y`	`x <= y`	**Numeric Relation: Less Equal**	Returns true iff $x \leq y$	DNLP	alt
`rel_lt(x,y)`	`x lt y`	`x < y`	**Numeric Relation: Less Than**	Returns true iff $x < y$	DNLP	alt
`rel_ne(x,y)`	`x ne y`	`x <> y`	**Numeric Relation: Not Equal**	Returns true iff $x \neq y$	DNLP	alt

Time and Calendar Functions

GAMS offers several functions that relate to time and dates. The fundamental measurement of time in GAMS is the serial day number beginning with January 1, 1900. This serial day number is a real number whose integer part contains a unique number for each day and whose fractional part contains information about hours, minutes, and seconds. We can think of the serial day number as being a `date.time` pair. The day information extracted from serial day numbers is based on the Gregorian calendar.

Note

In all functions given in Table 7, *serial day 1* is *January 1, 1900*.

All of the functions in Table 7 can be used at compile time.

Table 7: ***Time and Calendar Functions***

Function	Description	End. Classif.
gday(SDAY)	Returns **Gregorian day** from a serial day number date.time.	any
gdow(SDAY)	Returns **Gregorian day of week** from a serial day number date.time.	any
ghour(SDAY)	Returns **Gregorian hour** of day from a serial day number date.time.	any
gleap(SDAY)	Returns 1 if the year that corresponds to a serial day number date.time, is a **leap year**, else returns 0.	any
gmillisec(SDAY)	Returns **Gregorian milli second** from a serial day number date.time.	any
gminute(SDAY)	Returns **Gregorian minute** of hour from a serial day number date.time.	any
gmonth(SDAY)	Returns **Gregorian month** from a serial day number date.time.	any
gsecond(SDAY)	Returns **Gregorian second** of minute from a serial day number date.time.	any
gyear(SDAY)	Returns **Gregorian year** from a serial day number date.time.	any
jdate(YEAR,MONTH,DAY)	Returns a **serial day number**.	any
jnow	Returns the **current time** as a serial day number.	none
jstart	Returns the **time of the start of the GAMS job** as a serial day number.	none
jtime(HOUR,MIN,SEC)	Returns **fraction of a day** that corresponds to hour, minute and second.	any

GAMS Utility and Performance Functions

GAMS provides several functions that may be used to get (and in some cases set) GAMS system information, for example:

```
scalar o;
o = heapLimit;
heapLimit = 1024;
```

Table 8: ***GAMS Utility and Performance Functions***

Function	Description	End. Classif.	Compile Time
embeddedHandle	Returns the handle for the last embedded code section executed, see section Syntax in chapter Embedded Code Facility for details.	none	no
errorLevel	**Return code** of the most recently used command.	none	yes
execError	Get or set the **number of execution errors**.	none	no
gamsRelease	Returns the **release number** of the running GAMS system, for example 24.7.	none	yes

Function	Description	End. Classif.	Compile Time
`gamsVersion`	Returns the **version number** of the running GAMS system, for example 247.	none	yes
`handleCollect(HANDLE)`	Tests if the solve of the model instance identified by the calling argument `HANDLE` is done: if so, it loads the solution into GAMS. For details, see Table 1 in section Grid Computing.	none	no
`handleDelete(HANDLE)`	Deletes the model instance identified by `HANDLE`. For details, see Table 1 in section Grid Computing.	none	no
`handleStatus(HANDLE)`	Tests if the solve of the model instance identified by `HANDLE` is done: if so, it loads the solution into a GDX file. For details, see Table 1 in section Grid Computing.	none	no
`handleSubmit(HANDLE)`	Resubmits the model instance identified by `HANDLE` for solution. For details, see Table 1 in section Grid Computing.	none	no
`heapFree`	Get the amount of **free memory** in the heap in MB, i.e. memory allocated to the process and available for future use by GAMS.	none	no
`heapLimit`	Get or set the current **heap limit** (maximum allowable dynamic memory usage) in Mb.	none	no
`heapSize`	Get the current **heap size** in Mb.	none	no
`jobHandle`	Returns the **process ID** (`PID`) of the last job started.	none	yes
`jobKill(PID)`	Sends a **kill signal** to the job with process ID `PID`. The return value is 1 if this was successful, 0 otherwise.	none	yes
`jobStatus(PID)`	Get the **status** of the job with process ID `PID`. Possible return values are: 0: error (input is not a valid `PID` or access is denied) 1: process is still running 2: process is finished with return code which could be accessed by errorlevel 3: process not running anymore or was never running, no return code available	none	yes
`jobTerminate(PID)`	Sends an **interrupt signal** to the job with process ID `PID`. The return value is 1 if this was successful, 0 otherwise.	none	yes

Function	Description	End. Classif.	Compile Time
`licenseLevel`	Get an indicator for the **type of license**: 0: demo license, limited to small models 1: full unlimited developer license 2: run time license, no new variables or equations can be introduced besides those inherited from a work file 3: application license, only works with a specific work file which is locked to the license file 5: community license, limited to somewhat bigger models compared to demo license.	any	no
`licenseStatus`	Returns non-zero if a **license error** has occurred.	any	no
`mapVal(x)`	Returns an integer value that indicates what special value (if any) is stored in the input x. Possible results: 0: x is not a special value 4: x is `UNDF` (undefined) 5: x is `NA` (not available) 6: x is `INF` (∞) 7: x is `-INF` ($-\infty$) 8: x is `EPS` >8: x is an acronym	any	no
`maxExecError`	Get or set the **maximum number of execution errors**. See also maxExecError.	none	no
`numCores`	Get the number of logical CPU cores in the system	any	yes
`platformCode`	Returns an integer indicating the current platform. Possible return values can be seen here.	any	yes
`readyCollect(HANDLES[,MAXWAIT])`	Waits until a model solution is ready to be collected. For details, see Table 1 in section Grid Computing.	none	no
`sleep(SEC)`	Pause or **sleep** execution for `SEC` seconds.	none	yes
`timeClose`	Returns the **accumulated closedown time**, i.e. the time GAMS uses to save its state to disk prior to a solve.	none	no
`timeComp`	Returns the **compilation time** in seconds.	none	no
`timeElapsed`	Returns the **elapsed time** since the start of a GAMS run in seconds.	none	no
`timeExec`	Returns the **execution time** in seconds.	none	no

Function	Description	End. Classif.	Compile Time
timeStart	Returns the **accumulated startup time**, i.e. the time GAMS uses to restore its state from disk after a solve.	none	no

4.26.4.2 Extrinsic Functions

Using the GAMS Function Library Facility, functions may be imported from an external library into a GAMS model. Apart from the import syntax, the imported functions may be used in the same way as intrinsic functions. In particular, they may be used in equation definitions. Some function libraries are included with the standard GAMS software distribution, but GAMS users can also create their own libraries using an open programming interface. The GAMS Test Library instances **[TRILIB01]**, **[TRILIB02]**, **[TRILIB03]**, and **[CPPLIB00]** are simple examples (in the programming languages C, Delphi, Fortran and C++, respectively) that come with every GAMS system.

For details on using and creating extrinsic function libraries, and on the extrinsic function libraries that are provided with GAMS, see the chapter on Extrinsic Functions.

4.26.4.3 Function Suffixes

Up to this point, this section has described and discussed functions without mentioning their derivatives. These derivatives are very important, though: most of the nonlinear solvers integrated with GAMS will require first derivatives to solve models, and many will also use or require second derivatives as well. It is sometimes useful (e.g. when testing an extrinsic function) to evaluate the derivatives of GAMS functions instead of the functions themselves. This can be done via function suffixes.

Function suffixes can specify the evaluation of a gradient (i.e. first derivative), a Hessian (i.e. second derivative), a minimum or maximum value of the function over a given range, or the minimum or maximum value of the gradient over a given range. A full list of function suffixes is given in Table 9. For functions whose arguments are constant, the derivatives are zero, so typically `func` is a mathematical function listed in Table 3 above. Note that function suffixes are *not* defined for functions without arguments (for example, `pi`), so specifying something like `pi.grad` results in a compilation error.

Table 9: ***Function Suffixes***

Function Suffix	Description
func.value(x)	Value of the function `func` at `x`, equals `func(x)`.
func.grad(x)	Value of the gradient of the function `func` at `x`.
func.gradn(x)	Value of the gradient of the function `func` at `x`, computed numerically.
func.hess(x)	Value of the Hessian of the function `func` at `x`.
func.hessn(x)	Value of the Hessian of the function `func` at `x`, computed numerically.
func.low(x1:x2)	Lower bound of the function `func(x)` on the interval `[x1,x2]`.
func.high(x1:x2)	Upper bound of the function `func(x)` on the interval `[x1,x2]`.
func.gradL(x1:x2)	Lower bound of the gradient of the function `func(x)` on the interval `[x1,x2]`.
func.gradH(x1:x2)	Upper bound of the gradient of the function `func(x)` on the interval `[x1,x2]`.

Consider the following simple example:

```
scalars g, l, h, gl, gh;

g  = sin.grad(0);
l  = sin.low(pi/3:pi/2);
h  = sin.high(pi/3:pi/2);
gl = sin.gradL(pi/3:pi/2);
gh = sin.gradH(pi/3:pi/2);

display g, l, h, gl, gh;
```

For univariate functions like sine or cosine, there is no need to specify a variable index when evaluating derivatives. For multivariate functions, the default is to take partial derivatives w.r.t. the first variable. To specify other variables, the colon syntax in the example below is used.

```
scalars
  x   / 1 /
  ylo / -1 /
  yup / 2 /
  e0, e1, elo, ehi
  e_1, e_2
  e_11, e_22, e_21
  ;
e0 = edist(x,ylo);
e1 = edist(x,yup);
elo = edist.low (x,ylo:x,yup);
ehi = edist.high(x,ylo:x,yup);

e_1 = edist.grad(x,yup);
* e_1 = edist.grad(1:x,yup);  same as above
e_2 = edist.grad(2:x,yup);

e_11 = edist.hess(x,yup);
* e_11 = edist.hess(1:1:x,yup);  same as above
e_22 = edist.hess(2:2:x,yup);
e_21 = edist.hess(2:1:x,yup);

display x, ylo, yup, e0, e1, elo, ehi, e_1, e_2, e_11, e_22, e_21;
```

For more examples, see model **[FUNCS4]** in the GAMS Test Library.

Note

- The function suffixes `value`, `grad`, `gradn`, `hess` and `hessn` are also defined for extrinsic functions. When implementing an extrinsic function, be aware that missing derivatives will be computed numerically: see model **[TRILIB01]** for an example.

To compute derivatives numerically, GAMS uses finite difference approximations. This computation can be controlled with two options: the FDOpt option controls which variant of the finite difference method is used, while the FDDelta option controls the step size.

4.26.5 Extended Range Arithmetic and Error Handling

GAMS uses an *extended range* arithmetic to handle missing data, the results of undefined operations, infinite values, and zeros that are stored explicitly. The *special values* used in this arithmetic are listed and described in Table 10 below, along with the value of the mapVal function that corresponds to these values. We can think of special values as any value for which mapVal does not return 0.

Special value	Description	mapVal
INF	Plus infinity. Similar to IEEE plus infinity. Behaves in the expected way in computations, so that e.g. min(x,INF) = x unless x is also special.	6
-INF	Minus infinity. Similar to IEEE minus infinity. Behaves in the expected way in computations, so that e.g. max(x,INF) = x unless x is also special.	7
NA	Not available - used to indicate missing data. It is a sticky value: e.g. max(x,NA) = NA even for x = INF	5
UNDF	Undefined - indicates the result of an undefined or illegal operation. Similar to IEEE NaN. A user cannot directly set a value to UNDF unless the dollar control option onUNDF is active. For details, see the chapter on Dollar Control Options.	4
EPS	A *stored* zero value. If the dollar control option onEPS is active, zeros in a parameter or table statement are treated as EPS. For details on dollar control options, see chapter Dollar Control Options.	8

Table 10: Special Values for Extended Range Arithmetic

Attention

Avoid creating or using numbers with very large (1.0e299 or above) or very small (1.0e-299 or below) magnitudes. Large numbers may be treated by GAMS as undefined (UNDF) or other special values, leading to unpredictable and unusable results. Always use INF (or -INF) explicitly for arbitrarily large (or small) numbers.

GAMS has defined the results of all arithmetic operations and all function evaluations that use these special values. The behavior is designed to both maximize utility and minimize surprise, and is illustrated in the library model [**CRAZY**]. For example: 1+INF evaluates to INF, 1-EPS to 1, NA $*$ 2 to NA, and EPS$*$INF to UNDF.

The following table shows a selection of results for exponentiation and division for a variety of input parameters including NA and INF.

a	b	a**b	power(a,b)	a/b
2	2	4	4	1
-2	2	UNDF	4	-1
2	2.1	4.28	UNDF	.952
NA	2.5	NA	NA	NA
3	0	1	1	UNDF
INF	2	INF	INF	INF
2	INF	UNDF	UNDF	0

Table 11: Extended Range Arithmetic in Exponentiation and Division

Note that most extended range identifiers may be used in assignment statements, as illustrated below.

```
a(row,'col-3') =  NA;
a(row,'col-4') =  INF;
a(row,'col-5') = -INF;
```

The values most often used in assignments are NA in incomplete tables and INF for variable bounds.

The special value EPS is numerically equal to zero, so when used in the context of numerical comparisons, it behaves as zero. For example, the logical expressions x > 0 and x > EPS have the same result. However, EPS is a *stored* zero, so setting a parameter to zero (e.g. a(row,col)=0) results in no values being stored for a, while setting a parameter to EPS (e.g. a(row,col)=EPS) results in the value EPS being stored for every (row,col) tuple. This is sometimes useful, especially if a is used later in loops (see Programming Flow Control Features) or dollar conditions (see Conditional Expressions, Assignments and Equations). For example, consider the statement

```
a(row,col)$[a(row,col)] = INF ;
```

In the case where a was originally assigned to be zero, the dollar-condition is always false so no infinities are assigned. If a was originally assigned the value EPS, the dollar-condition is always true and every tuple in a will be set to infinity.

When an attempted arithmetic operation is illegal or has undefined results because of the value of arguments (division by zero is the normal example), an error is reported and the result is set to undefined (UNDF). The error is marked on the output file with a row of four asterisks '****' making this sequence a good search target in the editor. GAMS will also report the line number of the offending statement and give as much detail as possible about the cause. From there on, the resulting UNDF is treated as a proper data value and does not trigger additional error messages. For more on error messages, see chapter GAMS Output.

Note

GAMS will not solve a model if an error has been detected, and it will terminate with an error condition.

It is therefore wise to anticipate and avoid operations like division by zero that will cause errors. This is most easily done with the dollar control, and is discussed in section The Dollar Condition.

4.26.6 Predefined Symbols

GAMS offers several *predefined* symbols including: sameAs, diag, sortedUels, solvers, licenseCodes, and solverCapabilities. With the exception of sameAs, diag, and sortedUels these predefined symbols are for very special (and mostly internal) purposes, but the way they work is identical.

These symbols can be used in a GAMS program without being declared but work in an idiosyncratic way. For example, the complete program

```
display solvers, sameAs;
```

compiles and executes without error and produces the following result in the listing file:

```
----      1 SET Solvers  Available Solvers

                                                    ( EMPTY )

----      1 SET SameAs  Set Element Comparison Without Checking

                                                    ( EMPTY )
```

However, if labels have been declared (in any set) that corresponds to a solver name, then set solvers will contain this label. Consider the following example:

```
Set titles / Lord, Baron, Duke /;
Set greeks / Euclid, Pythagoras, Antigone /;
display solvers, sameAs;
```

Note that `Antigone` and `Baron` are the names of solvers in GAMS and therefore they will be displayed as a member of the set `solvers`. Also note that the set `sameAs` contains the diagonal elements for both sets:

```
----      3 SET Solvers  Available Solvers
Baron    ,    Antigone

----      3 SET SameAs  Set Element Comparison Without Checking
                 Lord       Baron        Duke      Euclid  Pythagoras    Antigone
Lord              YES
Baron                         YES
Duke                                     YES
Euclid                                               YES
Pythagoras                                                        YES
Antigone                                                                       YES
```

User defined symbols can have the same name as a predefined symbol. In this case the user symbol *hides* the predefined symbols as demonstrated in the following example:

```
Set diag / 1*3 /;
$if setType diag $log diag is a set
$if preType diag $log diag is a predefined type
```

The log will only contain the line `diag is a set`. The test `$if preType diag` fails.

The list of all predefined symbols can be retrieved by declaring and displaying a set using some system data `set pre /system.predefinedSymbols/; display pre;` which results in

```
----      1 SET pre
SameAs                ,    Diag                  ,    Solvers
LicenseCodes          ,    SolverCapabilities    ,    SortedUels
```

4.26.7 Summary

GAMS provides powerful facilities for data manipulation with parallel assignment statements, built-in functions and extended range arithmetic.

4.27 Data Entry: Parameters, Scalars and Tables

4.27.1 Introduction

Data handling is of crucial importance in all modeling applications. The quality of the numbers and the intelligence with which they are used is likely to be at least as important as the logic of the model in determining if an application is successful or not. GAMS has been designed to have a complete set of facilities for entering information, manipulating it and reporting on the results. In this chapter we will concentrate on data entry. Chapter Data Manipulations with Parameters introduces and discusses data manipulations. For details on reporting, see chapters GAMS Output, The Display Statement, The Put Writing Facility, and GAMS Data eXchange (GDX).

One very important principle will motivate all our discussions on data:

Note

Data should be entered in its most basic form and each data item should be entered only once.

There are two reasons for adopting this principle. Numbers are almost certain to change, and when they do we want to be able to make the process of changing them as easy and safe as possible. We also want to make our model easy for others to read and understand. Keeping the amount of data as small as possible will certainly help. All the data transformations are shown explicitly in the GAMS representation, which makes it possible to reproduce the results of a study and shows the reader all the assumptions made during data manipulation. Another advantage is that everything needed to run or change the model is included in one program that can easily be moved from place to place or from one machine to another.

This chapter deals with the data type *parameter*. For other data types, see section Data Types and Definitions. Data for *parameters* can be entered in three basic formats: scalar, list oriented, or tables of two or more dimensions. For each of these formats, GAMS offers a separate keyword:

Keyword	Description
Scalar	Single (scalar) data entry.
Parameter	List oriented data, defined over one or more sets.
Table	Table oriented data, must involve two or more dimensions.

Table 1: Parameters, Scalars and Tables

Note that the term *parameter* is used in two ways: as data type and as keyword, so one could also see `scalars` and `tables` as special formats of *parameters*. Each of the data input formats will be introduced and discussed in the following sections. At the end of the chapter the special data type acronym is introduced.

Note

- By default, parameters in all input formats may only be initialized once, thereafter data must be modified with assignment statements. This can be changed using the dollar control option $onMulti.
- This chapters explains the complete syntax to declare *parameters* which includes the optional initialization. So, while it is possible to initialize the data at declaration, often the data is read from other sources like databases or spreadsheets. More information about this can be found in the chapter Data Exchange with Other Applications.

4.27.2 Scalars

The `scalar` statement is used to declare and (optionally) initialize a GAMS parameter of dimensionality zero. This means that there are no associated sets, so there is exactly one number associated with the parameter.

4.27.2.1 The Syntax

In general, the syntax for a `scalar` declaration in GAMS is as follows:

```
scalar[s] scalar_name [text] [/numerical_value/]
       {  scalar_name [text] [/numerical_value/]} ;
```

The keyword `scalar[s]` indicates that this is a `scalar` statement and `scalar_name` is the internal name of the scalar in GAMS, it is an identifier. The optional explanatory text is used to describe the scalar and the optional `numerical_value` is assigned to be the value of `scalar_name`. `Numerical_value` can be given as fixed number or as constant evaluation. Alternatively, the special data type `acronym` may be used as value. For details on acronyms, see section Acronyms.

Note that more than one scalar may be declared in one scalar statement. The entries have to be separated by commas or by end of line. For advice on explanatory text and how to choose a `scalar_name`, see the tutorial Good Coding Practices.

Note that scalars may be declared but not initialized in the scalar statement. A value can also be assigned later as illustrated in the example that follows.

4.27.2.2 An Illustrative Example

An example of a `scalar` definition in GAMS is shown below.

```
Scalar
    rho  "discount rate"                            / .15 /
    irr  "internal rate of return"
    life "financial lifetime of productive units"  / 20  /;
```

The statement above initializes `rho` and `life`, but not `irr`. Later on another scalar statement can be used to initialize `irr` or an assignment statement could be used to provide the value:

```
irr  =  0.07;
```

For more on scalar assignments and parameter assignments in general, see section Data Entry by Assignment.

4.27.3 Parameters

The parameter format is used to enter list oriented data which can be indexed over one or several sets.

4.27.3.1 The Syntax

In general, the syntax for a `parameter` declaration in GAMS is as follows:

```
parameter[s] param_name[(index_list)] [text] [/ element [=] numerical_value
                                               {,element [=] numerical_value} /]
           {,param_name[(index_list)] [text] [/ element [=] numerical_value
                                               {,element [=] numerical_value} /]} ;
```

The keyword `parameter[s]` indicates that this is a parameter statement and `param_name` is the internal name of the parameter in GAMS, it is an identifier. A parameter may be defined over one or more sets that may be specified in the `index_list`. Note that the specification of the index list in the declaration is optional. However, mostly it is advisable to specify it for reasons of clarity and to enable domain checking. For more on domain checking, see section Domain Checking. The optional explanatory text is used to describe the parameter.

Parameter initialization requires a list of data elements, each consisting of a label or label-tuple and a value. `Element` is an element of the defining set or - if there is more than one defining set - a combination of the elements of the defining sets. The referenced set elements must belong to the set that the parameter is indexed over. Finally, `numerical_value` is the value assigned to the record defined by the set element or element tuple. It can be given as fixed number or as constant evaluation. Alternatively, the special data type `acronym` may be entered as value. For details on acronyms, see section Acronyms.

Note

The default value of a `parameter` is 0.

Slashes must be used at the beginning and end of the list, and commas must be used if several data elements are listed in one line. An equals sign or a blank separates the label tuple from its associated value. A parameter can be defined in a similar syntax to that used for a set. For advice on explanatory text and how to choose a parameter name, see the tutorial Good Coding Practices.

Note

Several parameters may be declared in one parameter statement.

4.27.3.2 Illustrative Examples

The following example illustrates the parameter statement. It is adapted from [**MEXSS**]. We also show the set definitions because they make the example clearer. For more on sets definitions, see chapter Set Definition.

```
Set i     "steel plants"   / hylsa      monterrey
                             hylsap     puebla  /
    j     "markets"        / mexico-df, monterrey, guadalaja /;

Parameter
    dd(j) "distribution of demand"
                         /  mexico-df   55,
                            guadalaja   15 /;
```

The index specification for the parameter `dd` means that there will be a vector of data associated with it, one number corresponding to every member of the set `j`. The numbers are specified along with the declaration in a format very reminiscent of the way we specify sets: in this simple case a label followed by a blank separator and then a value. Any of the legal number entry formats are allowable for the value. For details on number formats in GAMS, see subsection Numbers. The default data value is zero. Since `monterrey` has been left out of the data list, the value associated with `dd('monterrey')` is zero. As with sets, commas are optional at end of line.

We may also list several data elements on a line, separated by commas as in the following example:

```
Parameter
    a(i)  / seattle  =  350,  san-diego  =  600 /
    b(i)  / seattle    2000,  san-diego    4500 /;
```

If a parameter is defined over a set and all elements of the set are assigned the same value, then the following notation may be used as a shortcut:

```
parameter param_name[(set_name)]  [text]  /(#|set.)set_name numerical_value/;
```

Here `set` is a reserved word and `set_name` is the name of the set as it has been declared in a previous set declaration statement. Instead of `set`. one could also use the `#` sign. The following artificial example illustrates this notation:

```
Set        j      /j1, j2/;
Parameter  hh(j)  /set.j 10/
           gg     /#j    10/;
```

This resolves in `hh('j1') = hh('j2') = gg('j1') = gg('j2') = 10`.

Note

By default it is not possible to define an empty parameter at declaration. This may be changed using the dollar control option $onEmpty, as shown in the following example:

```
Set i     / seattle,  san-diego /;
$onEmpty
Parameter
    a(i)  /  /;
```

That initializes a('seattle') and a('san-diego') to 0. So it is not the same as this:

```
Set i     / seattle,  san-diego /;
Parameter
    a(i);
```

Here, a is declared, but not initialized (so, it is not defined yet) and one would get an error when trying to read it.

4.27.3.3 Parameter Data for Higher Dimensions

A parameter may have several dimensions. For the current maximum number of permitted dimensions, see Dimensions. The list oriented data initialization through the parameter statement can be easily extended to data of higher dimensionality. The label that appears on each line in the one-dimensional case is replaced by a label tuple for higher dimensions. The elements in the n-tuple are separated by dots (.) just like in the case of multi-dimensional sets.

The following example illustrates the use of parameter data for higher dimensions:

```
Parameter
    salaries(employee,manager,department)
        / anderson   .murphy   .toy          = 6000
          hendry     .smith    .toy          = 9000
          hoffman    .morgan   .cosmetics    = 8000 /;
```

All the mechanisms using asterisks and parenthesized lists that we introduced in our discussion of sets are available here as well. For details see section Multi-Dimensional Sets. Below is an artificial example, in which a very small fraction of the total data points are initialized. GAMS will mark an error if the same label combination (or label-tuple) appears more than once in a data list.

```
Set row / row1*row10 /
    col / col1*col10 /;
Parameter
    a(row, col)
        /  (row1,row4) . col2*col7    12
            row10      . col10        17
            row1*row7  . col10        33 /;
```

In this example the twelve elements row1.col2 to row1.col7 and row4.col2 to row4.col7 are all initialized at 12, the single element row10.col10 at 17, and the seven elements rows1.col10 to row7.col10 at 33. The other 80 elements (out of a total of 100) remain at their default value, which is 0. This example shows the ability of GAMS to provide a concise initialization or definition for a sparse data structure.

4.27.4 Tables

Tabular data can be declared and initialized in GAMS using a table statement. For two and higher-dimensional parameters this provides an easier and more concise method of data entry than the list based approach, since - at least in smaller tables - each label appears only once.

4.27.4.1 The Syntax

In general, the syntax for a `table` declaration in GAMS is as follows:

```
table table_name[(index_list)]  [text] [EOL
                element                { element }     EOL
  element    numerical_value       { numerical_value} EOL
 {element    numerical_value       { numerical_value} EOL}] ;
```

The keyword `table` indicates that this is a `table` declaration and `table_name` is the internal name of the table in GAMS, it is an identifier. The name of the parameter can be followed by the `index_list`. In the `index_list` the sets are specified over which the table is defined. Note that the specification of the index list in the declaration is optional. However, mostly it is advisable to specify it for reasons of clarity and to enable domain checking. For more on domain checking, see section Domain Checking. The optional explanatory text is used to describe the table, followed by `EOL` which means "end of line", a line break. `Element` is an element of one of the driving sets. More details follow below. `Numerical_value` is the value of the entry associated with the corresponding element combination. It can be given as fixed number or as constant evaluation. Alternatively, the special data type `acronym` may be used as value. For details on acronyms, see section Acronyms. For advice on explanatory text and how to choose a `table_name`, see the tutorial Good Coding Practices.

Attention

> By default, the table statement is the only statement in the GAMS language that is not free format. This may be changed using the dollar control option $onDelim.

The following rules apply:

- The relative positions of all entries in a table are significant. This is the only statement where end of line (EOL) has meaning. The character positions of the numeric table entries must overlap the character positions of the column headings.
- The column section has to fit on one line.
- The sequence of values forming a row must be on the same line.
- The element definition of a row can span more than one line.
- A specific column can appear only once in the entire table.

The rules for building simple tables are straightforward. The components of the header line are

```
keyword - identifier - index_list - text
```

Note that the `index_list` and the `text` are optional. Labels are used on the top and the left to map out a rectangular grid that contains the data values. The order of labels is unimportant, but if domain checking has been specified (i.e. the `index_list` has been given in the first line of the table declaration) each label must match one in the associated set. Labels must not be repeated, but can be left out if the corresponding numbers are all zero or not needed. At least one blank must separate all labels and data entries. Blank entries imply that the default value (zero) will be associated with that label combination.

Note

- Tables must have at least two dimensions. For the current maximum number of permitted dimensions, see Dimensions.
- The `table` statement can also be terminated with a `;` in the first line, so without entering data. In this case the parameter `table_name` would be declared but not initialized.
- In contrast to the `set`, `scalar`, and `parameter` statements, only one identifier may be initialized in a `table` statement, though multiple ones could be declared.

4.27.4.2 An Illustrative Example

In the following example a simple table is presented. It is adapted from [**KORPET**], the relevant set definitions are also given.

```
Set i   "plants"
        / inchon, ulsan, yosu /
    m   "productive units"
        / atmos-dist   "atmospheric distillation unit"
          steam-cr     "steam cracker"
          aromatics    "aromatics unit"
          hydrodeal    "hydrodealkylator"  /;

Table ka(m,i) "initial cap. of productive units (100 tons per yr)"
                     inchon      ulsan      yosu
    atmos-dist         3702      12910      9875
    steam-cr                       517      1207
    aromatics                      181       148
    hydrodeal                      180
;
```

In this example the row labels are drawn from the set `m` and those on the column from the set `i`. Note that the data for each row is aligned under the corresponding column headings. Entries that are not specified are assigned the default value zero.

Note

If there is any uncertainty about which column a number belongs to, GAMS will protest with an error message and mark the ambiguous entry.

Attention

Special care has to be taken, if tabs are used. The GAMS command line option TabIn controls the tab spacing. Note that this spacing might be different form the spacing that the editor is showing, hence the visible alignment might be different from the alignment that GAMS is actually using.

4.27.4.3 Continued Tables

If a table has too many columns to fit nicely on a single line, then the columns that don't fit may be continued on additional lines. We use the same example to illustrate:

```
Table ka(m,i) "initial cap. of productive units (100 tons per yr)"
                    inchon    ulsan
    atmos-dist      3702      12910
    steam-cr                    517
    aromatics                   181
    hydrodeal                   180

        +           yosu
    atmos-dist      9875
    steam-cr        1207
    aromatics        148
;
```

The crucial item is the plus `'+'` sign above the row labels and to the left of the column labels in the continued part of the table. The row labels have been duplicated, except that `hydroreal` has been left out, since it does not have any associated data. Tables may be continued as many times as necessary.

4.27.4.4 Tables with more than Two Dimensions

Tables may have more than two dimensions. For the current maximum number of permitted dimensions, see Dimensions. As usual, dots are used to separate adjacent labels and may be used in the row or column position. The label on the left of the row corresponds to the first set in the index list, and that on the right of each column header to the last. Obviously, there must be the same number of labels associated with each number in the table, as there are sets in the index list.

The best layout depends on the size of the defining sets and the amount of data. It should provide the most intuitively satisfactory way of organizing and inspecting the data. For most people it is easier to look down a column of numbers than across a row. However, putting extra labels on the row has the advantage of greater density of information.

The following example, adapted from [**MARCO**], illustrates the use of tables with more than two dimensions.

```
Set ci   "commodities :   intermediate"
         / naphtha    "naphtha"
           dist       "distillate"
           gas-oil    "gas-oil"  /
    cr   "commodities :   crude oils"
         / mid-c       "mid-continent"
           w-tex       "west-texas"  /
    q    "attributes of intermediate products"
         / density, sulfur /;

Table attrib(ci, cr, q) "blending attributes"
                          density    sulfur
    naphtha. mid-c          272        .283
    naphtha. w-tex          272       1.48
    dist   . mid-c          292        .526
    dist   . w-tex          297       2.83
    gas-oil. mid-c          295        .98
    gas-oil. w-tex          303       5.05
;
```

The table `attrib` could also be laid out as shown below:

```
Table attrib (ci,cr,q) "blending attributes"
            w-tex.density  mid-c.density  w-tex.sulfur  mid-c.sulfur
     naphtha      272             272         1.48          .283
     dist         297             292         2.83          .526
     gas-oil      303             295         5.05          .98
;
```

4.27.4.5 Condensing Tables

All the mechanisms using asterisks and parenthesized lists that were introduced in the discussion of sets are available here as well. For details on these mechanisms, see section Multi-Dimensional Sets. The following example shows how repeated columns or rows can be condensed with asterisks and lists in parentheses. The set membership is not shown, but can easily be inferred.

```
Table upgrade(strat,size,tech)
                small.tech1  small.tech2  medium.tech1  medium.tech2
    strategy-1       .05          .05          .05           .05
    strategy-2       .2           .2            .2           .2
    strategy-3       .2           .2            .2           .2
    strategy-4                                  .2           .2

Table upgradex(strat,size,tech) "alternative way of writing table"
                                          tech1*tech2
    strategy-1.(small,medium)                 .05
    strategy-2*strategy-3.(small,medium)      .2
    strategy-4.medium                         .2;
```

4.27.4.6 Handling Long Row Labels

It is possible to continue the row labels in a table on a second, or even third line in order to accommodate a reasonable number of columns. The break must come after a dot, and the rest of each line containing an incomplete row label-tuple must be blank.

The following example, adapted from [**INDUS**], is used to illustrate. This table actually has nine columns and many rows, here we have reproduced just a small part to show continued row label-tuples.

```
Table yield (c,t,s,w,z) "crop yield (metric tons per acre)"
                                                          nwfp     pmw
    wheat.(bullock, semi-mech).la-plant.
                                    (heavy, january)     .385     .338
    wheat.(bullock, semi-mech).la-plant. light           .506     .446
    wheat.(bullock, semi-mech).la-plant. standard        .592     .524
    wheat.(bullock, semi-mech).(qk-harv, standard).
                                    (heavy, january)     .439     .387
```

4.27.5 Constant Evaluation

Instead of fixed numerical values, one can also use constant expressions to assign values to parameters in a data statement. The syntax of constant expressions used in data statements follows the GAMS syntax as described in Data Manipulations with Parameters, but is restricted to scalar values and a subset of the GAMS intrinsic functions, as summarized below:

- Real numbers only
- Evaluation left to right
- Operator precedence:
  - ^ **
  - * /
  - + - binary and unary
  - < <= = <> >= > LE LE EQ NE GE GT
  - NOT
  - AND
  - OR XOR EQV IMP
- See Functions for list of supported functions

When used in a data statement, the constant expressions have to be enclosed in a pair of square brackets [ ] or curly brackets { }. Spaces can be used freely inside those brackets. Here is a little example:

```
Scalars x "PI half"       / [pi/2] /
        e "famous number" / [ exp( 1 ) ] /;

Parameter y "demo" / USA.(high,low) [1/3]
                     USA.medium {1/4}    /;
```

4.27.6 Data Entry by Assignment

Data may also be entered using assignment statements. Assignments are introduced and discussed in detail in section The Assignment Statement. This section here is a short outlook and shows how parameters that have already been declared may be assigned values. The general assignment statement has the following form:

```
parameter_name[(index_list)] = expression;
```

Here `parameter_name` is the name of a parameter that has been declared previously in a scalar, parameter or table statement, `index_list` indicates the controlling indices and may either contain a set or sets, a label or label tuple or a combination of those, and `expression` may be a number, a numerical expression or an acronym. For details on numerical expressions, see section Expressions.

The following examples illustrate how assignments may be used for data entry.

```
Set             j       /j1, j2, j3/;
Scalar          a1;
Scalars         a2      /11/;
Parameter       cc(j),
                bc(j)   /j2 22/;
a1 = 10;
a2 = 5;
cc(j) = bc(j)+10;
cc("j1") = 1;
```

The scalar `a1` is declared but not initialized in the first scalar statement. It is assigned the value of 10 in the first assignment. The scalar `a2` is initialized in the second scalar statement and this value is changed to 5 in the second assignment. Note that the original data is not retained. In the parameter statement the parameter `cc(j)` is declared but not initialized and the parameter `bc(j)` is only initialized for `j2`. This means that `bc('j2') = 22` and `bc('j1') = bc('j3') = 0`, the default value. Now, the third assignment sets the parameter `cc(j)` and assigns to all elements of the set `j` the value of the parameter `bc(j)` plus 10. So we have `cc('j2') = 32` and `cc('j1') = cc('j2') = 10`. Note that in this example the set `j` has only three elements so only 3 assignments are made simultaneously. However, suppose that the number of set elements is large, say 100,000, then to each element a value is assigned with just one assignment statement. Finally, the value of `cc('j1')` is changed to 1.

Observe that in the examples above assignments either refer to one specific set element or to the whole set. It is also possible to make assignments to only a part of the set. The mechanisms for partial set references are discussed in section Restricting the Domain in Assignments. Set elements that are not assigned new values in an assignment with a partial set reference retain their previous values. Recall that these may be the default value, values from the parameter or table statement, or values resulting from previous calculations.

4.27.7 Acronyms

An acronym is a special data type that allows the use of strings as values. Note that acronyms have *no numeric values* and are treated as character strings only.

4.27.7.1 The Syntax

The declaration for an acronym is similar to a set or parameter declaration. The basic format is as follows:

```
Acronym[s]  acronym_name [text] {, acronym_name [text]};
```

The keyword `acronym[s]` indicates that this is an `acronym` statement and `acronym_name` is the internal name of the acronym in GAMS, it is an identifier. The optional explanatory text is used to describe the acronym. For advice on explanatory text and how to choose an `acronym_name`, see the tutorial Good Coding Practices.

Note that more than one acronym may be declared in one acronym statement. The entries have to be separated by commas or by end of line. A simple example illustrates this:

```
Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
```

4.27.7.2 Acronym Usage

Acronyms may be used as data in scalar, parameter and table statements. An example for acronyms in a parameter statement follows.

```
Set machines / m-1*m-5 / ;
Acronym
    Monday, Tuesday, Wednesday, Thursday, Friday;
Parameter
    shutdown(machines)
     /   m-1  Tuesday
         m-2  Wednesday
         m-3  Friday
         m-4  Monday
         m-5  Thursday /;
```

Acronyms may also be used in assignments as in the example below. For more on assignments, see section The Assignment Statement.

```
Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
Scalar dayOfWeek;
dayOfWeek = Wednesday;
```

Note that *numerical* operations like addition or subtraction are not allowed with acronyms. Such operations would be meaningless since acronyms do not have numeric values.

Another context where acronyms may be used is in logical conditions. For more on logical conditions, see chapter Conditional Expressions, Assignments and Equations. This is shown in the following example:

```
Acronym Monday, Tuesday, Wednesday, Thursday, Friday;
Scalar dayOfWeek
       workHours /6/;
dayOfWeek = Wednesday;
workHours$(dayOfWeek <> Friday) = 8;
```

Note that only the equality and inequality operators may be used with acronyms. Other operations like addition and division are meaningless since acronyms do not have numeric values.

Acronyms are specific to GAMS and hence difficult to deal with when exchanging data with other systems. Users often replace parameters that contain acronyms with dynamic sets that have an additional index whose values correspond to the acronyms found in the original parameter. The machine shutdown data from above can be represented via a two-dimensional set as follows:

```
Set machines / m-1*m-5 /
    weekdays / Monday, Tuesday, Wednesday, Thursday, Friday /
    shutdown(machines,weekdays)
     /   m-1.Tuesday
         m-2.Wednesday
         m-3.Friday
         m-4.Monday
         m-5.Thursday /;
```

4.27.8 Summary

In this chapter, the declaration and initialization of parameters with the Scalar, Parameter, and Table statement have been discussed. Chapter Data Manipulations with Parameters will describe how this data can be changed with assignment statements.

4.28 Variables

4.28.1 Introduction

This chapter covers the declaration and manipulation of GAMS `variables`. Many of the concepts covered in the previous chapters are directly applicable here.

A `variable` is the GAMS name for what are called *endogenous variables* by economists, *columns* or *activities* by linear programming experts, and *decision variables* by industrial Operations Research practitioners. They are the entities whose values are generally unknown until after a model has been solved. A crucial difference between GAMS variables and columns in traditional mathematical programming terminology is that one GAMS variable is likely to be associated with many columns in the traditional formulation.

4.28.2 Variable Declarations

A GAMS variable, like all other identifiers, must be declared before it may be referenced.

4.28.2.1 The Syntax

The declaration of a `variable` is similar to a `set` or `parameter` declaration, in that domain lists and explanatory text are allowed and recommended, and several variables may be declared in one statement. The syntax is given below.

```
[var_type] variable[s] var_name [(index_list)] [text] [/var_data/] {, var_name [(index_list)] [text]
```

The keyword `var_type` denotes the optional variable type that is explained in detail in the next subsection. `Variable[s]` is the keyword that indicates that this is a variable statement. `Var_name` is the internal name of the variable in GAMS, it is an identifier. In the optional `index_list` the set or sets may be specified over which an indexed variable is declared. The optional explanatory text may be used to describe the variable for future reference and to ease readability. Specifying variable data is another optional element in the variable statement. Variable data allows to initialize variable attributes at compile time. For an example and details on variable attributes, see section Variable Attributes.

A typical variable statement adapted from the model [**RAMSEY**] is shown below for illustration:

```
Variables
   k(t)      capital stock (trillion rupees)
   c(t)      consumption (trillion rupees per year)
   i(t)      investment (trillion rupees per year)
   utility   utility measure;
```

The declaration of `k` above implies, as usual, that references to `k` are restricted to the domain of the set `t`. A model that includes `k` will probably have several corresponding variables in the associated mathematical programming problem: most likely one for each member of `t`. In this way, very large models can be constructed using a small number of variables. (It is quite unusual for a model to have as many as 50 distinct variables.) It is still unclear from the declaration whether `utility` is not domain checked or whether it is a scalar variable, i.e., one without associated sets. Later references will be used to settle the issue. For more details on domain checking, see section Domain Checking.

As the syntax indicates, the explanatory text is optional. However, it is important that variable declarations include explanatory text and that this be as descriptive as possible, since the text is used to annotate the solution output. Note the use of `'per'` instead of `'/'` in the text above: slashes are illegal in all unquoted text.

Note

- Variable names, the contained set element names plus the explanatory text must obey the general rules for language items.
- Variables can be defined over from 0 up to 20 sets
- The sets over which variables are declared indicate that these variables are potentially defined for every element of the defining sets. However the actual definition of variables does not occur until variables appear in an equation definition where the equation needs to be part of a model that in turn occurs in a solve statement.

4.28.2.2 Variable Types

There are nine basic types of variables that may be used in variable statements. These are shown in table Table 1.

Keyword	Description	Default Lower Bound	Default Upper Bound
`free` (default)	No bounds on variable. Both bounds may be changed from the default values by the user.	`-inf`	`+inf`
`positive` or `nonnegative`	No negative values are allowed for variable. The user may change both bounds from the default value.	`0`	`+inf`
`negative`	No positive values are allowed for variables. The user may change both bounds from the default value.	`-inf`	`0`
`binary`	Discrete variable that can only take values of 0 or 1. For details see section Types of Discrete Variables. In relaxed Model types the integrality requirement is relaxed.	`0`	`1`

Keyword	Description	Default Lower Bound	Default Upper Bound
integer	Discrete variable that can only take integer values between the bounds. The user may change both bounds from the default value. The default upper bound inside GAMS is +inf but when the variable is passed on to the solver, the option or command line parameter IntVarUp decides what upper bound (by default +inf) is passed on to the solver in case GAMS has upper bound +inf. In relaxed Model types the integrality requirement is relaxed.	0	+inf
sos1	A set of variables, such that at most one variable within a group may have a non-zero value. For details see section Types of Discrete Variables.	0	+inf
sos2	A set of variables, such that at most two variables within a group may have non-zero values and the two non-zero values are adjacent. For details see section Types of Discrete Variables.	0	+inf
semicont	Semi-continuous, must be zero or above a given minimum level. For details see section Types of Discrete Variables.	1	+inf

Keyword	Description	Default Lower Bound	Default Upper Bound
`semiint`	Semi-integer, must be zero or above a given minimum level and integer. For details see section Types of Discrete Variables. The default upper bound inside GAMS is `+inf` but when the variable is passed on to the solver, the option or command line parameter IntVarUp decides what upper bound (by default `+inf`) is passed on to the solver in case GAMS has upper bound +inf. In relaxed Model types the integrality requirement is relaxed.	1	`+inf`

Table 1: Variable types and default bounds

The default type is `free`, which means that if the type of the variable is not specified, it will not be bounded at all. The most frequently used types are `free` and `positive`. The type `positive variables` is used for variables for which negative values are meaningless, such as capacities, quantities or prices. Note that bounds may be changed using variable attributes and assignment statements, see section Variable Attributes.

Note

- Every optimization model must contain at least one unrestricted named variable (i.e. one declared with the keywords `Variable` or `Free Variable`). This variable is the objective variable. Even an objective variable can have lower and upper bounds assigned via the `.lo` and `.up` variable attribute.
- If a model is unbounded, a frequent cause for the unboundedness is that the modeler forgot to make a variable positive.

4.28.2.3 Styles for Variable Declaration

Two styles are commonly used to declare variable types. The first is to list all variables with domain specifications and explanatory text as a group, and later to group them separately as to type. The example shown below is adapted from [**MEXSS**]. The default type is `free`, so `phi` and `phipsi` will be `free` variables in the example below. Note the use of variable names derived from the original mathematical representation.

```
Variables
    u(c,i)  "purchase of domestic materials (mill units per yr)"
    v(c,j)  "imports            (mill tpy)"
    e(c,i)  "exports            (mill tpy)"
    phi     "total cost         (mill us$)"
    phipsi  "raw material cost  (mill us$)";
Positive Variables  u, v, e;
```

The commas in the list of positive variables are required separators.

Attention

It is possible to declare an identifier more than once. However, the second and any subsequent declarations should only add new information that does not contradict what has already been entered.

The second popular way of declaring variables is to list them in groups by type. We rewrite the example above using this second method:

```
Free Variables
    phi     "total cost         (mill us$)"
    phipsi  "raw material cost  (mill us$)"

Positive Variables
    u(c,i)  "purchase of domestic materials (mill units per yr)"
    v(c,j)  "imports   (mill typ)"
    e(c,i)  "exports   (mill typ)";
```

The choice between the two approaches is best based on clarity.

4.28.3 Variable Attributes

While a GAMS parameter has one number associated with each unique label combination, a variable has several. They represent:

Variable Attribute	Symbol	Description
Lower bound	`.lo`	Lower bound for the variable. Set by the user either explicitly or through default values associated with the variable type.
Upper bound	`.up`	Upper bound for the variable. Set by the user either explicitly or through default values associated with the variable type.
Fixed value	`.fx`	A fixed value for the variable. If set it results in the upper and lower bounds of the variable to be set to the value of the `.fx` attribute.
Activity level	`.l`	Activity level for the variable, also the current value or starting point. This attribute is reset to a new value when a model containing the variable is solved. The activity level is used to construct a basis for the model.
Marginal	`.m`	The marginal value (or reduced cost) for the variable. This attribute is reset to a new value when a model containing the variable is solved. The activity level is used to construct a basis for the model.
Scale factor	`.scale`	Numerical scaling factor for all coefficients associated with the variable if the model attribute `scaleopt` is set to 1. For more on scaling, see section Model Scaling - The Scale Option. Only applicable for continuous variables.
Branching priority	`.prior`	Branching priority value used in mixed integer programming models if the model attribute prioropt is set to 1. For details see section Setting Priorities for Branching. It can also be used to relax discrete restrictions by setting `.prior = +inf` regardless of the `prioropt` setting. Only applicable for discrete variables.
Stage	`.stage`	This attribute allows to assign variables to stages in a stochastic program or other block structured model. Thus, among other places, it is used for 2-stage stochastic programs solved with DECIS or the Benders partition in Cplex.

Table 2: Variable Attributes

Users distinguish between these values when necessary by appending the suffix to the variable name. Examples are given below.

It is possible to specify initial values for these variable attributes at compile time. This can be done within the variable declaration statement as illustrated in the following example or during execution time as explained in section Assigning Values to Variable Attributes.

```
Variable x1(j) my first  / j1.up 10 , j1.lo 5, j1.l 7, j1.m 0, j1.scale 20 /;
```

The upper bound of the variable `x1("j1")` is set to 10, the lower bound is set to 5, the starting value for the activity level is set to 7 and the starting value for the marginal is set to 0. The variable is also scaled by the factor 20, which means it is *multiplied* by 20.

Note that it is also possible to use a table structure to assign values to variable attributes. The following example is adapted from model [**TRNSPORT**].

```
Variable Table x(i,j) initial values
                            l      m
    seattle.  new-york      50
    seattle.  chicago      300
    san-diego.new-york     275
    san-diego.chicago           0.009;
```

Note

- `.fx` and attributes `.lo` and `.up` on the same variable cannot be in a data statement. `.fx` sets both `.lo` and `.up` and hence we would have a double definition of the same attribute. Since attribute `.scale` is applicable for continuous variables and attribute `.prior` for discrete variables, they share the same internal space in a GAMS variable. Some solvers can make use of priorities even for continuous variables (e.g. BARON). Such priorities need to be supplied via a solver option file.
- The attribute `.stage` uses the same internal space as `.scale` and `.prior`. So a model cannot specify scale factor and branching priorities together with stages.
- Fixing a semi-continuous or semi-integer variable to a non-zero value like `x.fx = 4` does not result in a truly fixed variable. The domain of the variable remains {0,4}. To really fix a semi-continuous or semi-integer variable, the discrete restriction could be relaxed by setting the branching priority to infinity (`x.prior=inf`).
- For variables in discrete models (such as MIP, MINLP), the `.m` attribute provides the marginals obtained by fixing all the discrete variables and solving the resulting continuous problem (such as LP, NLP). Many solvers allow to enable/disable solving such a fixed problem. When disabled, no marginals will be provided for discrete models.

In addition to the variable attributes introduced above, there are a number of variable attributes that cannot be assigned or exported via execute_unload∗ but may be used in computations. They are given in Table 3.

Variable Attribute	Symbol	Description
Range	`.range`	The difference between the lower and upper bounds for a variable. It becomes zero if the lower equals the upper bound, e.g. if the `fx` attribute is set.

Variable Attribute	Symbol	Description
Slack upper bound	`.slackup`	Slack from variable upper bound. This is defined as the greater of two values: zero or the difference between the upper bound and the level value of a variable.
Slack lower bound	`.slacklo`	Slack from variable lower bound. This is defined as the greater of two values: zero or the difference between the level value and the lower bound of a variable.
Slack	`.slack`	Minimum slack from variable bound. This is defined as the minimum of two values: the slack from the variable lower bound and the slack from the variable upper bound.
Infeasibility	`.infeas`	Amount by which a variable is infeasible falling below its lower bound or above its upper bound. This is defined as the smallest of three values: zero, the difference between the lower bound and the level value, the difference between the level value and the upper bound of a variable, i.e. max(0, lower-level, level-upper).

Table 3: Additional Variable Attributes that Cannot Be Assigned but May Be Used in Computations.

4.28.3.1 Bounds on Variables

All default bounds set at declaration time may be changed using assignment statements.

Attention

> For discrete variable types, the consequences of the type declaration cannot be completely undone (e.g. the `scale` attribute is not available) but their value domain can be changed to continuous by setting attribute `prior` to infinity.

Bounds on variables are the responsibility of the user. After variables have been declared, default bounds have already been assigned: for many purposes, especially in linear models, the default bounds are sufficient. In nonlinear models, however, bounds play a far more important role. It may be necessary to provide bounds to prevent undefined operations, such as division by zero. In nonlinear programming it is often necessary to define a 'reasonable' solution space that will assist in efficiently finding a solution.

Attention

> The lower bound cannot be greater than the upper bound: if you happen to impose such a condition, GAMS will generate an execution error, namely "`**** Matrix error - lower bound > upper bound`" when executing a solve statement.

Note that the upper bound on integer and semi-integer variables needs special consideration. The default upper bound is `+inf` and the option or command line parameter IntVarUp controls what upper bound is sent to the solver. With the current default value (0) of IntVarUp, an upper bound of `+inf` is sent to the solver. Setting IntVarUp to one will pass 100 as the default upper bound to the solver. The other available values for IntVarUp work like zero, but enable special reports/execution errors in case the solution reports a level value greater than 100 for any integer variable with a default bound of `+inf`.

4.28.3.2 Fixing Variables

GAMS allows the user to fix variables through the `.fx` variable suffix. This is almost equivalent to setting the lower bound and upper bound equal to the fixed value. The attribute .fx also resets the activity level .l to the fixed value. When setting .lo and .up the activity level remains unchanged. A `solve` statement will project the activity level within the active bounds. Fixed variables can subsequently be freed by changing the lower and upper bounds.

4.28.3.3 Activity Levels of Variables

GAMS allows the user to set the activity levels of variables through the `.l` variable suffix. These activity levels of the variables prior to the solve statement serve as initial value for the solver. This is particularly important for nonlinear programming problems. For discrete models in many cases the solver needs an additional indicator to interpret the activity levels as a feasible integer solution via a solver option (e.g. Cplex' mipstart).

Attention

- GAMS only stores variables with non-default values (similar to storing only non-zero values of parameters). Non-default variables can be accidentally created by using harmlessly looking assignments like

  ```
  x.up(i,j,k,l) = 0;
  ```

  Even if the equations only reference such variables over a small subset of `(i,j,k,l)` this statement creates `card(i)*card(j)*card(k)*card(l)` variable records in the GAMS database. Such fixings of `x(i,j,k,l)` to 0 can be avoided by using dynamic sets or dollar conditions in the equation algebra to only reference tuples of `(i,j,k,l)` for which `x(i,j,k,l)` can possible have a non-zero value.
- In order to filter only necessary tuples for an equation the filtering conditions needs to be provided only once when defining the equation (`equ(i,j,k)`). This is different for variables because they appear in many equations and the filtering condition needs to be potentially repeated many time. Therefore it is good practice and reduces GAMS model generation time if the filtering of the variables is governed by a dynamic set:

  ```
  sum((i,j)$(ord(i)>ord(j) and cap(i,j)>0), x(i,j))
  ```

  versus

  ```
  set net(i,j); net(i,j) = ord(i)>ord(j) and cap(i,j)>0;
  sum(net(i,j), x(i,j))
  ```

4.28.4 Variables in Display and Assignment Statements

GAMS allows the modeler to use the values associated with the various attributes of each variable in assignment and `display` statements. The next two subsections explain the use of variables on the left and right-hand sides of assignment statements respectively. Then we will explain the use of variables in `display` statements.

4.28.4.1 Assigning Values to Variable Attributes

Assignment statements operate on one variable attribute at a time, and require the suffix to specify which attribute is being used. Any index list comes after the suffix.

The following code snippets are from models [**MEXSS**] and [**RAMSEY**]. The first example illustrates the use of assignment statements to set upper and lower bounds on variables.

```
x.up(c,i,j)  =  1000 ;  phi.lo  =  inf  ;
```

A very common use is to bound one particular entry individually:

```
p.up('pellets', 'ahmsa', 'mexico-df')  = 200  ;
```

Or to put small lower bounds on a variable identifier used as a divisor in a nonlinear program:

```
c.lo(t)   =  0.01  ;
```

Or to provide initial values for a nonlinear problem:

```
c.l(t)   =  4*cinit(t)  ;
```

Remember that the order is important in assignments, and notice that the two pairs of statements below produce very different results. In the first case, the lower bound for `c('1985')` will be 0.01, but in the second, the lower bound is 1.

```
c.fx('1985') = 1;              c.lo(t)       = 0.01 ;
c.lo(t)      = 0.01 ;          c.fx ('1985') = 1 ;
```

Everything works as described in the previous chapter, including the various mechanisms described there of indexed operations, dollar operations, subset assignments and so on. An example from model LOCATION follows.

```
ship_sm.lo(sl,m)$(ord(sl) = 1 and ord(m) = 1) = 1;
```

The lower bound of the variable `ship_sm(sl,m)` is set to 1 and this assignment is only valid for `ship_sm('s1','d1')`, the realization of the variable where both indices are the first members of their respective sets.

4.28.4.2 Variable Attributes in Assignments

Using variable attributes on the right-hand side of assignment statements is important for a variety of reasons. Two common uses are for generating reports and for generating initial values for some variables based on the values of other variables. For more on variable attributes in report writing, see section Displaying Variable Attributes below and especially chapter The Put Writing Facility.

The following examples adapted from model **[CHENERY]** illustrate the use of variable attributes on the right-hand side of assignment statements:

```
* initial values for variables

y.l(i) = 250 ; x.l(i) = 200 ;
e.l(t) =   0 ; m.l(t) =   0 ;

g.l(t) = mew(t) + xsi(t)*m.l(t) ;
h.l(t) = gam(t) - alp(t)*e.l(t) ;

[...]

* generating report after solve

Scalar
    cva   "total value added at current prices"
    rva   "real value added"
    cli   "cost of living index"  ;

cva  =  sum (i, v.l(i)*x.l(i))  ;
cli  =  sum(i, p.l(i)*ynot(i))/sum(i, ynot(i))  ;
rva  =  cva/cli  ;

Display cli, cva, rva ;
```

As with parameters, a variable must have some non-default data values associated with it before it can be used in a `display` statement or on the right-hand side of an assignment statement. After a `solve` statement has been processed or if non-default values have been set with an assignment statement, this condition is satisfied. `Solve` statements are introduced and discussed in chapter Model and Solve Statements.

Attention

The `.fx` suffix is mostly just a shorthand for `.lo` and `.up` and can therefore only be used only on the left-hand side of an assignment statement.

Note

In general, the variable level needs to be specified via the attribute `.l` for assignment statements. However, the dollar control option $on/offDotL allows the implicit use of the attribute `.l` in assignment statements, thus it facilitates using the same algebra in model definitions and assignment statements. This is especially useful in the context of macros.

4.28.4.3 Displaying Variable Attributes

The `display` statement is introduced and discussed in detail in chapter The Display Statement. Here we demonstrate how variable attributes are used in `display` statements.

Since several values are associated with each variable, the user must specify which attribute should be displayed when using variables in `display` statements. As before, appending the appropriate suffix to the variable name does this and no domain specification may appear. As an example, we show how to `display` the level of `phi` and the level and the marginal values of `v` from [**MEXSS**]:

```
display phi.l, v.l, v.m;
```

The output looks similar, except that (of course) the listing shows which of the values is being displayed. Because zeroes, and especially all zero rows or columns, are suppressed, the patterns seen in the level and marginal displays will be quite different, since non-zero marginal values are often associated with activity levels of zero.

```
Mexico Steel - Small Static   (MEXSS,SEQ=15)
E x e c u t i o n

----    203 VARIABLE  PHI.L                =      538.811 total cost
                                                         (mill us$)
----    203 VARIABLE  V.L           imports
                                     (mill tpy)
                     ( ALL       0.000 )

----    203 VARIABLE  V.M           imports
                                     (mill tpy)
        mexico-df   monterrey   guadalaja
steel       7.018      18.822       6.606
```

We should mention here a clarification of our previous discussion of displays. It is actually the default values that are suppressed on display output. For parameters and variable levels and marginals, the default is zero, and so zero entries are not shown. For bounds, however, the defaults can be non-zero. The default value for the upper bound of a positive variable is `+INF`, and if we would also display `v.up` above, for example, we would see:

```
----    203 VARIABLE  V.UP          imports
                                    (mill tpy)
                    ( ALL        +INF )
```

If any of the bounds have been changed from the default value, then only the entries for the changed elements will be shown. This may sound confusing, but since few users display bounds it has not proved troublesome in practice.

Note

The attribute `.range` may be used in `display` statements. It provides a convenient way to check whether a variable is fixed as it lists the 0 values explicate because the default for range (which won't be displayed) is +inf.

4.28.5 Summary

Remember that wherever a parameter may appear in a display or an assignment statement, a variable may also appear - provided that it is qualified with one of the suffixes. The only places where a variable name may appear without a suffix is in a variable declaration, as has been shown in this chapter, in an equation definition, which is discussed in chapter Equations, or in a $on/offDotL block.

4.29 Equations

4.29.1 Introduction

The keyword `equation` defines GAMS names that may be used in the model statement. A GAMS equation name is associated with the symbolic algebraic relationships that will be used to generate the constraints in a model. The algebraic relationships are defined by using constants, mathematical operators, functions, sets, parameters and variables. As with variables, one GAMS equation may be defined over a group of sets and in turn map into several individual constraints associated with the elements of those sets. Most of the example code in this chapter is from the model `location`.

This chapter is organized as follows. First, we introduce how equations are declared and defined, then we discuss expressions in equation definitions, followed by a section on equation attributes. A summary and quick reference conclude the chapter.

4.29.2 Declaring Equations

An `equation` must be declared before it can be defined and used in a model.

4.29.2.1 The Syntax

The declaration of an `equation` is similar to a `set` or `parameter` declaration. The syntax is given below.

```
Equation[s] eqn_name [(index_list)] [explanatory text] [/eqn_data/] {, eqn_name [(index_list)] [expla
```

`Equation[s]` is the reserved word that indicates that one or more blocks of equations are about to be declared. A block of equations may initiate one or more individual constraints. `Eqn_name` is the internal name of the equation, an identifier in GAMS. In the optional `index_list` the set or sets are specified over which an indexed equation is declared. The optional explanatory text may be used to describe the equation for future reference and to ease readability. Specifying equation data is another optional element in the equation declaration. Equation data allows to initialize equation attributes at compile time. For an example see the next section. For more on equation attributes see section Equation Attributes.

One or more equations may be declared in one `equation` statement. The equation names have to be separated by commas or by a line break as in the example that follows. The end of the declaration statement is indicated by a semicolon.

Note

> It is good practice to end the equation declaration with a semicolon, even though it is not mandatory if the next statement starts with a GAMS keyword.

For advice on choosing equation names and phrasing the explanatory text see chapter Good Modeling Practices.

4.29.2.2 An Illustrative Example

The following example is from the model `location`. In addition to the equation declarations the relevant set definitions are given.

```
Sets
  sl                    'supply locations'              /s1, s2/
  wh                    'warehouse locations'           /a, b, c/;

Equations
  tcost_eq              'total cost accounting equation'
  supply_eq(sl)         'limit on supply available at supply location'
  capacity_eq(wh)       'warehouse capacity'  /a.scale 50, a.l 10, b.m 20/;
```

The keyword `Equations` marks the beginning of the equation declaration. Each equation name is optionally followed by its domain (associated set or sets) unless it is a *scalar equation*. It is possible but not good practice to declare indexed equations without their domains. The name of the first equation is `tcost_eq` and it is followed by the explanatory text '`total cost accounting equation`'. The name of the equation `tcost_eq` is not followed by any associated sets. Since we follow good practice here we assume that `tcost_eq` is a scalar equation. Scalar equations do not have any associated sets and will generally produce one equation in the model. For more on scalar equations see subsection Scalar Equations.

The other two equations are *indexed equations*, they are declared over a set. The equation `supply_eq` is declared over the set `sl` and the equation `capacity_eq` is declared over the set `wh`. In typical circumstances an indexed equation declaration implies that a block of constraints will be generated. For example, equation `supply_eq(sl)` implies that two constraints will be generated, one for each element of the set `sl`. For more on indexed equations see subsection Indexed Equations.

The declaration of the equation `capacity_eq` specifies some equation attributes. The first entry indicates that the equation `capacity_eq('a')` is scaled by a factor of 50, which means division of all entries in that equation by 50 upon model passage to the solver. For more on scaling see section Model Scaling - The Scale Option. The second entry sets the initial value of the equation `capacity_eq('a')` to 10 and `b.m` means that the initial marginal value of the equation `capacity_eq('b')` is set to 20. Alternatively, a table structure may be used to specify the values of equation attributes. The following table may replace the notation above.

```
Equation Table capacity_eq(wh)  'warehouse capacity'
      scale    l       m
   a   50      10
   b                   20 ;
```

For more on equation attributes see section Equation Attributes.

Note

An equation may be declared over more than one set.

4.29.3 Defining Equations

After declaring equations they have to be defined. The definition of an equation specifies the algebraic structure of the equation in GAMS. The syntax is given first, an illustrative example follows and in the remainder of this section some of the key components of equation definitions are discussed.

4.29.3.1 The Syntax

The syntax for defining an equation is as follows:

```
eqn_name(index_list)[$logical_condition(s)].. expression eqn_type expression ;
```

`Eqn_name` is the name of the equation as introduced in the equation declaration, that may be followed by an `index_list` for indexed equations. In the `index_list` the set or sets are specified over which an indexed equation is defined. These sets are also called *domain of definition* of the equation. One or more logical conditions are optional. For an example see Indexed Equations. For more on logical conditions in equation definitions see Dollar Control over the Domain of Definition. The two dots `'..'` are mandatory and indicate the start of the algebra. It is good practice to end the definition of an equation with a semicolon, even though it is not mandatory if the next statement starts with a GAMS keyword.

Attention

An equation must be declared before it is defined.

`Expression` refers to an algebraic expression which may include variables, parameters, functions, and constants among other items. For details on expressions in GAMS, see section Expressions.

Attention

Only variables that appear at least once with a nonzero coefficient in an equation definition will appear in a model.

`Eqn_type` refers to the equation type denoted by the symbol between the right-hand side and left-hand side expressions that form the equation. The symbols that are allowed are given in Table 1.

Type	Description
`=e=`	Equality: right-hand side must equal left-hand side.
`=g=`	Greater than: left-hand side must be greater than or equal to right-hand side.
`=l=`	Less than: left-hand side must be less than or equal to right-hand side.
`=n=`	No relationship implied between left-hand side and right-hand side. This equation type is ideally suited for use in MCP models and in variational inequalities.
	Equation is defined by external programs. See External Equations.

Table 1: Equation Types

Equation definitions may be carried over as many lines of input as needed. Blanks may be inserted to improve readability, and expressions may be arbitrarily complicated.

Note that an equation can only be defined once. By using logical conditions it is possible to control which constraints are generated. In addition, the components of an equation may be modified by changing the data it uses. However, if the logic of the equation needs to be changed then a new equation with a new name has to be declared and defined.

4.29.3.2 An Illustrative Example

Consider the following example adapted from the model [**mexss**]. The associated variable and equation declarations are also included.

```
Variables  phi, phipsi, philam, phipi, phieps ;
Equations  obj ;
obj.. phi  =e=  phipsi + philam + phipi - phieps ;
```

The name of the equation being defined is `obj` and the symbol `=e=` indicates that this is an equality. Any of the following forms of the equation are mathematically equivalent.

```
obj..  phipsi + philam + phipi - phieps =e= phi  ;
obj..  phieps - phipsi  =e=  philam - phi + phipi  ;
obj..  phi + phieps - phipsi - philam - phipi  =e=  0 ;
obj..  0 =e= phi + phieps - phipsi - philam - phipi  ;
```

Note

The arrangement of the terms in the equation is a matter of choice, but often a particular one is chosen because it makes the model easier to understand.

4.29.3.3 Scalar Equations

A scalar equation will produce one equation in the associated optimization problem. The equation `obj` defined above is an example of a scalar equation which contains only scalar variables. Note that scalar equations may contain indexed variables. However, they must occur with an indexed operator such as `sum` or `prod`, unless the indexed variables refer to a singleton set (a set with only one element). Consider the following example from the model `location`. Note that the set `wh` has three elements.

```
configure_eq..  sum(wh,build(wh)) =l= 1;
```

The variable `build` is defined over the set `wh`, it is an indexed variable. It may be used in the scalar equation `configure_eq` since it occurs in conjunction with the indexed operator `sum`.

4.29.3.4 Indexed Equations

All the set references in scalar equations are within the scope of indexed operators or they refer to singleton sets; thus many variable, set and parameter references can be included in one equation. In addition, GAMS also allows for equations to be defined over a domain, thereby developing a compact representation for constraints. The index sets to the left of '..' are called the *domain of definition* of the equation.

Note

- Domain checking ensures that the domain over which an equation is defined is the set (or the sets) or a subset of the set (or the sets) over which the equation was declared.
- As a corollary, domain checking also catches the error of the indices being listed in an inconsistent order. For example, declaring an equation as `myequation(s,t)` and then naming it in the definition as `myequation(t,s)` causes an error (unless `s` and `t` are aliases of the same set). For more information, see section Domain Checking.

The following indexed equation with a single index generates a separate constraint for each member of the driving (or controlling) set. It is taken from the model [**chenery**]. In this example, `t` is a set with three members, `mew` and `xsi` are parameters and `m` and `g` are variables.

```
dg(t)..  g(t)  =e= mew(t) + xsi(t)*m(t)  ;
```

As the set `t` has three members, three constraints will be generated, one for each member of `t` specifying the dependence of `g` on `m`. The data associated with the parameters `mew` and `xsi` are used to build the individual constraints. This data does not have to be known when the equation is defined, but it has to be populated before a model containing the equation is solved.

The extension to two or more indices on the left of '..' is obvious. There will be one constraint generated for each combination of set elements that can be constructed using the indices inside the parenthesis. Here are two examples from the model [**aircraft**], a scheduling model.

```
bd(j,h)..  b(j,h)  =e=  dd(j,h) - y(j,h) ;
yd(j,h)..  y(j,h)  =l=  sum(i, p(i,j)*x(i,j)) ;
```

The domain of definition of both equations is the Cartesian product of `j` and `h`: constraints will be generated for every set element pair that can be constructed from the members of these two sets.

The next example illustrates the use of the optional logical conditions in the definition of equations. It is taken from the production and distribution model [**ferts**].

```
CC(m,i)$mpos(m,i)..  sum(p$ppos(p,i), b(m,p)*z(p,i)) =l= util*k(m,i) ;
```

`CC` is a capacity constraint defined for elements of the sets `m` and `i`. However, in this case not all cases of `m` exist at each location `i`, and the mapping set `mpos(m,i)` tells the cases where `m` exists at `i` and thus is used to restrict the domain cases for which the constraints are actually generated. The control of the summation over `p` with `ppos(p,i)` is an additional logical condition, and is required because not all processes `p` are possible at all locations `i`.

The equation may alternatively written in the following way:

```
CC(mpos(m,i))..  sum(ppos(p,i), b(m,p)*z(p,i)) =l= util*k(m,i) ;
```

Instead of defining the equation over the indices `(m,i)` the equation is defined over the set `mpos` that is itself defined over the indices `(m,i)`. A similar logic applies to restricting the summation.

Conditional expressions are introduced and discussed in the section Conditional Expressions, Assignments and Equations. See specifically Dollar Control over the Domain of Definition for logical conditions in equation definitions.

4.29.3.5 Using Labels Explicitly in Equations

Sometimes it can be necessary to refer to specific set elements in equations. This can be done as with parameters - by using quotes or double quotes around the label. Consider the following example from the model `location`:

```
sum(m, ship_wm(wh,m)) =l= build(wh)*data(wh,"capacity") ;
```

4.29.3.6 Logic Equations

Logic equations use Boolean algebra and have to evaluate to `TRUE` (or 1) to be feasible. The Boolean functions available in GAMS and the default order of precedence of the operators are given in Table 2. Note that 1 denotes the highest order of precedence or the most binding operator and 3 denotes the lowest order of precedence or the least binding operators. As usual, the default order of precedence holds only in the absence of parentheses and operators on the same level are evaluated from left to right.

Function	Operator	Alternative Notation	Return Values	Order of precedence
Negation	`not x`	`bool_not(x)`	returns 1 if $x = 0$, else returns 0	1
Logical conjunction	`x and y`	`bool_and(x,y)`	returns 1 if $x \neq 0$ and $y \neq 0$, else returns 0	2
Logical disjunction	`x or y`	`bool_or(x,y)`	returns 0 if $x = y = 0$, else returns 1	3
Exclusive disjunction	`x xor y`	`bool_xor(x,y)`	returns 1 if *exactly* one argument is $\neq 0$, else returns 0	3
Material implication	`x imp y` or `x -> y`	`bool_imp(x,y)`	returns 0 if $x \neq 0$ and $y = 0$, else returns 1	3
Material equivalence	`x eqv y` or `x <=> y`	`bool_eqv(x,y)`	returns 0 if *exactly* one argument is 0, else returns 1	3

Table 2: Boolean Functions and Operator Precedence

There are three ways to declare and define logic equations:

1. The logic equation is declared using the keyword `Logic Equation` and the definition contains only Boolean algebra symbols.
2. The logic equation is declared like any other equation using the keyword `Equation` and in the definition the symbol `=b=` appears indicating that it is a logic equation.
3. This is a combination of the first two options: the equation is declared with the keyword `Logic Equation` and defined using the symbol `=b=`.

The following example demonstrates the first way to declare and define a logic equation. It is adapted from the food manufacturing problem [**foodemp**]. In this problem the blending of oils is modeled.

```
Sets
  m             "planning period (month)"    / m1*m6 /
  p             "raw oils"                   / v1*v2, o1*o3 / ;

Variables
  induse(m,p)   "indicator for usage of raw oil per month" ;
Binary variable induse;

Logic Equation
  deflogic(m)  "if some vegetable raw oil is used we also need to use the non-vegetable oil o3" ;

deflogic(m)..  induse(m,'v1') or  induse(m,'v2') -> induse(m,'o3');
```

The variable `induse` is a binary variable, it can only take the values 0 and 1. The equation ensures that in an optimal solution if either vegetable oil `v1` or vegetable oil `v2` is blended in a product, then non-vegetable oil `o3` is also blended in that product.

An alternative formulation of the equation `deflogic` using the `=b=` notation is given below.

```
deflogic(m)..  induse(m,'v1') or induse(m,'v2') -> induse(m,'o3') =b= 1;
```

Note that the value of 1 on the right-hand side means that the logic expression on the left-hand side must evaluate to `TRUE` in a feasible solution. To illustrate further, we could negate the left-hand side expression using the logic operator `not` and then the right-hand side would have to evaluate to zero or `FALSE` to yield the same result as above. The respective equation definition follows.

```
deflogic(m)..  not (induse(m,'v1') or  induse(m,'v2') -> induse(m,'o3')) =b= 0;
```

Logic equations with binary variables and boolean functions/operators are reformulated into linear constraints by the LOGMip solver using GAMS model type `EMP`. Solver CONVERT writes scalar models with logic equations, but no other solver currently can utilize logic equations. In principle (but currently not implemented), logic constraints can be used to express complex algebra, like indicator constraints:

```
defindic(m)..  induse(m,'v1') = 1 -> sum(p, use(m,p)) >= minuse(m);
```

4.29.4 Expressions in Equation Definitions

The arithmetic operators and some of the functions that are described in section Expressions may be used in equation definitions.

Consider the following example adapted from the model **[chenery]** demonstrating the use of parentheses and exponentiation.

```
dem(i) ..    y(i)  =e= ynot(i)*(pd*p(i))**thet(i)  ;
```

A list of arithmetic operators is given in subsections Standard Arithmetic Operations and Indexed Operations.

4.29.4.1 Functions in Equation Definitions

All available GAMS functions are listed in the section Functions. Some functions are not allowed at all in equation definitions. They include random distribution functions and are marked with *none* in the third column of the tables listing all functions.

Attention

Some functions like `uniform` and `normal` are not allowed in equation definitions.

The use of the other functions is determined by the type of arguments in the model. There are two types of arguments:

1. *Exogenous arguments:* The arguments are known. Parameters and variable attributes (for example, `.l` and `.m` attributes) are used as arguments. The expression is evaluated once when the model is being set up and most mathematical functions as well as time and calendar functions are allowed.
2. *Endogenous arguments:* The arguments are variables and therefore unknown at the time of model setup. The function will be evaluated many times at intermediate points while the model is being solved. Note that the occurrence of any function with endogenous arguments implies that the model is not linear.

Functions that are allowed only with exogenous arguments are marked with *any* in the tables listing all functions.

There are two types of functions allowing endogenous arguments: *smooth* functions and *discontinuous* functions. Smooth functions are continuous functions with continuous derivatives (like `sin`, `exp`, `log`). Discontinuous functions include continuous functions with discontinuous derivatives (like `max`, `min`, `abs`) and discontinuous functions (like `ceil`, `sign`). Smooth functions may be used routinely in nonlinear models. However, discontinuous functions may cause numerical problems and should be used only if unavoidable, and only in a special model type called `DNLP`. For more details on model types see section Classification of Models.

Attention

The best way to model discontinuous functions is with binary variables. The result is a model of the type `MINLP`. The model **[ABSMIP]** demonstrates this formulation technique for the functions `abs`, `min`, `max` and `sign`. See also section Reformulating DNLP Models. We strongly discourage the use of the `DNLP` model type.

In Table 3 the use of functions in equation definitions is summarized.

Functions are allowed ...	Description of Functions
not at all	Functions that are marked *none* in the third column of the tables listing all functions in section Functions.
only with exogenous arguments	Functions that are marked *any* in the third column of the tables listing all functions in section Functions.
with endogenous arguments	Smooth functions. They are marked *NLP* in the third column of the tables listing all functions in section Functions.
with endogenous arguments	Discontinuous functions. They are marked *DNLP* in the third column of the tables listing all functions in section Functions.

Table 3: Functions in Equation Definitions

4.29.4.2 Preventing Undefined Operations in Equations

Some operations are not defined at particular values of the arguments. Two examples are division by 0 and the `log-` of 0. While this can easily be identified at model setup for exogenous functions and expressions, it is a lot more difficult when the terms involve variables. The expression may be evaluated many times when the problem is being solved and the undefined result may arise only under certain cases. One way to avoid an expression becoming undefined is adding bounds to the respective variables. Consider the following example from the model [**ramsey**]:

```
c.lo(t) = 0.01 ;
util ..   utility  =e= sum(t, beta(t)*log(c(t))) ;
```

Specifying a lower bound for `c(t)` that is slightly larger than 0 prevents the `log`-function from becoming undefined.

4.29.5 Equation Attributes

Equation attributes may be specified in a similar way as variable attributes. Five values are associated with each unique label combination of every equation. They are denoted by the suffixes `.l`, `.m`, `.lo`, `.up` and `.scale`. A list of the attributes and their description is given in Table 4.

Equation Attribute	Symbol	Description
Lower bound	`.lo`	Negative infinity for `=l=` equations. Right hand side value for `=g=`, `=e=`, and `=b=` equations. Zero for `=c=` equations.
Upper bound	`.up`	Positive infinity for `=g=` and `=c=` equations. Right hand side value for `=l=`, `=e=`, and `=b=` equations.
Equation level	`.l`	Level of the equation in the current solution, equal to the level of all terms involving variables.
Marginal	`.m`	Marginal value for equation. This attribute is reset to a new value when a model containing the equation is solved. The marginal value for an equation is also known as the shadow price for the equation and in general not defined before solution but if present it can help to provide a basis for the model
Scale factor	`.scale`	Numerical scaling factor that scales all coefficients in the equation. This is only used when the model attribute `scaleopt` is set to 1. For more on scaling, see section Model Scaling - The Scale Option.
Stage	`.stage`	This attribute allows to assign equations to stages in a stochastic program or other block structured model. Its current use is limited to 2-stage stochastic programs solved with DECIS.

Table 4: Equation Attributes

Note that all attributes except for `.scale` and `.stage` contain the attribute values of equations *after* a solution of the model has been obtained. For some solvers it can be useful to specify marginal values `.m` and level values `.l` on input to provide starting information. Also note that the marginal value is also known as the *dual* or *shadow price*. Roughly speaking, the marginal value `.m` of an equation is the amount by which the value of the objective variable would change if the equation level were moved one unit.

Equation attributes may be referenced in expressions and can be used to specify starting values (see section Declaring Equations). In addition, they serve for scaling purposes and for reporting after a model was solved. For example, they may be displayed using the `display` statement. The following example is from the model `location`.

```
Model warehouse 'warehouse location model'  /all/;
solve warehouse using mip min tcost;
display supply_eq.l;
```

The `display` statement generates the following output at the end of the listing file:

```
----    108 EQUATION supply_eq.L  limit on supply available at supply location

s1 50.000,    s2 75.000
```

The level values of the equation `supply_eq` are displayed. As expected, there are two level values, one for each member of the set `sl` over which the equation `supply_eq` was defined.

Note

By default, all equation attributes introduced above except for `.scale` are echoed to the solution report that is part of the listing file.

In addition to the equation attributes introduced above, there are a number of equation attributes that cannot be assigned or exported via execute_unload∗ but may be used in computations. They are given in Table 5.

Equation Attribute	Symbol	Description
Range	`.range`	The difference between the lower and upper bounds of an equation.
Slack lower bound	`.slacklo`	Slack from equation lower bound. This is defined as the greater of two values: zero or the difference between the level value and the lower bound of an equation.
Slack upper bound	`.slackup`	Slack from equation upper bound. This is defined as the greater of two values: zero or the difference between the upper bound and the level value of an equation.
Slack	`.slack`	Minimum slack from equation bound. This is defined as the minimum of two values: the slack from equation lower bound and the slack from equation upper bound.
Infeasibility	`.infeas`	Amount by which an equation is infeasible falling below its lower bound or above its upper bound. This is defined as max(0, lower bound - level, level - upper bound).

Table 5: Additional Equation Attributes that Cannot Be Assigned but May Be Used in Computations.

4.29.6 Summary and Quick Reference

In this chapter we have covered the declaration and definition of `equations` in GAMS, arithmetic operations and functions that may be used in equations and equation attributes. A list summarizing the main points to keep in mind follows.

- Equations must be declared before they may be defined. [1]
- It is good practice to add an explanatory text to the declaration. [2]
- More than one equation may be declared at once. The equation names have to be separated by commas or by a line break. [3]
- Equations may be declared and defined over sets which are called the domain of definition of the equation. [4]
- One indexed equation may generate many constraints depending on the size of the set(s) over which it is defined. [5]
- Equations may be defined over subsets. The dollar condition may be used to filter the members of a set so that only a subset of the members are considered. [6]
- The set(s) over which an equation is defined must be consistent with the set(s) over which the equation was declared, being the set(s) themselves or a subset of the set(s). [7]
- The arrangement of terms in an equation is up to the other. Variables can appear on both sides of an equation. [8]
- Labels of specific set elements may be used explicitly in equations. [9]
- All arithmetic operations that may be used to evaluate expressions are also allowed in equations. [10]
- Many functions that are defined in GAMS may be used in equations. [11]
- It is good practice to set bounds for variables to avoid undefined operations if equations contain operations that are undefined at certain values. [12]
- Equations have attributes similar to variables (`.l`, `.m`, `.lo`, `.up` and `.scale`). [13]

4.30 Model and Solve Statements

4.30.1 Introduction

This chapter brings together all the concepts discussed in previous chapters by explaining how to specify a model and solve it.

4.30.2 The Model Statement

The model statement is used to collect equations into groups and to label them so that they can be solved. The simplest form of the model statement uses the keyword `all`: the model consists of all equations declared before the model statement is entered. For most simple applications this is all the user needs to know about the model statement.

4.30.2.1 The Syntax

In general, the syntax for a model declaration in GAMS is as follows:

```
model[s] model_name [text] [/ (all | eqn_name {, eqn_name}) {, var_name(set_name)} /]
       {,model_name [text] [/ (all | eqn_name {, eqn_name}) {, var_name(set_name)} /]} ;
```

The keyword `model[s]` indicates that this is a `model` statement and `model_name` is the internal name of the model in GAMS, it is an identifier. The optional explanatory text is used to describe the model, `all` is a keyword as introduced above and `eqn_name` is the name of an equation that has been declared prior to the model statement. `Var_name(set_name)` is a couple of previous declared variable and and set to limit the domain of variables in the model. More details about this are described in the following subsection. For advice on explanatory text and how to choose a `model_name`, see the tutorial Good Coding Practices.

Note

Model statements for Mixed Complementarity Problem (MCP) and Mathematical Program with Equilibrium Constraints (MPEC) models require a slightly different notation, since complementarity relationships need to be included. For details see subsections Mixed Complementarity Problem (MCP) and Mathematical Program with Equilibrium Constraints (MPEC).

An example of a model definition in GAMS is shown below.

```
Model transport "a transportation model" / all /;
```

The `model` is called `transport` and the keyword `all` is a shorthand for all known (declared) equations.

Several models may be declared (and defined) in one model statement. This is useful when experimenting with different ways of writing a model, or if one has different models that draw on the same data. Consider the following example, adapted from **[PROLOG]**, in which different groups of the equations are used in alternative versions of the problem. Three versions are solved: the linear, nonlinear, and 'expenditure' versions. The model statement to define all three is:

```
Model  nortonl   "linear version"      / cb,rc,dfl,bc,obj /
       nortonn   "nonlinear version"   / cb,rc,dfn,bc,obj /
       nortone   "expenditure version" / cb,rc,dfe,bc,obj /  ;
```

Here `cb`, `rc`, etc. are the names of the equations. We will describe below how to obtain the solution to each of the three models.

Note

If several models are declared and defined with one model statement, the models have to be separated by commas or linefeeds and a semicolon terminates the entire statement.

If several models are declared then it is possible to use one previously declared model in the declaration of another. The following examples illustrate this:

```
Model one   "first model"                          / tcost_eq, supply_eq, demand_eq /
      two   "second model that nests first"        / one, balance_eq /
      three "third model that nests first and second" / two, capacity_eq, configure_eq /;
```

Model one is declared and defined using the general syntax, model two contains all the equations of model one and the equation balance_eq, and model three contains all of model two and the equations capacity_eq and configure_eq.

In addition to nesting models as illustrated above, it is also possible to use the symbols + and - to augment or remove items relative to models that were previously defined. The following examples serve as illustration:

```
Model four "fourth model: model three minus model one"          / three-one /
      five "fifth model: model three without eqn configure_eq" / three-configure_eq /
      six  "sixth model: model four plus model two"            / four+two /;
```

Model four contains the equations from model three except for those that belong to model one. Model five contains all equations from model three except for equation configure_eq. Model six contains the union of the equations in model four and two. Note that both model names and equation names may be used in association with the symbols + and -.

Limited domain for variables

As mentioned above, it is possible to limit the domain of variables used in a model in the model statement. This allows to restrict the generation of blocks of variables in a single place instead of using, e.g., dollar conditions at every place where this variable block is used in equations (which might be required for an efficient model generation).

The following examples is based on the basic transportation model [**TRNSPORT**]. To limit the transportation network in that model to certain links (e.g. because some are blocked because of some reason) one could introduce a subset of the free links and use that with dollar conditions in the equations like this:

```
* Initialize whole network as free
Set freeLinks(i,j) Useable links in the network / #i.#j /;

cost..      z =e= sum((i,j), c(i,j)*x(i,j)$freeLinks(i,j));

supply(i).. sum(j, x(i,j)$freeLinks(i,j)) =l= a(i);

demand(j).. sum(i, x(i,j)$freeLinks(i,j)) =g= b(j);

* Block a particular link
freeLinks('san-diego','topeka') = no;

Model transport / all /;
solve transport using lp minimizing z;
```

Now, instead of adding the dollar condition to each appearance of x in the model, one could simply add a domain restriction for that variable to the model statement directly by specifying a variable and the set that limits its domain. Using this approach, the previous example looks like this:

```
* Initialize whole network as free
Set freeLinks(i,j) Useable links in the network / #i.#j /;

cost..      z =e= sum((i,j), c(i,j)*x(i,j));
```

```
supply(i).. sum(j, x(i,j)) =l= a(i);

demand(j).. sum(i, x(i,j)) =g= b(j);

* Block a particular link
freeLinks('san-diego','topeka') = no;

Model transport / all, x(freeLinks) /;
solve transport using lp minimizing z;
```

Note

If one adds the domain restriction to the model statement, internally GAMS inserts a dollar condition to every appearance of the restricted variables in equations of the model. When doing this, the indices are copied as they appear with the variable. So, in the example above, `x(i,j)` becomes `x(i,j)$freeLinks(i,j)`. In the same way `x(i-1,j+1)` becomes `x(i-1,j+1)$freeLinks(i-1,j+1)` and `x('seattle','chicago')` becomes `x('seattle','chicago')$freeLinks('seattle','chicago')`.

Attention

As a consequence of above's note one could see some unexpected results, like "division by zero errors", if it is not done carefully. For example, the following dummy model, will trigger such an error, since we sum over all `i`, but some `x` were excluded leaving a `0` as divisor:

```
Set i      / i1*i3 /
    sub(i) / i2    /;

Positive Variable x(i);
Variable          z;
Equation          obj;

obj.. z =e= sum(i, 1/x(i));
x.lo(i) = 1;
Model m / obj, x(sub) /;
solve m min z use nlp;
```

4.30.2.2 Classification of Models

Various types of problems can be solved with GAMS. Note that the type of the model must be known before it may be solved. The model types are briefly discussed in this section. GAMS checks that the model is in fact the type the user thinks it is, and issues explanatory error messages if it discovers a mismatch - for instance, that a supposedly linear model contains nonlinear terms. Some problems may be solved in more than one way, and the user has to choose which way to use. For instance, if there are binary or integer variables in the model, it can be solved either as a `MIP` or as a `RMIP`.

The model types and their identifiers, which are needed in the a solve statement, are given in Table 1. For details on the solve statement, see section The Solve Statement.

GAMS Model Type	Model Type Description	Requirements and Comments
LP	Linear Program	Model with no nonlinear terms or discrete (i.e. binary, integer, etc) variables.

GAMS Model Type	Model Type Description	Requirements and Comments
NLP	Nonlinear Program	Model with general nonlinear terms involving only *smooth* functions, but no discrete variables. For a classification of functions as to smoothness, see section Functions.
QCP	Quadratically Constrained Program	Model with linear and quadratic terms, but no general nonlinear terms or discrete variables.
DNLP	Discontinuous Nonlinear Program	Model with *non-smooth* nonlinear terms with discontinuous derivatives, but no discrete variables. This is the same as `NLP`, except that non-smooth functions may appear as well. These models are more difficult to solve than normal `NLP` models and we strongly advise not to use this model type.
MIP	Mixed Integer Program	Model with binary, integer, SOS and/or semi variables, but no nonlinear terms.
RMIP	Relaxed Mixed Integer Program	Like `MIP`, except that the discrete variable requirement is relaxed. See the note below on relaxed model types.
MINLP	Mixed Integer Nonlinear Program	Model with both nonlinear terms *and* discrete variables.
RMINLP	Relaxed Mixed Integer Nonlinear Program	Like `MINLP` except that the discrete variable requirement is relaxed. See the note below on relaxed model types.
MIQCP	Mixed Integer Quadratically Constrained Program	Model with both quadratic terms *and* discrete variables, but no general nonlinear term.
RMIQCP	Relaxed Mixed Integer Quadratically Constrained Program	Like `MIQCP` except that the discrete variable requirement is relaxed. See the note below on relaxed model types.
MCP	Mixed Complementarity Problem	A square, possibly nonlinear, model that generalizes a system of equations. Rows and columns are matched in one-to-one complementary relationships.
CNS	Constrained Nonlinear System	Model solving a square, possibly nonlinear system of equations, with an equal number of variables and constraints.
MPEC	Mathematical Programs with Equilibrium Constraints	A difficult model type for which solvers and reformulations are currently being developed.
RMPEC	Relaxed Mathematical Program with Equilibrium Constraints	A difficult model type for which solvers and reformulations are currently being developed. See the note below on relaxed model types.
EMP	Extended Mathematical Program	A family of mathematical programming extensions.

GAMS Model Type	Model Type Description	Requirements and Comments
MPSGE	General Equilibrium	Not actually a model type but mentioned for completeness, see MPSGE.

Table 1: GAMS Model Types

Note

- The relaxed model types RMIP, RMINLP, RMIQCP, and RMPEC solve the problem as the corresponding model type (e.g. MIP for RMIP) but *relax* the discrete requirement of the discrete variables. This means that integer and binary variables may assume any values between their bounds. SemiInteger and SemiCont variables may assume any values between 0 and their upper bound. For SOS1 and SOS2 variables the restriction of the number of non-zero values is removed.
- Many "LP" solvers like Cplex offer the functionality of solving *convex* quadratic models. So the Q matrices in the model need to be positive semidefinite. An extension to to this are the second-order cone programs (SOCP) with either symmetric or rotated cones. See the solver manuals (e.g. on MOSEK) for details.
- Unlike other checks on the model algebra (e.g. existence of discrete variables or general non-linear terms), the GAMS *compiler* does not enforce a quadratic model to only consist of quadratic and linear terms. This requirement is enforced at *runtime* for a particular model instance.

Linear Programming (LP)

Mathematically, the Linear Programming (LP) problem looks like:

$$\begin{array}{ll} \text{Minimize or maximize} & cx \\ \text{subject to} & Ax \;\alpha\; b \\ & L \leq x \leq U, \end{array}$$

where x is a vector of variables that are continuous real numbers, cx is the objective function, and $Ax\,\alpha\,b$ represents the set of constraints. Here, α is an equation operator. For details on the equation types allowed in GAMS, see Equation Types. L and U are vectors of lower and upper bounds on the variables.

GAMS supports free (unrestricted) variables, positive variables, and negative variables. Note that users may customize lower and upper bounds, for details see section Bounds on Variables.

For information on LP solvers that can be used through GAMS see the Solver/Model type Matrix.

Nonlinear Programming (NLP)

Mathematically, the Nonlinear Programming (NLP) problem looks like:

$$\begin{array}{ll} \text{Minimize or Maximize} & f(x) \\ \text{subject to} & g(x) \;\alpha\; 0 \\ & L \leq x \leq U, \end{array}$$

where x is a vector of variables that are continuous real numbers, $f(x)$ is the objective function, and $g(x)\,\alpha\,0$ represents the set of constraints. For details on the equation types allowed in GAMS, see Equation Types. Note that the functions $f(x)$ and $g(x)$ have to be differentiable. L and U are vectors of lower and upper bounds on the variables.

For information on NLP solvers that can be used through GAMS see the Solver/Model type Matrix. See also the tutorial Good NLP Formulations.

Note

NLP models may have the nonlinear terms inactive. In this case setting the model attribute TryLinear to 1 causes GAMS to check the model and use the default LP solver if possible. For details on model attributes, see subsection Model Attributes.

Quadratically Constrained Programs (QCP)

Mathematically, the Quadratically Constrained Programming (QCP) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & cx + x'Qx \\ \text{subject to} & A_i x + x'R_i x \ \alpha \ b_i \quad \text{for all } i \\ & L \leq x \leq U, \end{array}$$

where x denotes a vector of variables that are continuous real numbers, cx is the linear part of the objective function, $x'Qx$ is the quadratic part of the objective function, $A_i x$ represents the linear part of the ith constraint, $x'R_i x$ its quadratic part and b_i its right-hand side. For details on the equation types allowed in GAMS, see Equation Types. Further, L and U are vectors of lower and upper bounds on the variables.

Note that a QCP is a special case of the NLP in which all the nonlinearities are required to be quadratic. As such, any QCP model can also be solved as an NLP. However, most "LP" vendors provide routines to solve LP models with a quadratic objective. Some allow quadratic constraints as well. Solving a model using the QCP model type allows these "LP" solvers to be used to solve quadratic models as well as linear ones. Some NLP solvers may also take advantage of the special (quadratic) form when solving QCP models.

Attention

In case a model with quadratic *constraints* is passed to a QCP solver that only allows a quadratic *objective*, a capability error will be returned (`solver status 6 CAPABILITY PROBLEMS`). Some solvers will fail when asked to solve a non-convex quadratic problems as described above.

Note

Using the model attribute TryLinear causes GAMS to see if the problem can be solved as an LP problem. For details on model attributes, see subsection Model Attributes.

For information on QCP solvers that can be used through GAMS see the Solver/Model type Matrix.

Nonlinear Programming with Discontinuous Derivatives (DNLP)

Mathematically, the Nonlinear Programming with Discontinuous Derivatives (DNLP) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & f(x) \\ \text{subject to} & g(x) \ \alpha \ 0 \\ & L \leq x \leq U, \end{array}$$

where x is a vector of variables that are continuous real numbers, $f(x)$ is the objective function, $g(x)\,\alpha\,0$ represents the set of constraints, and L and U are vectors of lower and upper bounds on the variables. For details on the equation types allowed in GAMS, see Equation Types. Note that this is the same as NLP, except that non-smooth functions, like `abs`, `min`, `max` may appear in $f(x)$ and $g(x)$.

For information on DNLP solvers that can be used through GAMS see the Solver/Model type Matrix.

Attention

- We strongly advise against using the model type `DNLP`. The best way to model discontinuous functions is with binary variables, which results in a model of the type MINLP. The model **[ABSMIP]** demonstrates this formulation technique for the functions `abs`, `min`, `max` and `sign`. See also section Reformulating DNLP Models.
- Solvers may have difficulties when dealing with the discontinuities, since they are really NLP solvers and the optimality conditions and the reliance on derivatives may be problematic. Using a global solver may alleviate this problem.

Mixed Integer Programming (MIP)

Mathematically, the Mixed Integer Linear Programming (MIP) problem looks like:

$$
\begin{array}{lrcllll}
\text{Maximize or Minimize} & c_1t + c_2u + c_3v + c_4w + c_5x + c_6y + c_7z & & & & & \\
\text{subject to} & A_1t + A_2u + A_3v + A_4w + A_5x + A_6y + A_7z & \alpha & b & & & \\
& t & \in & \mathbb{R} & & & \\
& u & \geq & 0 & \text{and } u \leq L_2 & \text{and } u \in \mathbb{Z} & \\
& v & \in & (0,1) & & & \\
& w & \in & \text{SOS1} & & & \\
& x & \in & \text{SOS2} & & & \\
& y & = & 0 & \text{or } L_6 \leq y & & \\
& z & = & 0 & \text{or } L_7 \leq z & \text{and } z \in \mathbb{Z}, &
\end{array}
$$

where

- $c_1t + c_2u + c_3v + c_4w + c_5x + c_6y + c_7z$ is the objective function,
- $A_1t + A_2u + A_3v + A_4w + A_5x + A_6y + A_7z \ \alpha \ b$ represents the set of constraints of various equality and inequality forms,
- t is a vector of variables that are continuous real numbers,
- u is a vector of variables that can only take integer values smaller than L_2,
- v is a vector of binary variables,
- w is a vector of variables that belong to `SOS1` sets; this means that at most one variable in the set is nonzero,
- x is a vector of variables that belong to `SOS2` sets; this means that at most two adjacent variables in the set are nonzero,
- y is a vector of variables that are semi-continuous; they are either zero or larger than L_6,
- z is a vector of variables that are semi-integer; they are integer and either zero or larger than L_7.

For details on the equation types allowed in GAMS, see Equation Types. For more details on MIPs in GAMS, especially the use of `SOS` and `semi` variables, see section Special Mixed Integer Programming (MIP) Features.

For information on MIP solvers that can be used through GAMS, see the Solver/Model type Matrix.

Attention

> Not all MIP solvers cover all the cases associated with `SOS` and `semi` variables. Please consult the solver manuals for details on capabilities.

Mixed Integer Nonlinear Programming (MINLP)

Mathematically, the Mixed Integer Nonlinear Programming (MINLP) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & f(x) + Dy \\ \text{subject to} & g(x) + Hy \ \alpha \ 0 \\ & L \leq x \leq U \\ & y = \{0, 1, 2, \cdots\}, \end{array}$$

where x is a vector of variables that are continuous real numbers, y denotes a vector of variables that can only take integer values, $f(x)+Dy$ is the objective function, $g(x)+Hy \ \alpha \, 0$ represents the set of constraints, and L and U are vectors of lower and upper bounds on the variables. For details on the equation types allowed in GAMS, see Equation Types. Further, $y = \{0, 1, 2, \cdots\}$ is the integrality restriction on y.

For information on MINLP solvers that can be used through GAMS see the Solver/Model type Matrix.

Note

- `SOS` and `semi` variables can also be accommodated by some solvers. Please consult the solver manuals for details on capabilities.
- The model attribute TryLinear causes GAMS to examine whether the problem may be solved as a MIP problem. For details on model attributes, see subsection Model Attributes.

Mixed Integer Quadratically Constrained Programs (MIQCP)

A Mixed Integer Quadratically Constrained Program (MIQCP) is a special case of the MINLP in which all the nonlinearities are required to be quadratic. For details see the description of the QCP, a special case of the NLP.

For information on MIQCP solvers that can be used through GAMS, see the Solver/Model type Matrix.

Note

The model attribute TryLinear causes GAMS to examine whether the problem may be solved as a MIP problem. For details on model attributes, see subsection Model Attributes.

Mixed Complementarity Problem (MCP)

Unlike the other model types we have introduced so far, the Mixed Complementarity Problem (MCP) does not have an objective function. An MCP is specified by three pieces of data: a function $F(z) : \mathbb{R}^n \mapsto \mathbb{R}^n$, lower bounds $l \in \{\mathbb{R} \cup \{-\infty\}\}^n$ and upper bounds $u \in \{\mathbb{R} \cup \{\infty\}\}^n$. A solution is a vector $z \in \mathbb{R}^n$ such that for each $i \in \{1, \ldots, n\}$, one of the following three conditions hold:

$$\begin{array}{lll} F_i(z) = 0 & \text{and} & \ell_i \leq z_i \leq u_i \quad \text{or} \\ F_i(z) > 0 & \text{and} & z_i = \ell_i \quad \text{or} \\ F_i(z) < 0 & \text{and} & z_i = u_i. \end{array}$$

This problem can be written compactly as

$$F(z) \ \perp \ L \leq z \leq U,$$

where the symbol $\perp$ (which means "perpendicular to", shortened to "perp to") indicates pair-wise complementarity between the function F and the variable z and its bounds.

The following special case is an important and illustrative example:

$$F(z) \perp z \geq 0.$$

In this example, the unstated but implied upper bound u is infinity. Since z is finite, we cannot have $z_i = u_i$ and the third condition above cannot hold: this implies $F(z) >= 0$. The remaining two conditions imply pair-wise complementarity between $z >= 0$ and $F(z) >= 0$. This is exactly the Nonlinear Complementarity Problem, often written as

$$F(z) \geq 0, \quad z \geq 0, \quad \langle F(z), z \rangle = 0.$$

None of this rules out the degenerate case (i.e. $F_i(z)$ and z_i both zero). In practice, these can be difficult models to solve.

Another special case arises when the bounds L and U are infinite. In this case, the second and third conditions above cannot hold, so we are left with $F(z) = 0$, a square system of nonlinear equations. And finally, we should mention a special case that occurs frequently in practice: if $\ell_i = u_i$ (i.e. z_i is fixed) then we have a complementary pair: one of the three conditions will hold as long as $F_i(z)$ is defined. Essentially, fixing a variable removes or obviates the matching equation. This is often useful when modeling with MCP.

The definition above describes the canonical MCP model as it exists when GAMS passes it to an MCP solver. Some models have exactly this form even in the GAMS code, but usually some processing is done by the GAMS system to arrive at a model in this form. Here we'll describe the steps of this process and illustrate with an example from the model library.

1. The process starts with the list of rows (aka single equations) and columns (aka single variables) that make up the MCP model, and potentially some matching information.
   - The usual rules apply: rows are part of the model because their associated equations are included in the model statement, but columns only become part of the model by use: a column enters the model only if it is used in some row of the model. Therefore including a variable symbol as part of a match in the model statement will not influence the set of columns belonging to the model.
   - Matches (where they exist) are pointers from rows to columns.
   - Technically, the MCP is defined via a function F while a model contains constraints. Given a constraint, we define an associated function as LHS - RHS, so e.g. $F_i \geq 0$ is consistent with a `=G=` constraint.
2. The explicit matches are processed: each match creates a complementary pair. What remains after the explicit matches are consumed are the unmatched rows and unmatched columns.
   - It is an error for any column to be matched to multiple rows, so the row-column matching is one-to-one.
   - For each match some consistency checks between the column bounds and the row type are made. For details, see Table 2.
   - For example, matching an =N= row with any column is good, matching an =E= row with a free column is good, matching an =E= row with a lower-bounded column is allowed, and matching a =G= row with an upper-bounded column results in an error.
3. Any fixed columns remaining are ignored: these columns can be treated like exogenous variables or parameters.
4. If what remains is a set of =E= rows and an *equal number* of unbounded columns, these can be matched up in any order and we have a well-defined MCP. If this is not what remains, an error is triggered.

To illustrate how this works, consider the spatial equilibrium model **[SPATEQU]** with the following model statement:

```
Model  P2R3_MCP        / dem, sup, in_out.p, dom_trad.x /;
```

1. The model P2R3_MCP includes the rows from equations dem, sup, in_out and dom_trad and exactly the columns used by these rows. Checking the listing file, we see columns for Qd, Qs, x, and p. In addition, the model statement specifies two matches: in_out.p and dom_trad.x. These matches always take the form of an equation.variable pair, with no indices or domains included.
2. In this example, the rows corresponding to the equation in_out match up perfectly with the columns from the variable p: there are no holes in the set of rows or columns because of some dollar conditions in the equation definition. We have a one-to-one match so all the rows of in_out and columns of p are consumed by the match in the model statement. The same holds for the dom_trad.x pair, so what is left are the rows of dem and sup and the columns of Qd and Qs, all of which are unmatched.
3. There are no fixed variables to remove.
4. Since dem and sup are =E= constraints and Qd and Qs are free variables, we can match them in any order without changing the solution set for this model. The counts of these unmatched equality rows and unmatched free variables are equal, so we get a well-defined MCP.

When rows are matched explicitly to columns, some care must be taken to match them consistently. For example, consider a row-column match *g.y*. The row g can be of several types: =N=, =E=, =G=, or =L=. An =N= row can be matched to any sort of variable: the =N= doesn't imply any sort of relationship, which works perfectly with our definition of $\perp$ above: the allowed sign or direction of g is determined completely by the bounds on the complementary variable y. If g is an =E= row, this is consistent with a free variable y, but what if y has an active lower bound? By definition we allow g to be positive at solution, but this violates the declaration as an =E= row. Such cases can be handled by marking the row with a ***redef***. The total number of redefs for a given model is available via the NumRedef model attribute and is shown in the report summary. Note that the set of rows marked depends on the solution: in the example above, if g is zero at solution it will not be marked as a redef, regardless of what the bounds are on y. Finally, some combinations are simply not allowed: they will result in a model generation error. The table below lists the outcome for all possible combinations.

Table 2: ***MCP Matching***

Column Bounds	=N=	=E=	=G=	=L=
lower	OK	redef	OK	ERROR
upper	OK	redef	ERROR	OK
free	OK	OK	OK	OK
double	OK	redef	redef	redef
fixed	OK	redef	redef	redef

The definition, process, and rules above have several implications for valid MCP models:

- It is always acceptable to use the =N= notation when defining the equations in an MCP model, *provided these equations are matched explicitly*. In this case the bounds on F(z) are implied by the bounds on the matching columns, and redefs will never occur.
- Variables that are known to be lower-bounded (no upper bound) will match consistently with =G= equations.
- Variables that are known to be upper-bounded (no lower bound) will match consistently with =L= equations.
- Variables that are known to be unbounded will match consistently with =E= equations.
- Where the bound structure is not known in advance, or both upper and lower bounds exist, a match with an =N= equation will always be consistent. Other equation types will result in errors or redefs.

- The model may initially have fewer rows than columns, as long as the "extra" columns are unmatched fixed columns that ultimately get removed from the MCP passed to the solver.
- Any bounded-but-not-fixed column must be matched explicitly to a row.
- The only rows that may be unmatched are =E= rows.
- It is customary to re-use the constraints of an LP or NLP model when formulating the MCP corresponding to the Karush-Kuhn-Tucker (KKT) conditions. If the original model is a *minimization*, the LP/NLP marginals `.m` and the variables for these marginals in the MCP will use the same sign convention, and the orientation for the constraints will be consistent between the two models, making re-use easier.

As mentioned above, it is typical to use the same equations in both NLP and MCP models. Sometimes, it is not the original equation that is wanted for the MCP, but rather the reoriented (aka negated or flipped) equation. For example, the flipped version of `x**1.5 =L= y` is `y =G= x**1.5`, while `sqr(u) - sqr(v) =E= 5` becomes `- sqr(u) + sqr(v) =E= -5`. Instead of re-implementing the equation in flipped form, the same result can be achieved by prefixing the equation name with a `-` in the model statement. See the **[mcp10]** model for an example of such usage. When equations are used in flipped form, they are marked with a ***redir*** in the listing file's solution listing.

An example of complementarity that should be familiar to many is the relationship between a constraint and its associated dual multiplier: if the constraint is non-binding, its dual multiplier must be zero (i.e. at bound) while if a dual multiplier is nonzero the associated constraint must be binding. In fact, the KKT or first-order optimality conditions for LP and NLP models can be expressed and solved as an MCP.

These complementarity relationships found in optimization problems are useful in understanding the marginal values assigned to rows and columns in the GAMS solution for MCP. With no objective function, the usual definition for marginal values and their interpretation isn't useful. Instead, the GAMS MCP convention for the marginal values of columns is to return the slack of the associated row (i.e. its value when interpreted and evaluated as a function). For the marginal values of rows, the level value (*not* the slack) of the associated column is returned. When we apply this convention to the NCP ($F(z) \geq 0, z \geq 0, \langle F(z), z \rangle = 0$) we see pairwise complementarity between the levels and marginals returned for each of the rows and columns in the model. This is also the case if we take the KKT conditions of an LP in a suitable standard form: minimization, $x \geq 0, Ax \geq b$.

MCPs arise in many application areas including applied economics, game theory, structural engineering and chemical engineering. For further details on this class of problems, see http://www.neos-guide.org/content/complementarity-problems.

For information on MCP solvers that can be used through GAMS, see Solver/Model type Matrix.

Constrained Nonlinear System (CNS)

The Constrained Nonlinear System (CNS) is the second GAMS model type that does not have an objective function. Mathematically, a CNS model looks like:

$$\begin{array}{ll} \text{Find} & x \\ \text{subject to} & F(x) = 0 \\ & L \leq x \leq U \\ & G(x) \ \alpha \ b, \end{array} \tag{4.1}$$

where x is a set of continuous variables and F is a set of nonlinear equations of the same dimension as x. This is a key property of this model type: the number of equations equals the number of variables, so we have a square system. The (possibly empty) constraints $L \leq x \leq U$ are not intended to be binding at the solution, but instead are included to constrain the solution to a particular domain, to avoid regions where $F(x)$ is undefined, or perhaps just to give the solver a push in the right direction. The (possibly

empty) constraints $G(x)\ \alpha\ b$ are intended to serve the same purpose as the variable bounds and are silently converted to equations with bounded slacks.

Note that since there is no objective in a CNS model, there are no marginal values for variables and equations. Any marginal values already stored in the GAMS database will remain untouched. CNS models also make use of some model status values that allow a solver to indicate if the solution is unique (e.g. for a non-singular linear system) or if the linearization is singular at the solution. For singular models (solved or otherwise), the solver can mark one or more dependent rows with a ***depnd***. The total number of rows so marked for a given model is available via the NumDepnd model attribute and is shown in the report summary.

The CNS model is a generalization of a square system of equations $F(x) = 0$. Such a system could also be modeled as an NLP with a dummy objective. However, there are a number of advantages to using the CNS model type, including:

- A check by GAMS that the model is really square,
- solution/model diagnostics by the solver (e.g. singular at solution, (locally) unique solution),
- CNS-specific warnings if the side constraints $L \leq x \leq U$ or $G(x)\ \alpha\ b$ are active at a solution,
- and potential improvement in solution times, by taking better advantage of the model properties.

For information on CNS solvers that can be used through GAMS, see the Solver/Model type Matrix.

Mathematical Program with Equilibrium Constraints (MPEC)

Mathematically, the Mathematical Program with Equilibrium Constraints (MPEC) problem looks like:

$$\begin{array}{ll} \text{Maximize or Minimize} & f(x,y) \\ \text{subject to} & g(x,y)\ \alpha\ 0 \\ & L_x \leq x \leq U_x \\ & F(x,y) \perp L_y \leq y \leq U_y, \end{array}$$

where x and y are vectors of continuous real variables. The variables x are often called the *control* or *upper-level* variables, while the variables y are called the *state* or *lower-level* variables. $f(x,y)$ is the objective function. $g(x,y)\,\alpha\,0$ represents the set of traditional (i.e. NLP-type) constraints; some solvers may require that these constraints only involve the control variables x. The function $F(x,y)$ and the bounds L_y and U_y define the equilibrium constraints. If x is fixed, then $F(x,y)$ and the bounds L_y and U_y define an MCP; the discussion of the "perp to" symbol $\perp$ in that section applies here as well. From this definition, we see that the MPEC model type contains NLP and MCP models as special cases of MPEC.

A simple example of an entire MPEC model is given below.

```
variable z, x1, x2, y1, y2;
positive variable y1;
y2.lo = -1;
y2.up =  1;

equations cost, g, h1, h2;

cost..  z =E= x1 + x2;
g..     sqr(x1) + sqr(x2) =L= 1;
h1..    x1 =G= y1 - y2 + 1;
h2..    x2 + y2 =N= 0;

model example / cost, g, h1.y1, h2.y2 /;
solve example using mpec min z;
```

Note that as in the MCP, the complementarity relationships in an MPEC are specified in the model statement via equation-variable pairs: the `h1.y1` specifies that the equation `h1` is perpendicular to the variable `y1` and the `h2.y2` specifies that the equation `h2` is perpendicular to the variable `y2`. For details on the solve statement, see section The Solve Statement.

While the MPEC model formulation is very general, it also results in problems that can be very difficult to solve. The state-of-the-art for MPEC solvers is not nearly as advanced as that for other model types. As a result, you should expect the MPEC solvers to be more limited by problem size and/or robustness issues than solvers for other model types.

For information on MPEC solvers that can be used through GAMS, see the Solver/Model type Matrix. For more details on MPECs and solver development, see `https://neos-guide.org/guide/types/mpec/`.

Extended Mathematical Programs (EMP)

Extended Mathematical Programming (EMP) is an (experimental) framework for automated mathematical programming reformulations. Using EMP, model formulations that GAMS cannot currently handle directly or for which no robust and mature solver technology exists can be automatically and reliably reformulated or transformed into models for which robust and mature solver technology does exist within the GAMS system. For more details, see the chapter on EMP. Currently EMP supports:

- Equilibrium problems including variational inequalities, Nash games, and Multiple Optimization Problems with Equilibrium Constraints (MOPECs).
- Hierarchical optimization problems such as bilevel programs.
- Disjunctive programs for modeling discrete choices with binary variables.
- Stochastic programs including two-stage and multi-stage stochastic programs, chance constraints and risk measures such as Variance at Risk (VaR) and Conditional Variance at Risk (CVaR).

Apart from the disjunctive and stochastic programming models mentioned above, EMP models are typically processed (aka solved) via the JAMS solver: this solver does the work of reformulation/transformation, calling GAMS to solve this reformulation, and post-processing the solution that results to bring it back in terms of the original EMP model.

Examples demonstrating how to use the EMP framework and the JAMS and DE solvers are available in the GAMS **EMP Library**. These solvers require no license of their own to run but can and do call subsolvers that *do* require a license.

4.30.2.3 Model Attributes

Models have attributes that hold a variety of information, including

- information about the results of a solve performed, a solve statement, the solution of a model,
- information about certain features to be used by GAMS or the solver,
- information passed to GAMS or the solver specifying various settings that are also subject to option statements.

Model attributes are accessed in the following way:

```
model_name.attribute
```

Here `model_name` is the name of the model in GAMS and `.attribute` is the specific attribute that is to be accessed. Model attributes may be used on the left-hand side and the right-hand side of assignments. Consider the following example:

```
transport.resLim = 600;
x = transport.modelStat;
```

In the first line the attribute `.resLim` of the model `transport` is specified to be 600 (seconds). In the second line the value of the attribute `.modelStat` of the model `transport` is assigned to the scalar `x`. Note that model attributes may also be used in display statements.

Some of the attributes are mainly used before the solve statement to provide information to GAMS or the solver link. Others are set by GAMS or the solver link and hence are mainly used after a solve statement.

Moreover, some of the attributes used before the solve may also be set via an option statement or the command line. Consider the following example:

```
option ResLim=10;
```

This line is an option statement and applies to all models. One can set the model attribute `.ResLim` to overwrite the global `ResLim` option. In order to revert the individual `.ResLim` to the global `ResLim` option, one needs to set the model attribute to `NA`. For more on option statements, see chapter The Option Statement.

```
gams mymodel ResLim=10
```

This sets the global `ResLim` option when invoking the gams run (e.g. from the command line). For more on command line parameters, see chapter The GAMS Call and Command Line Parameters.

Note that a model-specific option takes precedence over the global setting specified with an option statement and that a setting via an option statement takes precedence over a setting via the command line parameter.

The complete list of model attributes is given below. Observe that each entry is linked to a detailed description of the respective attribute, including information of whether the attribute is also available as command line parameter or option statement. Note that detailed descriptions of all GAMS command line parameters, options and model attributes are given in section Detailed Descriptions of All Options.

Model Attributes Mainly Used Before Solve

Attribute	Description
bRatio	Basis detection threshold
cheat	Cheat value, i.e. minimum solution improvement threshold
cutOff	Cutoff value for branch and bound
defPoint	Indicator for passing on default point
dictFile	Force writing of a dictionary file if dictfile > 0
domLim	Domain violation limit solver default
fdDelta	Step size for finite differences
fdOpt	Options for finite differences
holdFixed	Treat fixed variables as constants
integer1..5	Integer communication cells
iterLim	Iteration limit of solver

Attribute	Description
limCol	Maximum number of columns listed in one variable block
limRow	Maximum number of rows listed in one equation block
MCPRHoldFx	Print list of rows that are perpendicular to variables removed due to the holdfixed setting
nodLim	Node limit in branch and bound tree
optCA	Absolute Optimality criterion solver default
optCR	Relative Optimality criterion solver default
optFile	Default option file
priorOpt	Priority option for variable attribute .prior
real1..5	Real communication cells
reform	Reformulation level
resLim	Wall-clock time limit for solver
savePoint	Save solver point in GDX file
scaleOpt	Employ user specified variable and equation scaling factors
solPrint	Solution report print option
solveLink	Solver link option
solveOpt	Multiple solve management
sysOut	Solver Status file reporting option
threads	Number of processors to be used by a solver
tolInfeas	Infeasibility tolerance for an empty row of the form a.. 0∗x =e= 0.0001;
tolInfRep	This attribute sets the tolerance for marking infeasible in the equation listing
tolProj	Tolerance for setting solution values to a nearby bound when reading a solution
tryInt	Whether solver should make use of a partial integer-feasible solution
tryLinear	Examine empirical NLP model to see if there are any NLP terms active. If there are none the default LP solver will be used
workFactor	Memory Estimate multiplier for some solvers
workSpace	Work space for some solvers in MB

Model Attributes Mainly Used After Solve

Attribute	Description
domUsd	Number of domain violations
etAlg	Solver dependent timing information
etSolve	Elapsed time it took to execute a solve statement in total
etSolver	Elapsed time taken by the solver only
handle	Unique handle number of SOLVE statement
iterUsd	Number of iterations used
line	Line number of last solve of the corresponding model
linkUsed	Integer number that indicates the value of SolveLink used for the last solve
marginals	Indicator for marginals present
maxInfes	Maximum of infeasibilities
meanInfes	Mean of infeasibilities
modelStat	Integer number that indicates the model status
nodUsd	Number of nodes used by the MIP solver
number	Model instance serial number
numDepnd	Number of dependencies in a CNS model
numDVar	Number of discrete variables
numEqu	Number of equations

Attribute	Description
numInfes	Number of infeasibilities
numNLIns	Number of nonlinear instructions
numNLNZ	Number of nonlinear nonzeros
numNOpt	Number of nonoptimalities
numNZ	Number of nonzero entries in the model coefficient matrix
numRedef	Number of MCP redefinitions
numVar	Number of variables
numVarProj	Number of bound projections during model generation
objEst	Estimate of the best possible solution for a mixed-integer model
objVal	Objective function value
procUsed	Integer number that indicates the used model type
resCalc	Time spent in function and derivative calculations (deprecated)
resDeriv	Time spent in derivative calculations (deprecated)
resGen	Time GAMS took to generate the model in wall-clock seconds
resIn	Time to import model (deprecated)
resOut	Time to export solution (deprecated)
resUsd	Time the solver used to solve the model in seconds
rngBndMax	Maximum absolute non-zero value of bounds passed to the solver (excluding infinity)
rngBndMin	Minimum absolute non-zero value of bounds passed to the solver
rngMatMax	Maximum absolute non-zero value of coefficients in the model matrix passed to the solver (excluding infinity)
rngMatMin	Minimum absolute non-zero value of coefficients in the model matrix passed to the solver
rngRhsMax	Maximum absolute non-zero value of right hand sides passed to the solver (excluding infinity)
rngRhsMin	Minimum absolute non-zero value of right hand sides passed to the solver
rObj	Objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish
solveStat	Indicates the solver termination condition
sumInfes	Sum of infeasibilities
sysIdent	Solver identification number
sysVer	Solver version

4.30.3 The Solve Statement

Once a model has been defined using the model statement, the solve statement prompts GAMS to call one of the available solvers for the particular model type. This section introduces and discusses the solve statement in detail. For a list of GAMS model types, see Table 1. For information on how to specify desired solvers, see section Choosing a Solver.

Note

It is important to remember that GAMS does *not* solve the problem, but passes the problem definition to one of a number of separate solver programs that are integrated with the GAMS system.

4.30.3.1 The Syntax of the Solve Statement

In general, the syntax for a `solve` statement is as follows. Note that there are two alternatives that are equally valid:

```
solve model_name using model_type maximizing|minimizing var_name;
solve model_name maximizing|minimizing var_name using model_type ;
```

The keyword `solve` indicates that this is a `solve` statement. `Model_name` is the name of the model as defined by a `model` statement. Note that the `model` statement must be placed before the `solve` statement in the program. The keyword `using` is followed by `model_type`, which is one of the GAMS model types described above, see Table 1. The keywords `maximizing` or `minimizing` indicate the direction of the optimization. `Var_name` is the name of the objective variable that is being optimized. An example of a `solve` statement in GAMS is shown below.

```
Solve transport using lp minimizing cost ;
```

`Solve` and `using` are reserved words, `transport` is the name of the model, `lp` is the model type, `minimizing` is the direction of optimization, and `cost` is the objective variable. Note that an objective variable is used instead of an objective row or function.

Attention

> The objective variable must be scalar and of type `free`, and must appear in at least one of the equations in the model.

Recall that some model types (e.g. the Constrained Nonlinear System (CNS) or the Mixed Complementarity Problem (MCP)) do not have an objective variable. So their `solve` statement syntax is slightly different:

```
solve model_name using model_type;
```

As before, `solve` and `using` are keywords, `model_name` is the name of the model as defined by a `model` statement and `model_type` is the GAMS model type CNS or MCP. There is no objective variable and consequently no direction of optimization. An example from the spatial equilibrium model **[SPATEQU]** illustrates this `solve` statement:

```
Solve P2R3_MCP using mcp;
```

`P2R3_MCP` is the model name, the model type is `MCP` and as expected, there is no objective variable.

The EMP model type serves many purposes including some experimental ones. The solve statement with model type EMP can be with or without the objective variable and optimization direction. For more information, see chapter Extended Mathematical Programming (EMP).

4.30.3.2 Actions Triggered by the Solve Statement

When GAMS encounters a solve statement during compilation (the syntactic check of the input file) or execution (actual execution of the program), it initiates a number of special actions. The purpose is to prevent waste that would be caused by solving a model that has apparently been incorrectly specified. During compilation the following are verified:

1. All symbolic equations have been defined and the objective variable is used in at least one of the equations.
2. The objective variable is scalar and of type `free` (even though lower and upper bounds may have been specified)
3. `MCP` models are checked for appropriate complementarity and squareness.
4. Each equation fits into the specified problem class (linearity for `LP`, continuous derivatives for `NLP`, as outlined above).
5. All sets and parameters in the equations have values assigned.

Note

GAMS issues explanatory error messages if it discovers that the model is not according to type; for example, the presence of nonlinear terms in a supposedly LP model. For details on error messages, see chapter GAMS Output.

At execution time the solve statement triggers the following sequence of steps:

1. The model is translated into the representation required by the solution system to be used.
2. Debugging and comprehension aids that the user wishes to see are produced and written to the output file (`EQUATION LISTING`, etc). For customizing options (e.g. LimRow and LimCol), see chapter The Option Statement.
3. GAMS verifies that there are no inconsistent bounds or unacceptable values (for example, `NA` or `UNDF`) in the problem.
4. Any errors detected at this stage cause termination with as much explanation as possible, using the GAMS names for the identifiers causing the trouble.
5. GAMS designs a solution strategy based on the possible availability of level values or basis information from a previous solution: all available information is used to provide efficiency and robustness of operation. Any specifications provided by the user (Iteration limits etc.) are incorporated. A solver is chosen which is either the default solver for that problem type, the solver specified on the command line or the solver chosen by an option statement. For details see section Choosing a Solver.
6. GAMS passes control to the solution subsystem and waits while the problem is being solved.
7. GAMS reports on the status of the solution process and loads solution values back into the GAMS database. This causes new values to be assigned to the `.l` and `.m` fields for all individual equations and variables in the model. In addition, the post solution model attributes are assigned. The procedure for loading back the data associated with level and marginal values may be customized using the SolveOpt model attribute and option. A row by row and column by column listing of the solution is provided by default. It may be suppressed by the SolPrint model attribute or option. Any apparent difficulty with the solution process will cause explanatory messages to be displayed. Errors caused by forbidden nonlinear operations are reported at this stage.

Note

When the solver does not provide a dual solution (.m), then GAMS does not print the marginal column in the solution listing and set the marginal field in variables and equations to NA.

The outputs from these steps, including any possible error messages, are discussed in detail in chapter GAMS Output.

4.30.4 Programs with Several Solve Statements

Several solve statements can be processed in the same program. The next few subsections discuss various instances where several `solve` statements may be needed in the same file. If sequences of expensive or difficult models are to be solved, it might be useful to interrupt program execution and continue later. For details on this topic, see chapter The Save and Restart Feature.

4.30.4.1 Several Models

If there are different models then the solves may be sequential, as below. Each of the models in [PROLOG] consists of a different set of equations, but the data are identical, so the three solves appear in sequence with no intervening assignments:

```
Solve nortonl using nlp maximizing z;
Solve nortonn using nlp maximizing z;
Solve nortone using nlp maximizing z;
```

When there is more than one `solve` statement in the program, GAMS uses as much information as possible from the previous solution to provide a starting point or basis in the search for the next solution.

4.30.4.2 Loop: One Model, Different Data

Multiple solves may also occur as a result of a solve statement within a `loop` statement. Loop statements are introduced and discussed in detail in chapter Programming Flow Control Features; here we show that they may contain a solve statement and thus lead to multiple solves within one model. The example from [**MEANVAR**] computes the efficient frontier for return and variance for a portfolio selection problem at equidistance points.

```
loop(p(pp),
   v.fx = vmin + (vmax-vmin)/(card(pp)+1)*ord(pp) ;
   Solve var1 maximizing m using nlp ;
   xres(i,p)         = x.l(i);
   xres('mean',p)    = m.l;
   xres('var',p)     = v.l;
);
```

The set `p` is a set of point between the minimum and maximum variance, it is the driving set of the loop. A variance variable v is fixed at a equidistance points. With each iteration through the loop another variance level is used, the `NLP` model var1 is solved for each iteration and the outputs are stored in the parameter `xres(*,pp)`, to be used later for reporting. As often for reporting purposes, the universal set $*$ is used.

This example demonstrates how to solve the same model (in terms of variables and equations) multiple times with slightly different data. For such situations the Gather-Update-Solve-Scatter (GUSS) facility improves on the loop implementation by saving generation time and minimizing the communication with the solver. GUSS is activated by the additional keyword `scenario` in the solve statement followed by a set name that provides mapping information between parameters in the model and the scenario containers. A GUSS implementation of the loop would look as follows:

```
parameter vfx(p), px(p,i), pm(p);
set dict / p  .scenario.''
           v  .fixed   .vfx
           x  .level   .px
           m  .level   .pm /;
vfx(p(pp)) = vmin + (vmax-vmin)/(card(pp)+1)*ord(pp);
Solve var1 maximizing m using nlp scenario dict;
xres(i,p)      = px(p,i);
xres('mean',p) = pm(p);
xres('var',p)  = vfx(p);
```

4.30.4.3 Customizing Solution Management: SolveOpt

It is important to consider how GAMS manages solutions if multiple models are solved. By default, GAMS merges subsequent solutions with prior solutions. This is not an issue if all models operate over the same set of variables. However, recursive procedures, different equation inclusions or logical conditions may cause only part of the variables or different variables to appear in the models to be solved. In such a case it might be useful to modify the solution management procedure using the model attribute or option SolveOpt.

4.30.4.4 Sensitivity or Scenario Analysis

Multiple solve statements can be used not only to solve different models, but also to conduct sensitivity tests, or to perform case (or scenario) analyses of models by changing data or bounds and then solving the same model again. While some commercial LP systems allow access to "sensitivity analysis" through GAMS it is possible to be far more general and not restrict the analysis to either solver or model type. This facility is even more useful for studying many scenarios since no commercial solver will provide this information.

An example of sensitivity testing is in the simple oil-refining model [**MARCO**]. Because of pollution control, one of the key parameters in oil refinery models is an upper bound on the sulfur content of the fuel oil produced by the refinery. In this example, the upper bound on the sulfur content of fuel oil was set to 3.5 percent in the original data for the problem. First, the model is solved with this value. Next, a slightly lower value of 3.4 percent is used and the model is solved again. Finally, the considerably higher value of 5 percent is used and the model is solved for the last time. Key solution values are saved for later reporting after each solve. This is necessary because a following solve replaces any existing values. The key solution values are the activity levels of the process level `z`, a variable that is defined over a set of processes `p` and a set of crude oils `cr`. The complete sequence is:

```
parameter report(*,*,*) "process level report";

qs('upper','fuel-oil','sulfur') = 3.5  ;
Solve oil using lp maximizing phi;
report(cr,p,'base')  =  z.l(cr,p) ;
report('sulfur','limit','base') = qs('upper','fuel-oil','sulfur');

qs ('upper','fuel-oil','sulfur') = 3.4  ;
Solve oil using lp maximizing phi ;
report(cr,p,'one') = z.l(cr,p) ;
report('sulfur','limit','one') = qs ('upper','fuel-oil','sulfur');

qs('upper','fuel-oil','sulfur') = 5.0  ;
Solve oil using lp maximizing phi ;
report(cr,p,'two') = z.l(cr,p) ;
report('sulfur','limit','two') = qs('upper','fuel-oil','sulfur');

Display report ;
```

Note that the parameter `report` is defined over the universal set or short *universe*. In general, the universe is useful when generating reports, otherwise it would be necessary to provide special sets containing the labels used in the report. Any mistakes made in spelling labels used only in the report should be immediately apparent, and their effects should be limited to the report. The parameter `qs` is used to set the upper bound on the `sulfur` content in the `fuel-oil`, and the value is retrieved for the report. Note that the `display` statement in the final line is introduced and discussed in detail in chapter The Display Statement. This example shows not only how simply sensitivity analysis can be done, but also how the associated multi-case reporting can be handled.

The output from the `display` statement is shown below. Observe that there is no production at all if the permissible sulfur content is lowered. The *case attributes* have been listed in the row `SULFUR.LIMIT`. Section Global Display Controls contains more details on how to arrange reports in a variety of ways.

```
----    225 PARAMETER report  process level report

                          base         one         two

mid-c .a-dist           89.718                   35.139
mid-c .n-reform         20.000                    6.772
mid-c .cc-dist           7.805                    3.057
w-tex .cc-gas-oil                                 5.902
w-tex .a-dist                                    64.861
w-tex .n-reform                                  12.713
w-tex .cc-dist                                    4.735
w-tex .hydro                                     28.733
sulfur.limit             3.500       3.400        5.000
```

Note

For other ways to do comparative analyses with GAMS, see the tutorial Comparative Analyses with GAMS.

4.30.4.5 Iterative Implementation of Non-Standard Algorithms

Another use of multiple solve statements is to permit iterative solution of different blocks of equations, most often using solution values from the first solve as data for the next solve. These decomposition methods are useful for certain classes of problems because the subproblems being solved are smaller, and therefore more tractable. One of the most common examples of such a method is the Dantzig-Wolfe decomposition.

An example of a problem that is solved in this way is a multi-commodity network flow problem in [**DANWOLFE**].

4.30.5 Choosing a Solver

After a model has been checked and prepared as described above, GAMS passes the model to a solver. When the GAMS system is installed default solvers for all model types are specified and these solvers are used if the user doesn't specify anything else. It is easy to switch to other appropriate solvers provided the user has the corresponding license. There are multiple ways to switch solvers:

1. Using a command line parameter of the following form:

   ```
   gams mymodel model_type=solver name
   ```

   For example,

   ```
   gams mymodel lp=cbc
   ```

2. With an option command of the following form that is placed before the `solve` statement:

   ```
   Option model_type=solver_name;
   ```

   Here `option` is a keyword, `model_type` is the same model type that is used in the `solve` statement and `solver_name` is the name of one of the available solvers. For example,

   ```
   Option LP=cbc, NLP=conopt, MIP=cbc, MINLP=default;
   ```

   The `MINLP=default` switches back to the default solver for the MINLP model type.

3. Instead of providing a particular solver for a model type, the option Solver can be used to use a given solver for all model types this solver can handle.

```
Option Solver=cbc;
```

4. (Re)running `gamsinst` at any time and altering the choice of default solver as described in the installation notes.

Note

A list of all solvers and current default solvers may be generated in the `listing` file with `Option SubSystems;`.

4.30.6 Making New Solvers Available with GAMS

This short section is to encourage those of you who have a favorite solver not available through GAMS. Linking a solver program with GAMS requires some programming skills and the use of libraries provided by GAMS. There is a collection of open source solver links to GAMS at the COIN-OR project GAMSLinks. The benefits of a link with GAMS to the developer of a solver are several. They include:

- Immediate access to a wide variety of test problems.
- An easy way of making performance comparisons between solvers.
- The guarantee that a user has not somehow provided an illegal input specification.
- Elaborate documentation, particularly of input formats, is not needed.
- Access to the existing community of GAMS users, for marketing or testing.

This completes the discussion of the model and solve statements.

4.31 Conditional Expressions, Assignments and Equations

4.31.1 Introduction

This chapter deals with the way in which conditional assignments, expressions and equations are made in GAMS. The index operations already described are very powerful, but it is necessary to allow for exceptions of one sort or another. For example, heavy trucks may not be able to use a particular route because of a weak bridge, or some sectors in an economy may not produce exportable products. Exceptions such as these may easily be modeled with a logical condition combined with the dollar operator `'$'`, a very powerful feature of GAMS introduced in this chapter.

This chapter is organized as follows: We will introduce the general form of the dollar condition first and then we will focus on the various types of logical conditions. Next, we will discuss how dollar conditions are used to build conditional assignments, conditional indexed operations and conditional equations. We will conclude the chapter by showing that in certain cases conditions may be modeled using filtering sets instead of the dollar operator. Programming flow control features such as the if statement, the loop, the while statement, and the for statement are not covered in this chapter. These can be found in the chapter Programming Flow Control Features.

4.31.2 The Dollar Condition

The dollar operator is one of the most powerful features in GAMS. The general syntax for a conditional expression is:

```
term $ logical_condition
```

Here, `term` can be a number, a (indexed) symbol, and also a complex expression. The dollar operator may be read as *under the condition that* the following `logical_condition` evaluates to `TRUE` (or is unequal 0).

Consider the following simple condition, where `a` and `b` are scalars.

```
if (b > 1.5), then a = 2
```

This can be written in GAMS using the dollar operator as follows.

```
a $ (b > 1.5) = 2 ;
```

Note that the `term` is the scalar `a` and the `logical condition` is the expression ($b > 1.5$). If the condition is not satisfied, no assignment is made. To make it clear, this conditional assignment may be read as: '*given that b is greater than 1.5, a equals 2*'.

Logical conditions may take various forms, they are introduced in the next section. Conditional expressions may be used in the context of assignments, indexed operations and equations. These topics are covered in later sections of this chapter.

Note

Logical conditions used with the dollar operator cannot contain variables. However, variable attributes are allowed.

4.31.3 Logical Conditions

Logical conditions are special expressions that evaluate to a value of either `TRUE` or `FALSE`. Logical conditions may be numerical expressions and numerical relations, they may refer to set membership and they may also contain acronyms. In the following subsections this is shown in the context of simple conditional assignments with the dollar operator on the left-hand side (compare section Dollar on the Left).

In this section we use many examples to illustrate the concepts that are being introduced. In all these examples `a` and `b` are scalars, `s`, `t`, `u` and `v` are parameters, and `i` and `j` are sets.

4.31.3.1 Logical Conditions: Numerical Expressions

Numerical expressions may serve as logical conditions: a result of zero is treated as the logical value `FALSE` and a non-zero result is treated as the logical value `TRUE`. The following example illustrates this point.

```
b $ (2*a - 4) = 7;
```

Here the numerical expression $(2 * a - 4)$ is the logical condition. The numerical expression is zero if `a` equals 2, and non-zero otherwise. Hence the logical value of the expression is `FALSE` for $a = 2$ and `TRUE` for all other values of `a`. The assignment is only made if the numerical expression evaluates to `TRUE`, otherwise no assignment is made.

Attention

> Values of the extended range arithmetic such as `inf` are also allowed in logical conditions. If the result of a numerical expression used as a logical condition takes any of these values, the logical value is `TRUE`, even for e.g. `eps`, which is numerically 0.

Observe that functions are also allowed in logical conditions. If they evaluate to zero, the logical condition is `FALSE`, otherwise it is `TRUE`. Consider the following example:

```
b $ cos(a) = 7;
```

Note that the assignment is only made if the cosine of `a` does not equal zero.

4.31.3.2 Logical Conditions: Numerical Relational Operators

Numerical relational operators compare two numerical expressions and return a logical value. For completeness, all numerical relational operators are listed in Table 1.

Relation	Operator	Alternative Notation	Return Values
Strictly less than	`x < y`	`x lt y`	Returns `TRUE` if $x < y$, else returns `FALSE`.
Less than or equal to	`x <= y`	`x le y`	Returns `TRUE` if $x \leq y$, else returns `FALSE`.
Equal to	`x = y`	`x eq y`	Returns `TRUE` if $x = y$, else returns `FALSE`.
Not equal to	`x <> y`	`x ne y`	Returns `TRUE` if $x \neq y$, else returns `FALSE`.
Greater than or equal to	`x >= y`	`x ge y`	Returns `TRUE` if $x \geq y$, else returns `FALSE`.
Strictly greater than	`x > y`	`x gt y`	Returns `TRUE` if $x > y$, else returns `FALSE`.

Table 1: Numerical Relational Operators

Consider the following examples.

```
b $ (a < 0) = 10;
```

```
b $ (sqr(a) > a) = 12;
a $ ( sum(i, s(i)) > 0 ) = 7;
t(i) $ (a <> 0) = t(i) + 1;
```

In the first line the logical condition is the relational expression ($a < 0$). The assignment is only made if this expression is TRUE, that is, if the scalar a is negative. The logical condition in the second line is a bit more complex. It evaluates to FALSE if $0 \leq a \leq 1$. For all other values of a, it evaluates to TRUE. So the assignment is made for all values of a, except for those values of a that are in the closed interval $[0, 1]$. Note that if $a = -3$, then the logical condition in the first line will be TRUE, so b will become 10. In addition, the logical condition in the second line will be TRUE, so b will change to 12. The logical condition in the third line evaluates to TRUE if the sum of all values of the parameter s is strictly positive. Then a is assigned the value of 7. The assignment in the last line depends on whether a is non-zero. If a is zero no assignment is made, otherwise all entries of the parameter t are updated.

Note that acronyms may also be used with relational operators to build logical conditions. However, only the equality operator = and inequality operator <> are allowed in the context of acronyms, since they have no numerical values and the other operators would be meaningless. For an example, see section Acronym Usage.

4.31.3.3 Logical Conditions: Logical Operators

GAMS offers standard logical operators that may combine two or more logical conditions to build complex logical expressions. For example, if several expressions are required to be TRUE simultaneously, they may be connected with the operator and. The logical operators available in GAMS are listed in Table 2 and Table 3. Another way to construct complex logical conditions is by nesting them. For details, see subsection Nested Dollar Conditions below.

Operation	Operator	Alternative Notation	Description
Negation	not x		The logical condition x has to be FALSE, in order for the expression to be TRUE.
Logical conjunction	x and y		Two logical conditions are TRUE simultaneously.
Logical disjunction	x or y		At least one of two logical conditions applies.
Exclusive disjunction	x xor y		Exactly one of two logical conditions applies.
Logical implication	x imp y	x -> y	If the logical condition x is TRUE but at the same time the logical condition y is FALSE, then the whole expression is FALSE, in all other cases the expression evaluates to TRUE.
Logical equivalence	x eqv y	x <=> y	Both logical conditions are either TRUE simultaneously or FALSE simultaneously for the whole expression to be TRUE.

Table 2: Logical Operators

The logical values of these operators are summarized in the following truth table.

x	y	not x	x and y	x or y	x xor y	x imp y	x eqv y
FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	TRUE	TRUE

x	y	not x	x and y	x or y	x xor y	x imp y	x eqv y
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
TRUE	TRUE	FALSE	TRUE	TRUE	FALSE	TRUE	TRUE

Table 3: Truth Table of Logical Operators

The following somewhat artificial examples serve as illustration.

```
Set       i     / i1*i5 /;
Parameter s(i)  / i1 3, i2 5, i4 8 /
          t(i)  / i1*i4 13 /
          u(i)  / i2 1 /
          v(i)  / i1 7, i3 2 /;

u(i) $ (not s(i))               = v(i);
u(i) $ (s(i) and u(i) and t(i)) = s(i);
u(i) $ (s(i) or v(i) or t(i))   = 4;
```

Note that there are three conditional assignments for the parameter `u`. In the first assignment the logical condition is `(not s(i))`. This condition holds for all entries of `s` that are not specified and therefore zero by default: `s('i3')` and `s('i5')`. Hence `u('i3')` and `u('i5')` are assigned the values of `v('i3')` and `v('i5')` respectively. The value of `v('i3')` is 2 and the value of `v('i5')` is zero by default. After the first assignment we have `u('i2')=1` and `u('i3')=2`, all other values of `u` are zero. Note that the logical condition failed for `u('i2')` and therefore its value remained unchanged. The logical condition in the second assignment is `TRUE` for those labels of the set `i` that have non-zero entries in the parameters `s`, `u` and `t` simultaneously. This condition holds only for `i2`. Therefore `u('i2')=s('i2')=5` and all other values of `u` remain unchanged, resulting in non-zero values only for `u('i2')` and `u('i3')`. The logical condition in the last assignment evaluates to `TRUE` for all labels of the set `i` that have at least one non-zero entry in the parameters `s`, `v` and `t`. This holds for all labels except for `i5`. Therefore `u('i5')` stays zero and all other values of `u` are changed to 4. These examples demonstrate the power of the dollar operator combined with logical operators. Even more complex logical conditions are possible; see subsection Mixed Logical Conditions below for details.

4.31.3.4 Logical Conditions: Set Membership and Set Functions

Apart from numerical and relational expressions, set membership and functions referencing set elements may be used as a logical condition. Consider the following example as illustration for set membership as logical condition.

```
Set        i     / i1*i5 /
           j(i)  / i1*i3 / ;
Parameter  s(i)  / i1 3, i2 5, i3 11, i4 8, i5 1 /
           t(i);
t(i) $ j(i) = s(i) + 3;
```

Note that the set `j` is a subset of the set `i` and that the parameter `t` is declared but not defined. The conditional expression `t(i)$j(i)` in the last line restricts the assignment to the members of the subset `j` since only they satisfy the condition `j(i)`. The values for `t('i4')` and `t('i5')` remain unchanged. In this case this means that they are zero (by default). Note that there is an alternative formulation for this type of conditional assignment; for details see subsection Filtering Sets in Assignments below.

Note

Only the membership of subsets and dynamic sets may be used as logical conditions.

The use of set membership as a logical condition is an extremely powerful feature of GAMS, see section Conditional Equations below for more examples.

Logical conditions may contain predefined symbols or operators that return particular values depending on the position of elements in sets, the size of sets or the comparison of set elements to each other or text strings:

Predefined Symbol / Operator	Description and Comments
`sameAs(element1,element2)` or `sameAs("text",element2)` or `sameAs(element1,"text")`	Predefined symbol that returns `TRUE` if `element1` is identical to `element2` or `element1` is identical to `text`, and `FALSE` otherwise.
`diag(element1,element2)` or `diag("text",element2)` or `diag(element1,"text")`	Like `sameAs`, but returns numerical value 1 for `TRUE` and 0 for `FALSE`.
`card(set_name)`	Returns the number of elements in a set. Note that this can also be called with a quoted text and returns the number of characters of that text in that case. For more details, see section The Card Operator.
`ord(set_name)`	Returns the relative position of an element in a set. Note that `ord` may be used *only* with one-dimensional, static, ordered sets. For more details, see section The Ord Operator.

Table 4: Predefined Symbols / Operators Referencing Sets

In the following example we have two sets of cities and we want to know how many of them feature in both sets.

```
Set     i     / Beijing, Calcutta, Mumbai, Sydney, Johannesburg, Cairo /;
Set     j     / Rome, Paris, Boston, Cairo, Munich, Calcutta, Barcelona /;
Scalar  b;
b = sum((i,j)$sameAs(i,j),1);
```

Note that in the assignment statement we sum over both sets and we use the predefined symbol `sameAs` to restrict the domain of the indexed operation to those label combinations `(i,j)` where `sameAs` evaluates to `TRUE`. Thus only identical elements are counted.

Note that in this example the predefined symbol `diag` could have been used with the same result.

The operators `ord` and `card` are frequently used to single out the first or last element of an ordered set. For example, we may want to fix a variable for the first and last elements of a set:

```
x.fx(i) $ (ord(i) = 1)       = 3;
x.fx(i) $ (ord(i) = card(i)) = 7;
```

In the first assignment the variable `x` is fixed for the first element of the set `i` and in the second assignment `x` is fixed for the final element of `i`.

Note

As an alternative to the formulation above, one could also use the set attributes `first` and `last` to get the same result:

```
x.fx(i) $ (i.first) = 3;
x.fx(i) $ (i.last)  = 7;
```

4.31.3.5 Numerical Values of Logical Conditions

We have seen that logical conditions may take the form of numerical expressions, expressions with relational operators, complex expressions using logical operators, set membership and set functions. However, GAMS does not have a Boolean data type.

Attention

GAMS follows the convention that the result of a relational operation is zero if the assertion is `FALSE`, and 1 if it is `TRUE`.

Consider the following example as illustration.

```
x = (1 < 2) + (2 < 3);
```

The expression to the right of the assignment evaluates to 2 since both logical conditions within parentheses are `TRUE` and therefore assume a value of 1. Note that this is different from the assignment below:

```
x = (1 < 2) or (2 < 3)
```

This assignment evaluates to 1, since both statements to the left and right of `or` are `TRUE` and therefore the whole expression is `TRUE`.

4.31.3.6 Mixed Logical Conditions

The building blocks introduced in the subsections above may be combined to generate more complex logical conditions. These may contain standard arithmetic operations, numerical relational operations and logical operations. All operations, their symbols and their order of precedence are given in Table 5. Note that 1 denotes the highest order of precedence and 7 denotes the lowest order of precedence. As usual, the default order of precedence holds only in the absence of parentheses and operators (symbols) on the same level are evaluated from left to right.

Type of Operation	Operation	Operator	Order of precedence
Standard arithmetic operation	Exponentiation	**	1
Standard arithmetic operation	Multiplication, Division	*, /	2
Standard arithmetic operation	Unary operators: Plus, Minus	+, -	3
Standard arithmetic operation	Binary operators: Addition, Subtraction	+, -	3
Numerical Relational operation	All	<, <=, =, <>, >=, >	4
Logical operation	Negation	not	5
Logical operation	Logical Conjunction	and	6
Logical operation	All other logical opera-	or, xor, imp, eqv	7

Table 5: Complete Hierarchy for Operator Precedence in GAMS

Note

We recommend to use parentheses rather than relying on the order of precedence of operators. Parentheses prevent errors and make the intention clear.

Consider the following example:

```
x - 5*y and z - 5
(x - (5*y)) and (z-5)
```

These two complex logical conditions are equivalent. However, the parentheses make the second expression easier to understand.

Some simple examples of complex logical conditions, their numerical values and their logical values are given in Table 6.

Logical Condition	Numerical Value	Logical Value
(1 < 2) + (3 < 4)	2	TRUE
(2 < 1) and (3 < 4)	0	FALSE
(4*5 - 3) + (10/8)	18.25	TRUE
(4*5 - 3) or (10 - 8)	1	TRUE
(4 and 5) + (2*3 <= 6)	2	TRUE
(4 and 0) + (2*3 < 6)	0	FALSE

Table 6: Examples of Complex Logical Conditions

4.31.3.7 Nested Dollar Conditions

An alternative way to model complex logical conditions is by nesting them. The syntax is:

```
term $ (logical_condition1$(logical_condition2$(...)))
```

Note that in nested dollar conditions all succeeding expressions after the dollar operator must be enclosed in parentheses. The nested expression is equivalent to the following conditional expression that uses the logical operator `and` instead of the nesting:

```
term $ (logical_condition1 and logical_condition2 and ...)
```

Consider the following example. Note that `i`, `j(i)` and `k(i)` are sets, and `u(i)` and `v(i)` are parameters.

```
u(i) $ (j(i)$k(i)) = v(i) ;
```

The assignment will be made only for those members of the set `i` that are members of both sets `j` and `k`. Note the position of the parentheses in the dollar condition. The assignment statement above can be rewritten in the following way.

```
u(i) $ (j(i) and k(i)) = v(i) ;
```

Note

We recommend to use the logical `and` operator instead of nesting dollar operators, because this formulation is easier to read.

4.31.4 Conditional Assignments

A conditional assignment is an assignment statement with a dollar condition on the left-hand side or on the right-hand side. Most examples until now were conditional assignments with the dollar operator on the left.

Attention

> The effect of the dollar condition is significantly different depending on which side of the assignment it is located.

The next two subsections describe the use of the dollar condition on each side of the assignment. Note that in many cases it may be possible to use either of the two forms of the dollar condition to describe an assignment. We recommend to choose the clearer formulation.

Note that if the logical condition in an assignment statement refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section Filtering Sets in Assignments below.

4.31.4.1 Dollar on the Left

If the dollar condition is on the left-hand side of an assignment, an assignment is made only in case the logical condition is satisfied. If the logical condition is not satisfied then no assignment is made and the previous content of the parameter on the left will remain unchanged. In case the parameter on the left-hand side of the assignment has not previously been initialized or assigned any values, zeros will be used for any label for which the assignment was suppressed.

Consider the following example adapted from [**CHENERY**]. Note that the parameter `sig` has been previously defined in the model.

```
rho(i) $ (sig(i) <> 0) = (1./sig(i)) - 1. ;
```

In this assignment `rho(i)` is calculated and the dollar condition on the left protects against dividing by zero. If any of the values associated with the parameter `sig` turns out to be zero, no assignment is made and the previous values of `rho(i)` remain. As it happens, `rho(i)` was not previously initialized, and therefore all the labels for which `sig(i)` is zero will result in a value of zero.

Now recall the convention that non-zero implies `TRUE` and zero implies `FALSE`. The assignment above could therefore be written as:

```
rho(i) $ sig(i)  =  (1./sig(i)) - 1. ;
```

In the following examples `i` is a set and `s` and `t` are parameters.

```
s(i) $ t(i) = t(i);
s(i) $ ((t(i)-1) > 0) = t(i)**0.5;
```

Note that the first assignment is suppressed if the value of the parameter `t` equals zero. The second assignment is suppressed for values of the parameter `t` that are smaller or equal to 1.

Note

> The first of the last two examples is special in the way that the logical condition used with the dollar on the left and the expression on the right hand side of the assignment are actually the same. This allows to use a shorter notation using the combined `$=` assignment. That assignment is equivalent to the following one (which called sparse assignment):
>
> ```
> s(i) $= t(i);
> ```

4.31.4.2 Sparse Assignments

Sparse assignments will assign a value to the left-hand side of the = sign only if the right-hand side is nonzero. This behavior is triggered using the $= notation as in the following example.

```
Set       i                                          /a,b,c/
Parameter d1(i)  "Initial data"                      /a 1, b  1, c 1/
          d2(i)  "Data to be used to overwrite d1"   /a 0, b -2 /
          d3(i)  "Empty data parameter";

* Initialize d3 to d1
d3(i) = d1(i);
* Use dollar on the left to overwrite d3 with non-zero values from d2
d3(i)$d2(i) = d2(i);
* Result: d3('a')=1; d3('b')=-2; d3('c')=1;

* Re-Initialize d3 to d1
d3(i) = d1(i);
* Use sparse assignment to overwrite d3 with non-zero values from d2
d3(i) $= d2(i);
* Result: d3('a')=1; d3('b')=-2; d3('c')=1;
```

Note that in the final assignments above, the values of the parameter d3(i) are replaced with values from parameter d2(i) only if the entry in d2(i) is non-zero, the other values are left untouched. Both assignments are equivalent.

4.31.4.3 Dollar on the Right

If the dollar condition is on the right-hand side of an assignment statement, an assignment will *always* be made. In case the logical condition is not satisfied the value of zero is assigned.

Above we had the following simple conditional assignment:

```
a $ (b > 1.5) = 2;
```

Now we move the dollar condition to the right-hand side:

```
a = 2 $ (b > 1.5)  ;
```

This is equivalent to:

```
if (b > 1.5)   then (a = 2),    else (a = 0)
```

Note that an if-then-else type of construct is implied, but the else operation is predefined and never made explicit. The else could be made explicit with the following formulation:

```
a = 2 $ (b > 1.5) + 0 $ (b <= 1.5) ;
```

The use of this feature is more apparent in instances when an `else` condition needs to be made explicit. Consider the next example adapted from the fertilizer model [**FERTD**] . The set `i` is the set of plants, and we are calculating `mur(i)`, the cost of transporting imported raw materials. In some cases a barge trip must be followed by a road trip because the plant is not alongside the river and we must combine the separate costs. The assignment is:

```
mur(i) = (1.0 +. 0030*ied(i,'barge')) $ ied(i,'barge')
       + (0.5 + .0144*ied(i,'road' )) $ ied(i,'road' );
```

This means that if the entry in the distance table `ied` is not zero, then the cost of shipping using that link is added to the total cost. If there is no distance entry, there is no contribution to the cost, presumably because that mode is not used.

Consider another example for a conditional assignment with the dollar operator on the right:

```
b = sum(i,t(i)) $ (a > 0) + 4;
```

Here `a` and `b` are scalars, `i` is a set and `t` is a parameter. If the scalar `a` is positive, the scalar `b` is assigned the sum of all values of the parameter `t` plus 4. If `a` is zero or negative, `b` becomes just 4. Note that the sum is only computed if the condition holds, this potentially makes the program faster.

4.31.5 Conditional Indexed Operations

We have seen how exceptions in assignments are modeled with dollar conditions. Dollar conditions are also used in indexed operations, where they control the domain of operation. This is conceptually similar to the conditional assignment with the dollar on the left.

Consider the following example adapted from [**GTM**], a gas trade model for interrelated gas markets. Here the set `i` contains supply regions and the parameter `supc` models supply capacities. The scalar `tsupc` is computed with the following statement:

```
tsupc  =  sum(i $ (supc(i) <> inf), supc(i))  ;
```

This assignment restricts the sum to the *finite* values of the parameter `supc`.

In indexed operations the logical condition is often a set. This set is called the *conditional set* and assignments are made only for labels that are elements of the conditional set. This concept plays an important role in dynamic sets.

Multi-dimensional sets are introduced in section Multi-Dimensional Sets. In the example used there a two-dimensional set is used to define the mapping between countries and ports. Another typical example for a multi-dimensional set is a set-to-set mapping that defines the relationship between states and regions. This is useful for aggregating data from the state to the regional level. Consider the following example:

```
Sets r          "regions" / north, south /
     s          "states"  / florida, texas, vermont, maine /
     corr(r,s)            / north.(vermont,maine)
                            south.(florida,texas) /;

Parameter y(r)      "income for each region"
          income(s) "income of each state"
                          / florida   4.5, vermont   4.2
                            texas     6.4, maine     4.1 / ;
```

The set corr links the states to their respective regions, the parameter income is the income of each state. The parameter y is computed with the following assignment statement:

```
y(r) = sum(s $ corr(r,s), income(s)) ;
```

The conditional set corr(r,s) restricts the domain of the summation: for each region r the summation over the set s is restricted to the label combinations (r,s) that are elements of the set corr(r,s). Conceptually, this is analogous to the Boolean value TRUE or the arithmetic value non-zero. The effect is that only the contributions of vermont and maine are included in the total for north, and south is the sum of the incomes from only texas and florida.

Note that the summation above can also be written as:

```
y(r) = sum(s,income(s) $ corr(r,s));
```

In this formulation the parameter income is controlled by the conditional set corr instead of the index s. Note that both formulations yield the same result, but the second alternative is more difficult to read.

In the next example the logical condition on the right-hand side appears in the context of an indexed operation, but it does not refer to the index. Thus it is treated similarly to a dollar on the right logical condition. Note that a and b are scalars, i is a set and s is a parameter.

```
b = sum(i $ a, s(i));
```

Here the scalar b is assigned the sum of the values of the parameter s only if a does not equal zero. Otherwise b will be assigned the value zero. Observe that the following alternative formulation will generally be faster than the assignment above:

```
b = sum(i, s(i)) $ a;
```

Note that if the logical condition in the context of indexed operations refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section Filtering Controlling Indices in Indexed Operations below.

4.31.6 Conditional Equations

The dollar operator is also used for exception handling in equations. The next two subsections discuss the two main uses of dollar operators in the context of equations: in the body of an equation and over the domain of definition.

4.31.6.1 Dollar Operators within the Algebra of Equations

A dollar operator in the algebraic formulation of an equation is analogous to the dollar control on the right of assignments, as presented in section Dollar on the Right. Assuming that "the right" means the right of the `'..'` then the analogy is even closer. As in the context of assignments, an `if-else` operation is implied. It is used to exclude parts of the definition from some of the generated constraints.

Consider the following example adapted from the production model [**CHENERY**].

```
Set         i       "sectors"     / light-ind, food+agr, heavy-ind, services /
            t(i)    "tradables"   / light-ind, food+agr, heavy-ind /;

Variables   x(i)    "quantity of output"
            y(i)    "final consumption"
            e(i)    "quantity of exports"
            m(i)    "quantity of imports";

Equations   mb(i)   "material balance";

mb(i)..     x(i) =g= y(i) + (e(i) - m(i)) $ t(i) ;
```

Note that in the equation definition in the last line, the term `(e(i) - m(i))` on the right-hand side of the equation is added only for those elements of the set `i` that also belong to the subset `t(i)`, so that the element `services` is excluded.

Further, conditional indexed operations may also feature in expressions in equation definitions. The supply balance equation from the gas trade model [**GTM**] is an example. Note that the set `i` contains the supply regions, the set `j` contains the demand regions, and the two-dimensional set `ij` is the set of feasible links; the variable `x` denotes the shipment of natural gas and the variable `s` denotes the regional supply.

```
sb(i)..  sum(j $ ij(i,j), x(i,j))  =l=  s(i)  ;
```

Similar to the assignment example seen before, the conditional set `ij(i,j)` restricts the domain of the summation: for each supply region `i` the summation over the demand regions `j` is restricted to the label combinations `(i,j)` that are elements of the set of feasible links `ij(i,j)`.

4.31.6.2 Dollar Control over the Domain of Definition

In case constraints should only be included in the model if particular conditions are met, a dollar condition in the domain of definition of an equation may be used to model this restriction. Such a dollar condition is analogous to the dollar control on the left of assignments. Assuming that "the left" means the left of the symbols `'..'` then the analogy is even closer.

Note

> The dollar control over the domain of definition of equations restricts the number of constraints generated to less than the number implied by the domain of the defining sets.

Consider the following example adapted from the Andean fertilizer model [**ANDEAN**] :

```
gple(w,wp,te) $ ple(w,wp)..  yw(w,te) - yw(wp,te) =l= dpack;
```

Here `w`, `wp` and `te` are sets, `ple` is a two-dimensional parameter, `yw` is a variable and `dpack` is a scalar. Note that the dollar condition restricts the first two indices of the domain of the equation to those label combinations that have non-zero entries in the two-dimensional parameter `ple`.

Sometimes the desired restriction of an equation may be achieved either way: through a condition in the algebra or a condition in the domain of definition. Compare the following two lines, where `eq1` and `eq2` are equations, `i` and `j` are sets, `b` is a scalar, `s` is a parameter and `x` is a two-dimensional variable.

```
eq1(i) $ b..  sum(j, x(i,j))     =g=  -s(i);
eq2(i)..      sum(j, x(i,j)) $ b =g=  -s(i) $ b;
```

In the first line the dollar condition is in the domain of definition, in the second line the dollar conditions are in the algebraic formulation of the equation. If `b` is non-zero, the generated equations `eq1` and `eq2` will be identical. However, if `b` is 0, no equation `eq1` will be generated, but for each `i` we will see a trivial equation `eq2` of the form `0 =g= 0;`.

Note that if the logical condition in the domain of definition of an equation refers to set membership, then under certain conditions the restriction may be expressed without the use of the dollar operator. For details, see section Filtering the Domain of Definition below.

4.31.7 Filtering Sets

If the logical condition refers to set membership, the restriction modeled with a dollar condition may sometimes be achieved *without* the dollar operator. Consider the following statement, where `i` and `j(i)` are sets, and `u` and `s` are parameters:

```
u(i) $ j(i) = s(i) ;
```

Note that the assignment is made only for those elements of the set `i` that are also elements of the subset `j`. This conditional assignment may be rewritten in a shorter way:

```
u(j) = s(j) ;
```

In this statement the assignment has been filtered through the condition without the dollar operator by using the subset `j` as the domain for the parameters `u` and `s`. This formulation is cleaner and easier to understand. It is particularly useful in the context of multi-dimensional sets (tuples), and it may be used in assignments, indexed operations and the domain of definition of equations.

4.31.7.1 Filtering Sets in Assignments

Suppose we want to compute the transportation cost between local collection sites and regional transportation hubs for a fictional parcel delivery service. We define sets for the collection sites and transportation hubs and a two-dimensional set where the collection sites are matched with their respective hubs:

```
Set i       "local collection sites"       / miami, boston, chicago, houston, phoenix /
    j       "regional transportation hubs" / newyork, detroit, losangeles, atlanta    / ;

Set r(i,j)  "regional transportation hub for each local collection site"  /
    boston   .newyork
    miami    .atlanta
    houston  .atlanta
    chicago  .detroit
    phoenix  .losangeles  / ;

Table distance(i,j)     "distance in miles"
           newyork     detroit  losangeles     atlanta
miami         1327        1387        2737         665
boston         216         699        3052        1068
chicago        843         275        2095         695
houston       1636        1337        1553         814
phoenix       2459        1977         398        1810;

Parameter factor        "cost estimate per unit mile"
          shipcost(i,j) "cost of transporting parcels from a local collection site to a regional hub

factor = 0.009 ;
shipcost(i,j) $ r(i,j) = factor*distance(i,j) ;
```

The distance between collection sites and transportation hubs is given in the table. The last line is a conditional assignment for the parameter `shipcost`. This assignment is only made if the label combination `(i,j)` is an element of the set `r`. Note that in each instance the indices `i` and `j` appear together. Thus the assignment may be simply written as:

```
shipcost(r) = factor*distance(r) ;
```

Note that the assignment is explicitly restricted to the members of the set `r`; the dollar operator is not necessary. Observe that if the indices `i` or `j` appear separately in any assignment, the above simplification cannot be made. For example, consider the case where the shipping cost depends not only on the parameter `factor` and the distance between collection sites and regional hubs, but also on the congestion at the regional hub. We introduce a new parameter `congestfac` that models the congestion at each regional hub and is indexed only over the set `j`:

```
Parameter congestfac(j)   "congestion factor"  /
   newyork      1.5
   detroit      0.7
   losangeles   1.2
   atlanta      0.9/ ;
```

The new cost of shipment is computed as follows:

```
shipcost(i,j) $ r(i,j) = factor*congestfac(j)*distance(i,j) ;
```

Note that this conditional assignment *cannot* be reformulated as:

```
shipcost(r) = factor*congestfac(j)*distance(r) ;
```

In the representation above the index `j` appears on the right-hand side, but not on the left-hand side. GAMS will flag this assignment as an error. However, the following representation will work:

```
shipcost(r(i,j)) = factor*congestfac(j)*distance(r) ;
```

In this formulation the set `r` is explicitly denoted as a tuple of the sets `i` and `j`. The set `j` may then appear on the right-hand side.

4.31.7.2 Filtering Controlling Indices in Indexed Operations

Similarly, the controlling indices in indexed operations may be filtered through the conditional set without the use of the dollar operator. We continue with the shipping cost example from the last subsection. The total cost of shipment is obtained through the equation that follows. We also include the variable definitions for clarity.

```
Variable shipped(i,j), totcost ;
Equation costequ ;

costequ.. totcost =e= sum((i,j) $ r(i,j), shipcost(i,j)*shipped(i,j));
```

Here the variable `shipped` is the number of parcels shipped from the local collection site `i` to the regional transportation hub `j`, and the variable `totcost` is the total cost of all shipments. Note that she summation in the equation is restricted to the label combinations that are elements of the set `r`. Alternatively, the equation above may be written as:

```
costequ.. totcost =e= sum(r, shipcost(r)*shipped(r));
```

In this formulation the summation is performed explicitly only over the elements of the set `r`, no dollar condition is necessary. However, if the expression in the equation included a term dependent only on index `j`, then we would have to reformulate differently. Suppose the equation included also the congestion factor `congestfac` that is indexed only over `j`:

```
cost.. totcost =e= sum((i,j) $ r(i,j), factor*congestfac(j)*distance(i,j)*shipped(i,j));
```

In this case the equation needs to be simplified in the following way:

```
cost.. totcost =e= sum(r(i,j), factor*congestfac(j)*distance(r)*shipped(r));
```

Like before, the domain of the indexed operation `sum` is the set `r`. But this time the domain of `r` has to be named explicitly, so that the parameter `congestfac` which is indexed only over the set `j` is permitted in the scope of the indexed operation. Note that this reasoning is analogous to the reasoning for filtering sets in assignments in the subsection above.

4.31.7.3 Filtering the Domain of Definition

The rules for filtering sets that we have introduced in subsections Filtering Sets in Assignments and Filtering Controlling Indices in Indexed Operations also apply in the context of equation domains. We continue with the parcel transport example introduced above and add a binary variable `bin`, the parameter `bigM` and the equation `connect` to the model. Recall that `shipped(i,j)` is a variable and `r(i,j)` is a set.

```
Parameter        bigM(i,j) ;
Binary variable  bin(i,j) ;

Equation         connect(i,j) ;

connect(i,j) $ r(i,j).. shipped(i,j) =l= bigM(i,j)*bin(i,j) ;
```

The dollar condition restricts the domain of definition of the equation `connect` to those label combinations of the sets `i` and `j` that are elements of the set `r`. The equation relates the continuous variable `shipped(i,j)` to the binary variable `bin(i,j)`. Note that each domain in the equation is the index pair `(i,j)`. So the equation may be simplified as follows:

```
connect(r).. shipped(r) =l= bigM(r)*bin(r) ;
```

In this formulation the domain of the equation is explicitly restricted to the members of the set `r`, without the use of a dollar condition. Note that if the right-hand side of the equation contained any term that was indexed over `i` or `j` separately, then the domain of definition of the equation would have to be simplified as:

```
connect(r(i,j))
```

The reasoning is the same as in the case of assignments and indexed operations.

4.32 The Display Statement

4.32.1 Introduction

The `display` statement in GAMS is a quick way to write data into the listing file. It does not provide a publication quality reporting function, but is instead aimed at functionality that is easy to use and provides graceful defaults. These defaults may be modified by the user to control the layout and appearance of the output. Both defaults and customization options are presented in this chapter. For information on the listing file, see chapter GAMS Output. In addition to the `display` statement, GAMS offers the put writing facility, a more sophisticated way to generate reports, where the user has much more flexibility and control over the output of individual items.

4.32.2 The Syntax

In general, the syntax for the `display` statement in GAMS is as follows:

```
display ident | quoted text {, ident | quoted text};
```

The keyword `display` indicates that this is a `display` statement. It is followed by an identifier. If the identifier is a set or a parameter, only the name of the set or parameter itself is specified. If the identifier is a variable, an equation or a model, it must be followed by a suffix, i.e. a variable attribute, equation attribute or model attribute respectively. As usual, `quoted text` must be surrounded by single or double quotes. The identifiers and the text may be mixed and matched in any order, and the whole statement may be continued over several lines. For conditional displays a dollar condition may follow the keyword `display`.

The output produced by a display statement consists of labels and data. For multi-dimensional sets the keyword `yes` (indicating membership) is used instead of values.

Note

Only the non-default values are displayed for all data types.

The default value is generally zero. Exceptions are some upper and lower bounds of variables and equations (the attributes `.lo` and `.up`). Their default values are listed in sections Variable Attributes and Equation Attributes respectively.

The syntax of the display statement is illustrated by the following example:

```
Sets       s       / s1*s4 /
           t       / t5*t7 /
           st(s,t) / s1.t5, s1.t7, s2.t6, s3.t7, s4.t5 / ;
Parameters p(s)    / s1  0.33, s3  0.67 /
           q(t)    / t5  0.33, t7  0.67 / ;
Variable   v(s,t) ;
v.l(s,t) = p(s)*q(t);
display 'first a simple set', s, 'then a two-dimensional set', st, 'then a parameter', p,
        'then the activity level of a variable', v.l;
```

The display statement will generate the following lines in the listing file:

```
----      8 first a simple set

----      8 SET s

s1,    s2,    s3,    s4


----      8 then a two-dimensional set

----      8 SET st

            t5          t6          t7

s1         YES                     YES
s2                     YES
s3                                 YES
s4         YES


----      8 then a parameter

----      8 PARAMETER p

s1 0.330,    s3 0.670


----      8 then the activity level of a variable

----      8 VARIABLE v.L

            t5          t7

s1       0.109       0.221
s3       0.221       0.449
```

In the case of multi-dimensional identifiers like the set st and the level values of the variable v, the data is by default reported in a tabular form that is easy to read. Note that only the non-default values are displayed. To display zero values, the special value EPS can be used. In the example above, all values of the parameter p can be displayed by adding the following lines.

```
display 'display all values';
p(s)$(not p(s)) = EPS;
display p;
```

The following lines will be displayed in the listing file.

```
----     11 display all values

----     13 PARAMETER p

s1 0.330,    s2   EPS,    s3 0.670,    s4   EPS
```

An alternative to displaying EPS is to use acronym but computations cannot be done with the resulting parameters.

4.32.3 Displaying Multi-Dimensional Identifiers: Label Order

We have seen that two-dimensional identifiers are displayed in table format. The default layout for displaying multi-dimensional identifiers is summarized in Table 1. Note that the default format may be modified with the local display control. The figures in the table refer to the index position in the domain list of the identifier. For example, if we display x, where x has been declared as x(i,j,k,l), then the i labels (the first index) will by default be associated with the individual subtables, the j and k with the row labels and the l (the fourth and last index) with the column headings.

Number of Indices	Subtable	Index Position(s) on the Row	Column
1		list format	
2	-	1	2
3	-	1,2	3
4	1	2,3	4
5	1,2	3,4	5
6	1,2,3	4,5	6

Table 1: Default Layout for Displaying Multi-Dimensional Identifiers

For 7 to the maximum number of indices the natural progression is followed. For the current maximum number of indices, see Dimensions. The labels vary slowest for the first index position and quickest for the highest. Within each index position the order of the label is the GAMS entry order of the labels. Some users may need to manipulate the order of the labels.

Note

The only way to change the order in which the labels for each index position appear in the display output is to change the order of appearance of the labels in the GAMS program.

This is most easily done by declaring a set whose only purpose is to list all the labels in the order that is needed. Make this set the very first declaration in the GAMS program.

Note that the order of the indices in an identifier is always as the order in the declaration statement. One can declare them in the order that is found appealing, or make an assignment to a new identifier with a different order, as illustrated in the example below.

4.32.3.1 Example for Multi-Dimensional Displays

Consider the following example. The parameter x has four dimensions or index positions. It is initialized using parameter format and then displayed as shown below.

```
Set i   first index      / first, second /
    j   second index     / one, two, three /
    k   third index      / a, b /
    l   fourth index     / i, ii /  ;

Parameter
    x(i,j,k,l) a four dimensional structure /
               second.one.a.i      0.00013,   first .three.b.i       -6.3161
               first .one.b.i      5.63559,   second.two  .b.i    20000.00
               second.one.b.ii   -17.29948,   first .two  .b.ii      10.3457
               first .two.a.ii     0.02873,   second.one  .a.ii       1.0037
               second.two.a.ii        +inf,   first .two  .a.i       -2.9393
               first .one.a.ii     0.00000  /
    y(k,l,i,j) a four dimensional structure with changed index order;
y(k,l,i,j) = x(i,j,k,l);
display x, y;
```

For variable x the code fragment produces the following output:

```
----     16 PARAMETER x  a four dimensional structure

INDEX 1 = first

                  i          ii

one  .b       5.636
two  .a      -2.939       0.029
two  .b                   10.346
three.b      -6.316

INDEX 1 = second

                i          ii

one.a 1.300000E-4       1.004
one.b                 -17.299
two.a                    +INF
two.b   20000.000
```

Notice that there are two subtables, one for each label in the first index position. Note further that the order of the labels is not the same as in the input data list of the parameter `x`, but is according to the label order in the driving sets. Observe that the numbers closer in magnitude to one are displayed in fixed format, rounded to three digits (the default). This default may be modified with display controls, the topic of the next section. In contrast, the especially small value appears in E-format: it would display as all zeros in fixed format. Note also that the zero in the list for `x('first','one','a','ii')` has vanished, since default values are not shown. Similarly, rows and columns containing only zero values are suppressed in each subtable separately.

Notice furthermore how parameter y can be used to display the same data with a different index order.

4.32.4 Display Controls

GAMS allows the user to modify the number of row and column labels in the display listing as well as the accuracy of the data being displayed. There are global and local display controls. Global display controls allow the user to affect more than one display statement. The local display control may be used to override the global controls if specific data need to be listed in a particular format. Both controls are expressed with option statements.

4.32.4.1 Global Display Controls

The first option for global display controls specifies the number of digits shown after the decimal point. It affects the numbers appearing in all display output following the option statement, unless it is changed for a specific identifier as shown below. The general form of this option statement is:

```
option decimals = value;
```

The keyword `option` indicates that this is an option statement, `decimals` is the specific option and `value` is an integer between 0 and 8. The value 0 supresses decimals as well as the decimal point. Note that the default is `decimals=3`, and that setting this option does not change the field width, just the number of decimals. Increasing `decimals` results in more digits being displayed; this may in turn cause numbers to appear in E-format if the longer fixed-format result would exceed the field width.

A word about rounding for display purposes is in order. In the event of ties, the display routines use the **round-to-even** tie-breaking rule. So for example, with `decimals=0`, both 1.5 and 2.5 will display as **2**. Similarly, with `decimals=1`, 2.25 displays as **2.2** while 2.75 displays as **2.8**.

Consider the following extension to the example discussed in the previous section.

```
option decimals = 1;  display x;
```

GAMS will display numbers using E-format where necessary to avoid exceeding the display width or showing all zeros. The output is as follows:

```
----     14 PARAMETER x  a four dimensional structure

INDEX 1 = first

                  i          ii

one  .b         5.6
```

```
two  .a        -2.9 2.873000E-2
two  .b                    10.3
three.b        -6.3

INDEX 1 = second

                i          ii

one.a 1.300000E-4         1.0
one.b                    -17.3
two.a                     +INF
two.b     20000.0
```

Note that GAMS overrode the decimal choice for `x('first','two','a','ii')` to ensure that small numbers are not displayed as zeroes.

Another option for global display control modifies the width of the display. This is particularly important for longer label names. Label names that are headers of columns are cut off after 9 characters when data is displayed in table format. The following option may be used to display longer names in full:

```
option dispwidth = value;
```

The name of the option is `dispwidth` and `value` is a positive integer smaller or equal to 31. An illustrative example follows.

```
Set i      'South African cities'       / Johannesburg, Cape_Town /
    j      'other African cities'       / Maputo, Dar_es_Salaam  /
    f(i,j) 'direct flight connections'  / Johannesburg.Maputo, Johannesburg.Dar_es_Salaam, Cape_Town
display f ;
```

The display statement generates the following output:

```
----      4 SET f  direct flight connections

                  Maputo  Dar_es_Sa~

Johannesburg         YES         YES
Cape_Town            YES
```

Note that only the label `'Dar_es_Salaam'` is cut off since it is heading a column. The label `'Johannesburg'` is not cut off as it is the label for a row. Now we add the following line to change the width of the display statement:

```
option dispwidth = 15; display f;
```

As expected, the output changes and the label `'Dar_es_Salaam'` is no longer cut off:

```
----      5 SET f  direct flight connections

                       Maputo    Dar_es_Salaam

Johannesburg              YES              YES
Cape_Town                 YES
```

4.32.4.2 Local Display Control

It is often useful to control the number of decimals for specific identifiers separately. The following option statement accomplishes this:

```
option ident:d-value;
```

`Ident` denotes the name of the identifier and `d-value` is an integer in the range 0 to 8. The number of decimal places shown on all displays of `ident` that follow will be `d-value`.

This option statement can be extended to control the layout of the data in addition to the decimal places. The general form is as follows:

```
option ident:d-value:r-value:c-value ;
```

Here `r-value` denotes the number of index positions that are combined to form the row label and `c-value` means the number of indices on the column headers.

The example discussed in the previous section is further extended in order to illustrate the local display control.

```
option x:5:3:1; display  x;
```

These two statements generate the following output:

```
----     17 PARAMETER x  a four dimensional structure

                           i          ii

first .one  .b     5.63559
first .two  .a    -2.93930     0.02873
first .two  .b                10.34570
first .three.b    -6.31610
second.one  .a     0.00013     1.00370
second.one  .b                -17.29948
second.two  .a                    +INF
second.two  .b 20000.00000
```

Note that five decimal places are shown, three labels are used to mark the rows and one label acts as head of the column. Since this is a four-dimensional structure, all indices are accounted for.

The next example places two indices on each of the row and and column labels and retains five decimal places:

```
option x:5:2:2; display x;
```

The output is:

```
----     18 PARAMETER x  a four dimensional structure

                      a.i        a.ii         b.i        b.ii

first .one                                  5.63559
first .two      -2.93930     0.02873                 10.34570
first .three                              -6.31610
second.one       0.00013     1.00370               -17.29948
second.two                      +INF 20000.00000
```

If in the option statement less dimensions are specified than the number of indices in the domain, then subtables will be generated for the labels of the beginning indices. Observe how the display layout will change if we use the following statement:

```
option x:0:1:1;  display x;
```

The output follows.

```
----     19 PARAMETER x  a four dimensional structure

INDEX 1 = first  INDEX 2 = one

            i

b           6

INDEX 1 = first  INDEX 2 = two

            i          ii

a          -3 2.873000E-2
b                      10

INDEX 1 = first  INDEX 2 = three

            i

b          -6

INDEX 1 = second  INDEX 2 = one

            i          ii

a 1.300000E-4           1
b                     -17

INDEX 1 = second  INDEX 2 = two

            i          ii

a                     +INF
b       20000
```

The option statement is checked for consistency against the dimensionality of the identifier and error messages will be issued if the number of the dimensions specified in the option statement is larger than the number of indices in the domain of the identifier.

4.32.4.3 Display Statement to Generate Data in List Format

The option statement for local display controls may be used to generate data in list format by setting the `r-value` to zero:

```
option ident:d-value:0:c-value;
```

In this case the `c-value` specifies the maximum number of items displayed on a line. The actual number will depend on the page width and the number and length of the labels.

Using the same example as in the previous sections, we change the `r-value` to zero in the option statement:

```
option x:5:0:1; display x;
option x:5:0:2; display x;
```

Note that we specified one and two columns to illustrate the impact of different `c-value` settings. These statements will generate the following output:

```
----     20 PARAMETER x  a four dimensional structure

first .one  .b.i      5.63559
first .two  .a.i     -2.93930
first .two  .a.ii     0.02873
first .two  .b.ii    10.34570
first .three.b.i     -6.31610
second.one  .a.i      0.00013
second.one  .a.ii     1.00370
second.one  .b.ii   -17.29948
second.two  .a.ii        +INF
second.two  .b.i  20000.00000


----     21 PARAMETER x  a four dimensional structure

first .one  .b.i      5.63559,    first .two  .a.i     -2.93930
first .two  .a.ii     0.02873,    first .two  .b.ii    10.34570
first .three.b.i     -6.31610,    second.one  .a.i      0.00013
second.one  .a.ii     1.00370,    second.one  .b.ii   -17.29948
second.two  .a.ii        +INF,    second.two  .b.i  20000.00000
```

This output nicely illustrates the label order used. The first index varies the slowest, the last the fastest, and each one runs from beginning to end before the next one to the left advances. This ordering scheme is also used in equation and column lists and in the solution report which are all generated by the solve statement.

4.32.5 Conditional Displays

This section assumes familiarity with the if statement and the dollar condition.

The display statement may be controlled by conditional expressions. Conditionals have the effect that the items are displayed only if a logical condition is satisfied. There are two ways to express conditional displays in GAMS: with dollar conditions and using `if` statements. The syntax using a dollar condition is as follows:

```
display$logical_condition ident | quoted text {, ident | quoted text};
```

Note that the dollar condition is inserted after the keyword `display`. For details on the various forms of conditional expressions in GAMS, see sections Logical Conditions and Filtering Sets. Consider the following simple example:

```
Scalars x, y;
x = 7;
y = 3;
display$(x-y < 0) "display if x minus y is less than 0", x, y;
```

The logical condition `(x-y < 0)` controls the display statement. In this case it is not satisfied, hence there will be no display output. If the value of `y` is changed to, say 10, then the logical condition will be satisfied and the display statement will generate the following output:

```
----      4 display if x minus y is less than 0
            PARAMETER x                    =        7.000
            PARAMETER y                    =       10.000
```

The syntax for conditional displays using an `if` statement is as follows:

```
if( logical_condition, display statement );
```

The simple example above may be reformulated as follows:

```
Scalars x, y;
x = 7;
y = 3;
if( x-y < 0,
  display "display if x minus y is less than 0", x, y;
);
```

4.33 Programming Flow Control Features

4.33.1 Introduction

In this chapter we will describe the programming flow control features available in GAMS. The if statement facilitates expressing complex conditional statements (see also chapter Conditional Expressions, Assignments and Equations
In addition, GAMS offers four loop constructs to handle looping requirements: the loop statement, the while statement, the for statement and the repeat statement. At the end of this chapter we will introduce the break, and continue statements, which give additional control over the execution of loop structures, and the abort statement, a statement that may be used to terminate the execution of a program.

Note that this chapter deals with programming flow control features at execution time. In addition, GAMS provides a dollar control option that allows for *conditional processing of input files at compile time*. For more information, see the detailed description of the option $if. For details on dollar control options in general, see chapter Dollar Control Options.

4.33.2 The If Statement

The `if` statement is useful to branch conditionally around a group of statements. In some cases this can also be written as a set of dollar conditions, but the `if` statement may make the GAMS code more readable. An optional `else` and/or `elseif` part allows the user to formulate traditional `if-then-else` constructs.

4.33.2.1 The If Statement: Syntax

The syntax for an `if` statement in GAMS is as follows:

```
if (logical_condition,
    statement; {statement;}
{ elseif logical_condition,
    statement; {statement;} }
[ else
    statement; {statement;} ]
);
```

The keyword `if` indicates that this is an `if` statement. The logical condition is followed by one or more statements that are executed if the logical condition is satisfied. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets. The `if` statement may end at this point, without any specifications for cases when the logical condition is `FALSE` and no action is taken in these cases. However, GAMS also allows further specifications: optional alternative `if` tests and optional statements for cases when all previous `if` tests have failed. The keyword `elseif` introduces an alternative `if` test with a logical condition and one or more statements. Note that the `elseif` case is optional and may be repeated multiple times. Note further that the `elseif` case implies that the logical condition of the `if` case has not been satisfied. The keyword `else` introduces the final part of the construct. It is optional and allows specification for cases when the logical condition of the `if` case (and possibly the `elseif` case(es)) has not been satisfied.

Attention

> Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside an `if` statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option onEnd.

4.33.2.2 The If Statement: Examples

Consider the following set of conditional assignment statements that use dollar conditions:

```
p(i)$(f <= 0) = -1 ;
p(i)$((f > 0) and (f < 1)) = p(i)**2 ;
p(i)$(f >= 1) = p(i)**3 ;
q(j)$(f <= 0) = -1 ;
q(j)$((f > 0) and (f < 1)) = q(j)**2 ;
q(j)$(f >= 1) = q(j)**3 ;
```

They may be expressed using an `if-elseif-else` statement:

```
if (f <= 0,
    p(i) = -1 ;
    q(j) = -1 ;
elseif ((f > 0) and (f < 1)),
    p(i) = p(i)**2 ;
    q(j) = q(j)**2 ;
else
    p(i) = p(i)**3 ;
    q(j) = q(j)**3 ;
) ;
```

Note that the body of the `if` statement may contain `solve` statements. For instance, consider the bit of GAMS code that follows. Note that `ml` is a GAMS model, `z` is a free variable, `j` is a set and `x` is a variable.

```
solve ml using lp minimizing z;
if (ml.modelstat = 4,
     display "model ml was infeasible, relax bounds on x and solve again";
     x.up(j) = 2*x.up(j) ;
     solve ml using lp minimizing z ;
else
    if (ml.modelstat <> 1,
        abort "error solving model ml" ;
    );
);
```

First the model `ml` is solved. For details on `solve` statements in GAMS, see section The Solve Statement. Then a post solution analysis is done with the `if` statement. If the model is infeasible, the upper bound on the variable `x` is relaxed and the model is solved again. If the original model is not infeasible and it is not optimal either, then the compilation is aborted and the error message above is reported. For more information on GAMS output, see chapter GAMS Output, particularly subsection Model Status for a list of all GAMS model statuses. The `display` statement is introduced in chapter The Display Statement. For details on the `abort` command, see abort.

The following GAMS code is illegal since it is not permitted to define equations inside an `if` statement.

```
if (s > 0,
    eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is also illegal since declarations inside an `if` statement are not allowed.

```
if (s > 0,
   scalar y ; y = 5 ;
);
```

4.33.3 The Loop Statement

The `loop` statement facilitates executing a group of statements for each member of a set. `Loop` statements are particularly useful for cases when parallel assignments are not sufficient. This is the case most often when there is no analytic relationship between the values to be assigned to a parameter. It is, of course, also useful to have a looping statement for general programming.

4.33.3.1 The Loop Statement: Syntax

The syntax for the `loop` statement in GAMS is as follows:

```
loop(index_list[$(logical_condition)],
     statement; {statement;}
) ;
```

The keyword `loop` indicates that this is a loop statement. The `index_list` is the controlling domain of the loop. Note that loops may be controlled by more than one set. In this case parentheses are required around the index list, which is also called the *loop set(s)*. Observe that dynamic sets are allowed as loop sets. The loop set(s) may be restricted by a logical condition. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets. For an introduction to dollar conditions in general, see chapter Conditional Expressions, Assignments and Equations. The index list is followed by one or more statements. Except for the final statement, each statement must end with a semicolon `';'`. The loop statements are executed for each member of the controlling domain in turn. The order of evaluation is the entry order of the labels. A loop is thus another, more general, type of indexed operation.

Attention

- Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a `loop` statement.
- It is illegal to modify any controlling set inside the body of the loop.

`Loop` statements in GAMS are often used for iterative calculations, generating reports with put statements and doing scenario based studies with solve statements. Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option onEnd.

4.33.3.2 The Loop Statement: Examples

Consider a hypothetical case when a growth rate is empirical:

```
Set        t          / 1985*1990 /;
Parameter  pop(t)     / 1985  3456 /
           growth(t)  / 1985  25.3,  1986  27.3,  1987  26.2
                        1988  27.1,  1989  26.6,  1990  26.6 /;
```

The `loop` statement is then used to calculate the cumulative sums iteratively:

```
loop(t, pop(t+1)  = pop(t) + growth(t) ) ;
```

In this example the driving set is `t` and we have just one statement in the scope of the loop.

The following small artificial examples illustrate the effects of a dollar condition in a `loop` statement.

```
Sets       i       / i1*i3 /
           j       / j1*j5 /
           k(i,j)  / i1.j1, i1.j3, i3.j3, i3.j5 /;
Parameter  c(i)    / i1 3, i2 1 /
           q(i,j)  / i1.j1 1, i1.j2 3, i1.j4 2 / ;
Scalars    x, y, z;
x = 1; y = 3; z = 1;

loop ( (i,j) $ (q(i,j) > 0), x = x + q(i,j));
loop ( i $ (c(i) + c(i)**2), z = z + 1);
loop ( i $ sum(j, abs(q(i,j))), z = z + 1);
loop ( j $ (ord(j) > 1 and ord(j) < card(j)), z = z + 1);
loop ( (i,j) $ k(i,j), y = y + ord(i) + 2*ord(j));
```

In the first `loop` statement the controlling domain is the set pair `(i,j)`. It is restricted to those label combinations whose values associated with the parameter `q` are greater than zero. The logical condition in the second `loop` statement is $c(i) + c(i)^2$. This is shorthand for $c(i) + c(i)^2 \neq 0$. The domain in the third `loop` statement is restricted to those elements of the set `i` where the sum over `j` of the absolute values of the parameter `q` does not equal zero. Note that this condition is satisfied only for the label `"i1"`. Observe that `i` and `j` are both ordered sets. In the fourth `loop` statement the first and the last element of the set `j` are excluded. For more on the set operators `ord` and `card`, see sections The Ord Operator and The Card Operator respectively. The dollar control in the last `loop` statement excludes all label combinations that are not members of the set `k`. For further details on dollar conditions, see chapter Conditional Expressions, Assignments and Equations.

Note

The dollar condition may be replaced by an if statement; see the example below.

The next example shows how a model can be solved for each element of a set `i` with different data using a `loop` statement. Note that `problemdata` is a scalar, `g(i)` and `d(i)` are parameters, `mymodel` is a GAMS model and `profit` is a free variable.

```
loop (i,
  problemdata = g(i);
  solve mymodel using lp maximizing profit;
  d(i) = profit.l;
);
```

In the first statement some data in the model is updated in accordance with the i^{th} element of the parameter `g`. In the second statement the model is solved. For details on the `solve` statement, see section The Solve Statement. The objective value for each iteration is saved in the parameter `d` in the third statement.

A loop is often used to perform iterative calculations. Consider the following example, which finds square roots by Newton's method. This example is purely for illustration - in practice, the function `sqrt` should be used. Newton's method is based on the assertion that if x is an approximation to the square root of v, then $(x + v/x)/2$ is a better approximation.

```
Set       i         "set to drive iterations"                 / i-1*i-100 /;
Parameter value(i)  "used to hold successive approximations";
Scalars   target    "number whose square root is needed"      / 23.456 /
          sqrtval   "final approximation to sqrt(target)"
          curacc    "accuracy of current approximation"
          reltol    "required relative accuracy"              / 1.0e-06 /;
```

```
abort$(target <= 0) "argument to newton must be positive";
value("i-1") = target/2  ;
curacc       =  1  ;
loop(i$(curacc > reltol),
       value(i+1) =  0.5*(value(i) + target/value(i));
       sqrtval    =  value(i+1);
       curacc     =  abs (value(i+1)-value(i))/(1+abs(value(i+1)))
) ;
abort$(curacc > reltol) "square root not found"
option decimals=8;
display  "square root found within tolerance",  sqrtval, value;
```

Note that in this example the dollar condition in the loop does not restrict the driving set `i`, but it is used to terminate the loop procedure. The scalar `curacc` is updated in every iteration. As soon as it becomes equal to or smaller than the required relative accuracy `reltol` the loop stops. As the output below shows, this is the case after seven iterations. The body of the `loop` statement consists of three statements. The first statement calculates the current approximation and assigns it to the parameter `value`. The second statement updates the scalar `sqrtval`, and the third statements computes the accuracy of the current approximation in each iteration. Note that before and after the loop statement we added lines to account for special cases. For details on the `abort` statement, see section The Abort Statement at the end of this chapter. The output generated by the display statement is given below.

```
----     19 square root found within tolerance

----     19 PARAMETER SQRTVAL              =   4.84313948 final approximation
                                                 to sqrt(target)

----     19 PARAMETER VALUE         used to hold successive approximations

i-1 11.72800000,    i-2  6.86400000,    i-3  5.14062471,    i-4  4.85174713
i-5  4.84314711,    i-6  4.84313948,    i-7  4.84313948
```

Note that a statement within the body of a loop may be an if statement (or any other programming flow control statement). Moreover, the logical condition in a `loop` statement may be expressed with an `if` statement instead of a dollar condition. The following example serves as illustration. Observe that `k` is a set and `s`, `t`, `u` and `a` are parameters.

```
loop (k,
   if((s(k) < 0 and t(k)),
      u(k) = a(k);
   );
);
```

Note that if the logical condition is not satisfied the assignment is not made and the parameter `u` remains unchanged.

Recall that subsets are connected with their supersets by arcs thus building a domain tree where the root node is the universal set. The following example demonstrates how the domain tree may be used in a loop statement.

```
Set i       / i1*i10 /
    ii(i)
    j(i)    / i1*i9 /
    jj(j)   / i1*i8 /
    jjj(jj) / i1*i7 /;

loop(i(jjj), ii(i) = yes;);
display ii;
```

Observe that the looping set is `i(jjj)`. This means that we loop over those elements of `i` that are also elements of the set `jjj`. This construct is permitted since `i` is in the domain tree on the path from `jjj` to the universe or universal set. It is allowed to go up the domain tree on one path and go down on another path. Therefore all the elements of `jjj` are assigned to `ii`. The outcome of the display statement confirms this:

```
----      8 SET ii

i1,    i2,    i3,    i4,    i5,    i6,    i7
```

4.33.4 The While Statement

The `while` statement facilitates the repeated execution of one or more statements as long as a logical condition is satisfied.

4.33.4.1 The While Statement: Syntax

The syntax for the `while` statement in GAMS is as follows:

```
while(logical_condition,
      statement; {statement;}
);
```

The keyword `while` indicates that this is a `while` statement. Inside the `while` statement a logical condition is followed by one or more statements. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets. The statements are executed as long as the logical condition is `TRUE`.

Attention

> Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a `while` statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option onEnd.

4.33.4.2 The While Statement: Examples

Consider the following simple example:

```
Scalar x; x = 1;
while ( round(x,2) < 10,
   x = x + 0.01;
);
display x;
```

Note that the scalar x is increased in each iteration until it equals 10. Note further, that to ensure an exact result, in numerical comparisons we need a stable check like we have above (round(x,2) < 10), otherwise rounding errors may occur. As soon as x reaches 10, the logical condition is no longer satisfied and therefore there will be no further passes. Hence the final value of x equals 10.

Note that the number of passes in a while statement may be restricted using the command line parameter or option forlim. For details on command line parameters and options, see section Specifying Options Through the Command Line and chapter The Option Statement respectively.

While statements may be used to control the solve statement. For instance, consider the following implementation of a random multi-start method for non-convex optimization.

```
scalar count   /  1  /;
scalar globmin / inf /;
option bratio = 1 ;
while(count <= 1000,
      x.l(j) = uniform(x.lo(j),x.up(j)) ;
      solve ml using nlp minimizing obj ;
      if (obj.l < globmin,
          globmin = obj.l ;
          bestsol(j) = x.l(j) ;
      ) ;
      count = count+1 ;
) ;
```

Note that we start from a random starting point by setting the initial level values randomly between the upper and lower bounds. This assumes that the bounds have been previously specified and are not infinity. When the method improves, that is, if the logical condition (obj.l < globmin) is satisfied, the best known solution is stored in the scalar globmin. The level values associated with the best known solution so far are then saved in the parameter bestsol. This procedure is repeated 1000 times. The model [**PRIME**] is another example where the use of the while statement is illustrated. In this model the set of all prime numbers smaller than 200 is generated.

The following GAMS code is illegal since equation definitions inside a while statement are not permitted.

```
while(s > 0,
      eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is also illegal since declarations inside a while statement are not allowed.

```
while(s > 0,
      scalar y ; y = 5 ;
);
```

4.33.5 The For Statement

The `for` statement provides a compact way to iterate over a range of values and execute one or more statements each time.

4.33.5.1 The For Statement: Syntax

The syntax for the `for` statement in GAMS is as follows:

```
for (a = start_value to|downto end_value [by incr],
     statement; {statement;}
);
```

The keyword `for` indicates that this is a `for` statement. The scalar `a` begins with the real number `start_value` and is changed after each pass of the loop by the increment `incr` until it reaches the real number `end_value`. Note that the specification of an increment is optional, the default is 1. If the increment is given, it has to be a positive real number. Note further that `to` indicates that the scalar `a` is *increased* and `downto` indicates that it is *decreased*. In each iteration one or more statements are executed.

Attention

> Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a `for` statement.

Examples are given in the next subsection. Note that there is an alternative syntax that is more in line with the syntax of some popular programming languages. For more information, see the detailed description of the dollar control option onEnd.

4.33.5.2 The For Statement: Examples

Consider the following simple example:

```
Scalar s;
for (s = -3.8 to -0.1 by 1.4,
     display s ;
);
```

Note that *negative* real numbers are possible for the start and end values. The resulting listing file will contain the following lines:

```
----      3 PARAMETER s                  =       -3.800
----      3 PARAMETER s                  =       -2.400
----      3 PARAMETER s                  =       -1.000
```

Observe that the value of `s` was increased by 1.4 with each pass of the loop as long as it did not exceed -0.1. In the next example the value of `s` is *decreased:*

```
Scalar s;
for (s = 3 downto -0.1 by 1.4,
     display s ;
);
```

Note that the number of passes in a `for` statement may be restricted using the command line parameter or option forlim. For details on command line parameters and options, see section Specifying Options Through the Command Line and chapter The Option Statement respectively.

Like `while` statements, `for` statements may be used to control the `solve` statement. The following example illustrates the use of the `for` statement by replicating the random search for a global optimum of a non-convex model that we discussed above.

```
scalar i ;
scalar globmin / inf / ;
option bratio = 1 ;
for (i = 1 to 1000,
     x.l(j) = uniform(x.lo(j),x.up(j)) ;
     solve m1 using nlp minimizing obj ;
     if (obj.l < globmin,
         globmin = obj.l ;
         bestsol(j) = x.l(j) ;
     );
);
```

Note that the logical condition in the `while` loop (`count <= 1000`) is replaced by the specification of the range of values for the scalar `i`. The body of the `for` loop is identical to the body of the `while` loop, except for the statement to update the scalar `count` that we needed in the `while` loop. This demonstrates the similarities and differences between the two loops.

The following GAMS code is illegal since it is not allowed to define equations inside a `for` statement.

```
for (s = 1 to 5,
     eq.. sum(i,x(i)) =g= 2 ;
);
```

The following GAMS code is also illegal since declarations inside a `for` statement are not permitted.

```
for (s = 1 to 5,
     scalar y ; y = 5 ;
);
```

4.33.6 The Repeat Statement

The `repeat` statement facilitates the repeated execution of one or more statements. This is done unconditionally at least once and stopped when a logical condition is satisfied.

4.33.6.1 The Repeat Statement: Syntax

The syntax for the `repeat` statement in GAMS is as follows:

```
repeat (
    statement; {statement;}
until logical_condition );
```

The keyword `repeat` indicates that this is a `repeat` statement. One or more statements are executed in each iteration. The keyword `until` introduces the termination criterion: if the logical condition is satisfied, the `repeat` loop is terminated. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets.

Note that the `repeat` statement is similar to the `while` statement, but a `repeat` loop is guaranteed to be executed at least one time since the logical condition is stated *after* the statements.

Attention

> Only *execution* statements are permitted in programming flow control statements. Consequently declaration statements and equation definitions are not allowed inside a `repeat` statement.

Examples are given in the next subsection.

4.33.6.2 The Repeat Statement: Examples

Consider the following simple example:

```
Scalar a / 1 /;
repeat (
    a = a + 1;
    display a;
until a = 5 );
```

The scalar `a` is increased in each iteration by 1. If `a` equals 5, the termination criterion is satisfied and the loop stops. Note that this example works nicely since both, the scalar and the increment, are integer. In case the entity on the right-hand side of the termination condition is not an integer or the increment is not an integer, we recommend a formulation of the check that is more stable to avoid rounding errors. An example for a stable termination condition follows.

```
Scalar a / 1 /;
repeat (
    a = a + 0.1;
    display a;
until abs(a-5) < 1e-6 );
```

Observe that in the next example the termination condition is `TRUE` from the start. In this case the statement in the body of the `repeat` statement is executed once and then the loop is terminated. Hence the final value of `a` will be 5.

```
Scalar a / 4 /;
repeat (
    a = a + 1;
    display a;
until a >= 3 );
```

Note that the number of passes in a while statement may be restricted using the command line parameter or option forlim. For details on command line parameters and options, see section Specifying Options Through the Command Line and chapter The Option Statement respectively.

Here is a little more complex example. A `repeat` statement is used to narrow the interval where a quadratic function passes through zero. Note that, as is often the case, one of the statements in the `repeat` loop is another programming flow control statement, in this case an `if` statement.

```
Scalar max             "current upper boundary of interval"   /10/
       min             "current lower boundary of interval"   /-10/
       root            "value where function equals zero"
       function_value1 "function value at min"
       function_value2 "function value at max"
       tolerance       "tolerance for root"                   /0.00000001/
       signswitch      "indicates that sign switch was found" /0/
       inc             "increment to try to find sign switch" ;

function_value1 = 6 - 5*min + sqr(min);
inc             = (max - min)/37;
root            = min;

repeat (
   root = root + inc;
   function_value2 = 6 - 5*root + sqr(root);

   if((    sign(function_value1) <> sign(function_value2)
       and abs(function_value1) > 0
       and abs(function_value2) > tolerance ),
      max = root;
      signswitch = 1;
   else
      if(abs(function_value2) > tolerance,
         function_value1 = function_value2;
         min = root;
      );
   );
until (signswitch > 0) or (root > max) );
display min, max, function_value1, function_value2;
```

The result of the `display` statement shows that the value of `min` is 1.892 and the value of `max` is 2.432, the interval was narrowed to 0.54, which is just a little less than the value of `inc`. As expected, `function_value1` and `function_value2` differ in sign, confirming that the root of the quadratic function is indeed in the interval.

4.33.7 The Break Statement

The `break` statement gives additional control over the execution of loop structures, namely the loop statement, the while statement, the for statement and the repeat statement. It allows to break the execution of a loop structure prematurely.

4.33.7.1 The Break Statement: Syntax

The syntax for the `break` statement is as follows:

```
break [n];
```

The keyword `break` indicates that this is a break statement. It terminates the `n` inner most control structures. `n` is optional and if it is omitted, it is set to 1.

Most often `break` statements are used in the context of if statements or with dollar conditions. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets.

4.33.7.2 The Break Statement: Examples

This is a simple, artificial example using the `break` statement to exit a `loop` statement:

```
Set    i   / i1*i10 /;
Scalar cnt / 0 /;

loop(i,
  break$sameas('i6',i);
  cnt = cnt+1;
);

display cnt;
```

The `break` statement in combination with the dollar condition terminates the execution of the loop body in the 6th iteration. This can be seen looking at the value of `cnt` after the loop, which will be 5.

Here is a little more complex example which uses the optional argument `n` to terminate more than one loop structure at once:

```
Set    i   / i1*i10 /;
Scalar x, y,
       cnt / 0 /;

for(x = 1 to 10,
  y = 0;
  while(y < 10,
    y = y+1;
    loop(i,
      break$sameas('i6',i) 2;
      cnt = cnt+1;
    );
  );
);
display cnt;
```

In this example, `cnt` will be 50 at the end. As in the previous example, it is increased 5 times in the inner most loop, before the `break` statement is executed. This time `break` is called with the argument `2`, which causes the two inner most control structures (namely the `loop` and the `while`) to be terminated. The outer most control structure (the `for`) is not influenced, so that its body gets executed 10 times, which results in a total of 50 increments for the scalar `cnt`.

Note, if `break` would be called with the argument 3 instead of 2, also the `for` loop would be terminated, so that `cnt` would be 5 at the end. If `break` would be called with 1 (or without additional argument) instead of 2, only the inner most `loop` would be terminated so that `cnt` would be 500 at the end.

4.33.8 The Continue Statement

The `continue` statement gives additional control over the execution of loop structures, namely the loop statement, the while statement, the for statement and the repeat statement. It allows to jump to the end of the inner most loop structure without executing the remaining statements in the body.

4.33.8.1 The Continue Statement: Syntax

The syntax for the `continue` statement is as follows:

```
continue;
```

The keyword `continue` indicates that this is a continue statement. It jumps to the end of the inner most control structure.

Most often `continue` statements are used in the context of if statements or with dollar conditions. For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets.

4.33.8.2 The Continue Statement: Examples

This is a simple, artificial example using the `continue` statement to skip parts of a `loop` body:

```
Set    i   / i1*i10 /;
Scalar cnt / 0 /;

loop(i,
  continue$(mod(ord(i),2)=0);
  cnt=cnt+1
);

display cnt;
```

In that example, every 2nd iteration of the `loop` statement is skipped. Therefore, `cnt` will be 5 at the end of the `loop`.

4.33.9 The Abort Statement

The `abort` statement is used to terminate the *execution* of a program. Most often the abort statement is used in the context of conditionals. Examples are given below. Also it may be used to display a text or an identifier in the listing file similar to the display statement but mostly to present the reason for the termination of the execution.

Note that the `abort` statement is to be distinguished from the dollar control option $abort which may be used to terminate the *compilation* of a program.

4.33.9.1 The Abort Statement: Syntax

The syntax for the `abort` statement is as follows:

```
abort ident | quoted text {, ident | quoted text};
```

The keyword `abort` indicates that this is an abort statement. `Ident` denotes an identifier. If the identifier is a set or a parameter, only the name of the set or parameter itself is specified, without any domains. If the identifier is a variable, an equation or a model, it must be followed by a suffix, since only *attributes* of variables, equations and models can be displayed. For more on variable and equation attributes including full lists, see sections Variable Attributes and Equation Attributes respectively. For details on model attributes, see section Model Attributes. Recall that sets also have attributes, they may also be displayed using the suffix notation. For details on set attributes, see section Set Attributes. As usual, `quoted text` must be surrounded by single or double quotes. The identifiers and the text may be mixed and matched in any order, and the whole statement may be continued over several lines.

An abort statement causes the termination of the execution with an execution error and the information in the statement will be displayed.

There is also a variant with the extension `.noError` that terminates the execution and displays the information, but does not cause an execution error. The syntax is as follows:

```
abort.noError ident | quoted text {, ident | quoted text};
```

Most often `abort` statements are used in the context of if statements or with dollar conditions. The syntax is as follows:

```
if (logical_condition, abort ident | quoted text {, ident | quoted text}; );
```

or

```
abort$logical_condition ident | quoted text {, ident | quoted text};
```

For details on the various forms of logical conditions in GAMS, see sections Logical Conditions and Filtering Sets.

4.33.9.2 The Abort Statement: Examples

Consider the following artificial example:

```
Set       i    / i1*i5 /;
Parameter p(i) / i1 1, i2 2, i3 3, i4 5, i5 8 /;
loop(i,
   if (p(i) > 3, abort "Parameter larger than 3", p);
   p(i) = p(i) + 2;
);
```

Note that the abort statement is part of an if statement which is part of a loop statement. The execution of this program will be terminated with the following display and error message:

```
----      4 Parameter larger than 3

----      4 PARAMETER p

i1 3.000,    i2 4.000,    i3 5.000,    i4 5.000,    i5 8.000

**** Exec Error at line 4: Execution halted: abort 'Parameter larger than 3'
```

Observe that the values of `p('1')`, `p('2')` and `p('3')` were updated, but the values of `p('4')` and `p('5')` are equal to the initialization values since the program was terminated before they could be updated.

If the extension `.noError` is used in the example above, the following lines will appear in the listing file:

```
----      4 Parameter larger than 3

----      4 PARAMETER p

i1 3.000,    i2 4.000,    i3 5.000,    i4 5.000,    i5 8.000

****
**** Execution halted from line 4: abort.noError 'Parameter larger than 3'
****
```

Note that the execution of the program is aborted as before, but there is no execution error in this case.

Instead of an if statement we may use a dollar condition in the loop:

```
Set       i    / i1*i5 /;
Parameter p(i) / i1 1, i2 2, i3 3, i4 5, i5 8 /;
loop(i,
   abort$(p(i) > 3) "Parameter larger than 3", p;
   p(i) = p(i) + 2;
);
```

Observe that this alternative formulation has the same outcome as above.

4.34 The Option Statement

4.34.1 Introduction

The `option` statement is used to set various global system parameters to control among other things output detail, the solution process and the layout of displays. GAMS provides default values for global system parameters that are adequate for the most purposes. However, there are always cases when the user would like to maintain control of some aspects of the run. In addition, the option statement provides an efficient and compact syntax to perform powerful operations on identifiers. Observe that option statements are processed at execution time unlike dollar control options that are processed at compile time.

This chapter is organized as follows. We will first introduce the general syntax of the option statement, then we will continue with a list of all available options that may be used with option statements and provide links to their detailed explanations in the GAMS Call chapter. Finally we turn to the special options that involve identifiers.

4.34.1.1 The Syntax of the Option Statement

The general form of an option statement is as follows:

```
option(s) key1 [= value1] { ,|EOL  key2 [= value2] } ;
```

The keyword `option` or `options` indicates that this is an option statement. It is followed by `key1`, which is one of the options that are listed in this chapter. Consider the following simple example:

```
option reslim=800;
```

Here the keyword `option` is followed by the `key` reslim. The option `reslim` specifies the maximum time in seconds that a solver may run before it terminates. Thus, in this example we give the solver 800 seconds to come up with a solution.

Note

- Option names are not reserved words and therefore they do not conflict with other uses of their names.
- Option statements do not allow expressions. It is therefore not possible to assign parameter values to an option.
  The following doesn't work:

  ```
  scalar p /3/;
  option reslim=p;
  ```

  However, if the desired option is available as model attribute, one can assign parameter values there:

  ```
  scalar p /3/;
  transport.reslim = p;
  ```

Observe that it depends on the respective option whether a value is expected and if so, what type of value. There are six different cases. An overview is given in Table 1.

Key	Value	Type of Value	Examples
yes	no	-	dmpOpt, eject, memoryStat
yes	yes	integer number	decimals, limcol, seed
yes	yes	real number	FDDelta, optCR, resLim
yes	yes	text string	LP, solPrint, sysout
yes	yes	identifier	See identifier options.
no	no	-	See identifier operations.

Table 1: Types of Options

Note that the last type is special: it does not involve a named option or `key`, but the keyword `option` is followed by identifiers and identifier operators. These special option statements are discussed in detail in section Special Options: Identifier Operations.

Observe that the value of an option may be reset as often as necessary, the new value will replace the previous value each time. Further, more than one option may be specified with one option statement and commas or end-of-line characters are both legal separators between options.

We will demonstrate with the following example how several options may be used. The code snippet may be added to the model [**DICE**].

```
option measure, limcol = 100
       optcr = 0.00, mip = xpress ;
solve xdice using mip max wnx;
option clear = comp;
```

Note that in the first option statement four options are specified: option measure has no associated value, option limcol expects an integer value, option optcr expects a real value and option MIP expects a text string as value. The second option statement specifies just one identifier option: clear, which has the variable `comp` as value.

Attention

> Option statements are executed in sequence with other instructions. Therefore, if an option statement is located between two solve statements, the new values will be assigned between the solves and thus they will apply only to the second solve statement.

4.34.2 List of Options

The options available through the option statement are grouped into the following functional categories:

- Options that affect output details
- Solver specific parameters
- Options that control choice of solver
- Options that affect input program control
- Other options
- Special options that involve identifiers

In the following subsections we will offer brief descriptions of the options in the first five categories. Note that each entry is linked to a detailed description of the respective option. Observe that detailed descriptions of all GAMS command line parameters, options and model attributes are given in section Detailed Descriptions of All Options. The options that belong to the last category are special, they are introduced and discussed in section Special Options that Involve Identifiers below.

4.34.2.1 Options that Control Output Details

Option	Description
asyncSolLst	Print solution listing when asynchronous solve (Grid or Threads) is used
decimals	Decimal places for display statements
dispWidth	Number of characters to be printed in the column labels of all subsequent display statements
eject	Inject a page break into the LST file
epsToZero	Treat eps as zero when unloading to GDX
limCol	Maximum number of columns listed in one variable block
limRow	Maximum number of rows listed in one equation block

Option	Description
maxGenericFiles	Maximum number of generic file names tried at execution time file creation
MCPRHoldFx	Print list of rows that are perpendicular to variables removed due to the holdfixed setting
profile	Execution profiling
profileTol	Minimum time a statement must use to appear in profile generated output
solPrint	Solution report print option
solSlack	Causes the equation output in the listing file to contain slack variable values instead of level values
sysOut	Solver Status file reporting option

4.34.2.2 Options that Control Solver-Specific Parameters

Option	Description
bRatio	Basis detection threshold
domLim	Domain violation limit solver default
holdFixedAsync	Allow HoldFixed for models solved asynchronously as well
intVarUp	Set mode for default upper bounds on integer variables
iterLim	Iteration limit of solver
optCA	Absolute Optimality criterion solver default
optCR	Relative Optimality criterion solver default
reform	Reformulation level
resLim	Wall-clock time limit for solver
savePoint	Save solver point in GDX file
solveLink	Solver link option
sys12	Pass model with generation errors to solver
threads	Number of processors to be used by a solver

4.34.2.3 Options that Control the Choice of Solver

Option	Description
CNS	Constrained Nonlinear Systems - default solver
DNLP	Non-Linear Programming with Discontinuous Derivatives - default solver
EMP	Extended Mathematical Programs - default solver
LP	Linear Programming - default solver
MCP	Mixed Complementarity Problems - default solver
MINLP	Mixed-Integer Non-Linear Programming - default solver
MIP	Mixed-Integer Programming - default solver
MIQCP	Mixed Integer Quadratically Constrained Programs - default solver
MPEC	Mathematical Programs with Equilibrium Constraints - default solver
NLP	Non-Linear Programming - default solver
QCP	Quadratically Constrained Programs - default solver
RMINLP	Relaxed Mixed-Integer Non-Linear Programming - default solver
RMIP	Relaxed Mixed-Integer Programming - default solver
RMIQCP	Relaxed Mixed Integer Quadratically Constrained Programs - default solver
RMPEC	Relaxed Mathematical Programs with Equilibrium Constraints - default solver
solver	Default solver for all model types that the solver is capable to process

4.34.2.4 Options that Affect Input Program Control

Option	Description
ECImplicitLoad	Allow implicit loading of symbols from embedded code or not
fdDelta	Step size for finite differences
fdOpt	Options for finite differences
gdxUels	Unload labels or UELs to GDX either squeezed or full
seed	Random number seed
solveOpt	Multiple solve management
strictSingleton	Error if assignment to singleton set has multiple elements
sys18	Use backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers
zeroToEps	Treat zero as eps

4.34.2.5 Other Options

Option	Description
checkErrorLevel	Check errorLevel automatically after executing external program
dmpOpt	Debugging option: causes GAMS to echo the runtime option settings
dmpSym	Debugging option: causes GAMS to echo the symbol table to the listing file
dmpUserSym	Debugging option: causes GAMS to echo the symbol table to the listing file for user defined symbols only
dualCheck	Output on the reduced cost condition
forLim	GAMS looping limit
integer1..5	Integer communication cells
measure	Output of time and memory use since the last measure statement or the program beginning
memoryStat	Show memory statistics in the LST file
real1..5	Real communication cells
subSystems	Lists all solvers available as well as the current default and active solvers in the LST file
sys10	Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer
sys11	Dynamic resorting if indices in assignment/data statements are not in natural order
sys15	Automatic switching of data structures used in search records
sys16	Disable search record memory (aka execute this as pre-GAMS 24.5)
sys17	Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5)
sys19	Disable permutation on Column Generation (aka execute this as pre-GAMS 36)
threadsAsync	Limit on number of threads to be used for asynchronous solves (solveLink=6)

4.34.2.6 Special Options that Involve Identifiers

Several options involve identifiers: they either take identifiers as values or have no key and value, but perform an operation on an identifier. In the following two subsections we will introduce and discuss these special options.

Special Options: Identifier Options

The value of identifier options is not a string or a number, but an identifier. In this subsection we will describe these options in detail.

clear

This option resets an identifier to its default value to free memory. The syntax is as follows:

```
option clear = identifier;
```

The following identifier types may be reset: sets, parameters, equations and variables. The option will free up memory to the GAMS heap manager, thus the memory may be used by GAMS but not by the operating system. To force the memory to be freed up to the operating system, the GAMS process needs to be terminated. One way to do this is to restart the execution system by solving a dummy model with the option solveLink set to zero.

kill

This option is a synonym to option clear. Observe that the dollar control option $kill is *not* a synonym to $clear.

shuffle

This option rearranges the values of a parameter in a random order. The syntax is as follows:

```
option shuffle = itemname;
```

Here `itemname` is a one-dimensional parameter. One-dimensional parameters may be declared (and defined) in four different ways, depending on the domain and the data. The following table gives an overview of the effect of the option `shuffle` in the four cases.

	No Data	Has Data
Universal set as domain	Use the universal set to initialize the data (case 1 in the example below).	Use the universal set to add zero values before shuffling the data (case 3 in the example below).
Specific set as domain	Use the domain to initialize the data (case 2 in the example below).	Use the domain to add zero values before shuffling the data (case 4 in the example below).

If the parameter was declared without data (the second column in the table above), the domain or the universal set will be used to assign the numbers 1 to `N`, where `N` is the number of elements in the domain or the universal set. If the parameter was declared with data (the third column in the table above), the domain or the universal set will be used to add zeroes for possibly missing entries. These zero values will participate in the random shuffle, but they will not be stored in the parameter. The following example serves as illustration:

```
Set i / i1*i5 /
    j / j1*j5 /;
option decimals = 0;
```

```
*Case 1: universal set as domain and no data
Parameter A(*) "The universe is used to fill the parameter";
option shuffle = A;
display A;

*Case 2: set j as domain and no data
Parameter B(j) "The set j is used to fill the parameter";
option shuffle = B;
display B;

*Case 3: universal set as domain and has data
Parameter C(*) "The universe is used to add zeroes" / j2 2, j4 4 /;
option shuffle = C;
display C;

*Case 4: set i as domain and has data
Parameter D(i) "The set i is used to add zeroes" / i1 10, i3 30, i5 50 /;
option shuffle = D;
display D;
```

The code above will generate the output that follows. Observe that in this example, the universal set is the union of the sets `i` and `j`, which means that all elements of the sets `i` and `j` are members of the universal set. Note that we use random numbers, therefore the outcomes from a different run may vary.

```
----      9 PARAMETER A  The universe is used to fill the parameter

i1  4,    i2  1,    i3  7,    i4  9,    i5  6,    j1 10,    j2  3,    j3  5
j4  8,    j5  2


----     14 PARAMETER B  The set j is used to fill the parameter

j1 1,    j2 5,    j3 2,    j4 4,    j5 3


----     19 PARAMETER C  The universe is used to add zeroes

j1 2,    j2 4


----     24 PARAMETER D  The set i is used to add zeroes

i2 30,    i4 50,    i5 10
```

In the next example we will demonstrate how to generate a random mapping of a set:

```
Set i        / i1*i6 /,
    rmi(i,i) "random mapping";
Parameter A(i);
option shuffle = A;

rmi(i, i + (- Ord(i) + A(i))) = yes;
display rmi;
```

A display of the set `rmi` follows. Note that there is exactly one element in each row and each column:

```
----      7 SET rmi  random mapping

            i1          i2          i3          i4          i5          i6

i1                                                                      YES
i2                                              YES
i3          YES
i4                                  YES
i5                                                          YES
i6                      YES
```

Observe that each use of the option `shuffle` will generate a new random data rearrangement.

Special Options: Identifier Operations

In some cases the keyword `option` in an option statement is followed by an identifier and one or more operators to perform identifier operations like display control, index matching, projection and aggregation of sets and parameters and permutation of sets and parameters.

Display Control

The display statement is introduced and discussed in chapter The Display Statement. While GAMS provides defaults for the displayed identifiers that suffice in most cases, the print format may be customized with the following option statement:

```
option ident:d;
option ident:d:r:c;
```

The keyword `option` is followed by the name of an identifier `ident`, a colon and an integer value `d`. Note that `d` may be between 0 and 8 and specifies the number of decimal places that will be displayed for the respective identifier. The specifications `r` and `c` are optional. They denote the number of index positions printed as row labels and the number of index positions printed as column labels respectively. Note that if `r` is zero, a list format will be used. For more information and examples, see sections Local Display Control and Display Statement to Generate Data in List Format.

Index Matching

Index matching is a very compact way to define multi-dimensional sets. The general syntax is as follows:

```
option set_name(index1:index2[:index3:...]);
```

The keyword `option` is followed by the name of the set, `set_name`, and two (or more) indices in parentheses that are linked with the matching operator `':'`. Note that the set must have been declared earlier in the program. If the set has also been defined earlier in the program, it will be cleared first and then the matching operation will be processed. Note furthermore that the list of identifiers may be expanded to more than two and that besides the matching operator `':'` also `','` may be used and will be interpreted as product operator. Consider the following example which also makes use of display control:

```
Set i / i1,i2/
    j / j1,j2,j3 /
    k / k1*k5 /
    ij(i,j), ijk(i,j,k), kij(k,i,j);
* index matching
Option ij(i:j), ijk(i,j:k), kij(k:i,j);
* display control
Option ij:0:0:1, ijk:0:0:1, kij:0:0:1;
Display ij, ijk, kij;
```

In its simplest form the matching operator is used to create the two dimensional set `ij`.

```
----      9 SET ij

i1.j1
i2.j2
```

Note that each member of the index `i` has been matched with a member of the index `j` until one of the indices ran out of members.

The index matching operations to define the three-dimensional sets `ijk` and `kij` illustrate a more sophisticated usage of the index operator. The sets `ijk` and `kij` are built with the indices from left to right using the product operator when a `','` is encountered or the matching operator when a `':'` is found.

For `ijk` the first operator is the `','` which is interpreted as product operator for the sets `i` and `j` and hence builds the Cartesian product of the two sets which has six 2-tuples as elements (`i1.j1, i1.j2, i1.j3, i2.j1, i2.j2, i2.j3`). The matching operator `':'` is then applied to match those 2-tuples with the five set elements in `k`. The resulting sets are:

```
----      9 SET ijk

i1.j1.k1
i1.j2.k2
i1.j3.k3
i2.j1.k4
i2.j2.k5


----      9 SET kij

k1.i1.j1
k1.i1.j2
k1.i1.j3
k2.i2.j1
k2.i2.j2
k2.i2.j3
```

The previous example can be extended to define sets of even higher dimension in the following way:

```
set h / h1*h4 /
    hijk_1(h,i,j,k)
    hijk_2(h,i,j,k);
* index matching
Option hijk_1(h:ijk);
Option hijk_2(h:i,j:k);
* display control
Option hijk_1:0:0:1, hijk_2:0:0:1;
Display hijk_1, hijk_2;
```

Note that sets hijk_1 and hijk_2 will be different even though hijk_1 uses set ijk and hijk_2 uses the same matching and product operation used at definition of ijk but spelled out. As already mentioned above, the matching operator builds up the sets with the indices from left to right. Hence, as set ijk is build first and then used on the right of the matching operator the two sets are built up differently.

```
----     19 SET hijk_1

h1.i1.j1.k1
h2.i1.j2.k2
h3.i1.j3.k3
h4.i2.j1.k4


----     19 SET hijk_2

h1.i1.j1.k1
h1.i1.j2.k2
h1.i1.j3.k3
h2.i2.j1.k4
h2.i2.j2.k5
```

Projection and Aggregation of Sets and Parameters

In GAMS, projection and aggregation operations on sets may be performed in two different ways: with an assignment and the sum operator, and with an option statement.

Using an assignment and the sum operator is the slower but more intuitive way. Assignments and the sum operator are introduced and discussed in detail in chapter Data Manipulations with Parameters and section Indexed Operations respectively. Here we only show how they may be used in the context of sets to perform projections and aggregations. The following example serves as illustration.

```
Sets i            / i1*i3 /
     j            / j1*j2 /
     k            / k1*k4 /
     ijk(i,j,k)   / #i.#j.#k /
     ij1a(i,j)
     ij1b(i,j);

Scalars Count_1a, Count_1b, Count_2a, Count_2b;

* Method 1: Using an assignment and the sum operator for a projection
ij1a(i,j) = sum(k,ijk(i,j,k));

* Method 1: Using an assignment and the sum operator for aggregations
Count_2a  = sum(ijk(i,j,k),1);
Count_1a  = sum(ij1a(i,j),1);
```

Note that the set ijk is a three-dimensional set, its elements are 3-tuples and all permutations of the elements of the three sets i, j and k are in its domain. Thus the number of elements of the set ijk is 3 x 2 x 4 = 24. The sets ij1a and ij1b are two-dimensional sets that are declared in the set statement, but not defined. The first assignment statement defines the members of the set ij1a. This is a projection from the set ijk to the set ij1a where the three-tuples of the first set are mapped onto the pairs of the second set, such that the dimension k is eliminated. This means that the four elements "i1.j1.k1", "i1.j1.k2", "i1.j1.k3" and "i1.j1.k4" of the set ijk are all mapped to the element "i1.j1" of the set ij1a. Note

that in this context, the result of the `sum` operation is *not* a number but a set. The second and third assignments are aggregations, where the number of elements of the two sets are computed. As already mentioned, the result of the first aggregation is 24 and the result of the second aggregation is 6 = 24 / 4.

The second way to perform projections and aggregations is faster and more compact, it uses an option statement. The general syntax of this option statement is as follows.

```
option ident1 < ident2 ;
option ident1 <= ident2 ;
```

The keyword `option` is followed by the identifiers `ident1` and `ident2` which are linked by the symbol `'<'` or `'<='`. Observe that in most cases the two symbols have the same effect. The exception is the special case when both identifiers are defined over domains that use at least one shared index set more than once, see the example below. Note that in general the dimension of the item on the left has to be equal or less than the dimension of the item on the right. Further, the index space of the two identifiers must be matchable. If the dimensions of the two identifiers are equal, then the same indices must appear in both, albeit the order may differ. If the dimension of the left item is less than the dimension of the right item, then the indices on the left must also appear on the right.

Observe that if both identifiers are sets, the operation will be a *projection*. However, if the identifier on the left-hand side is a scalar or a parameter and the identifier on the right-hand side is a set it will be an *aggregation*. The example that follows shows how the projection and the two aggregations above are accomplished with the option statement.

```
* Method 2: Option statement performs a projection
Option ij1b < ijk;

* Method 2: Option statements performs aggregations (counting of elements)
Option Count_2b < ijk;
Option Count_1b < ij1b;

display ijk, ij1a, ij1b, Count_1a, Count_1b, Count_2a, Count_2b;
```

In the example above, the set on the left-hand side, `ij1b`, has fewer indices than the set on the right-hand side, `ijk`. Observe that if the two sets differ only in the *order* of the indices then a projection will have the effect of a *permutation* of the tuple.

Note

The option statement for projection and aggregation operations may also be applied to parameters.

Until now the indices in the domain of the sets were unique. A special case arises when sets are defined over a domain with the same indices, for example the set `s(i,i,i)`. In this case, a projection always has the effect of a permutation. Users may choose whether they wish to perform the permutation from left to right or from right to left. The option statement

```
Option item1 < item2 ;
```

means a right-to-left permutation, while the option statement

```
Option item1 <= item2 ;
```

entails a left-to-right permutation. The following example clarifies the difference:

```
Set   i         / i1*i3 /
      s(i,i,i)  "Set members" / i1.i2.i3, i3.i3.i1/
      pR1(i,i)  "projection right to left with assignment"
      pR2(i,i)  "projection right to left with option statement"
      pL1(i,i)  "projection left to right with assignment"
      pL2(i,i)  "projection left to right with option statement" ;
Alias (i,ii,iii);

* Right-to-left permutation, two ways
pR1(i,ii) = sum(s(iii,ii,i),1);
option pR2 < s;

* Left-to-right permutation, two ways
pL1(i,ii) = sum(s(i,ii,iii),1);
option pL2 <= s;

option s:0:0:1, pR1:0:0:1, pR2:0:0:1, pL1:0:0:1, pL2:0:0:1;
display s, pR1, pR2, pL1, pL2 ;
```

Note that in the right-to-left permutation, the element `"i1.i2.i3"` is projected to `"i3.i2"` and the element `"i3.i3.i1"` is projected to `"i1.i3"`. In the left-to-right permutation however, the the element `"i1.i2.i3"` is projected to `"i1.i2"` and the element `"i3.i3.i1"` is projected to `"i3.i3"`. Hence, the left-to-right permutation (`<=`) might be more intuitive.

Our examples so far involved only sets. As mentioned above, projections and aggregations may also be performed with parameters. However, there are some subtle differences. The first difference refers to the terminology: we *aggregate* sets, but we *count* parameters. The second difference is the result of the operation if the domain of the left symbol is just a permuted version of the domain of the right symbol. Consider the following example:

```
Set i / i1*i3 /
    j / j1*j2 /;

Table p(i,j)
    j1 j2
i1   1  2
i2   3  4
i3   5  6;

parameter pperm(j,i);
option pperm < p;
option decimals = 0;
display p, pperm;
```

The output generated by the display statement follows:

```
----     13 PARAMETER p

            j1          j2

i1           1           2
i2           3           4
i3           5           6
```

```
----     13 PARAMETER pperm

            i1          i2          i3

j1           1           3           5
j2           2           4           6
```

Permutation of Sets and Parameters

In GAMS, the > sign can be used to create complete permutations of one- and multi-dimensional sets and parameters. Consider the following example based on the test library model [**PERM1**]:

First, a set `i` for which all permutations should be computed is declared and defined. A permutation of a one-dimensional set `i` can be represented by a two-dimensional set like `perm(i,i)` where the `i` index is duplicated. In the example, set `perm(i,i)` represents permutation `(1 2)(3)` in cycle notation. For a set with three elements, there are `3*2*1=6` permutations. The statement `option pall > i;` results in all permutations of `i` being computed and stored in three-dimensional set `pall(p,i,i)` where the first index `p` serves as a counter to enumerate all permutations.

```
set i           'set to permute'        / i1*i3 /
    perm(i,i)   'exemplary permutation' / i1.i2, i2.i1, i3.i3 /
$eval pmax fact(card(i))
    p           'permutation index' / p1*p%pmax% /
    pall(p,i,i) 'permutation set';
option pall > i;
option pall:0:0:1;
display pall;
```

Note that we use display control such that the display statement results in:

```
----      8 SET pall  permutation set

p1.i1.i1
p1.i2.i2
p1.i3.i3
p2.i1.i1
p2.i2.i3
p2.i3.i2
p3.i1.i2
p3.i2.i1
p3.i3.i3
p4.i1.i2
p4.i2.i3
p4.i3.i1
p5.i1.i3
p5.i2.i1
p5.i3.i2
p6.i1.i3
p6.i2.i2
p6.i3.i1
```

For multi-dimensional sets, the permutation operator follows the same logic. Consider the following example:

Sets `j`, `k` and a two-dimensional set `jk(j,k)` are defined. In order to find all permutations of `jk(j,k)`, the indices are again duplicated and a permutation index is introduced:

```
set j / j1*j2 /, k / k1*k5 /
    jk(j,k) / j1.k3, j1.k5, j2.k1 /
$eval pmax fact(card(jk))
    p / p1*p%pmax% /
    pall(p,j,k,j,k);
option pall > jk;
option pall:0:0:1;
display pall;
```

The display statement results in:

```
----      8 SET pall

p1.j1.k3.j1.k3
p1.j1.k5.j1.k5
p1.j2.k1.j2.k1
p2.j1.k3.j1.k3
p2.j1.k5.j2.k1
p2.j2.k1.j1.k5
p3.j1.k3.j1.k5
p3.j1.k5.j1.k3
p3.j2.k1.j2.k1
p4.j1.k3.j1.k5
p4.j1.k5.j2.k1
p4.j2.k1.j1.k3
p5.j1.k3.j2.k1
p5.j1.k5.j1.k3
p5.j2.k1.j1.k5
p6.j1.k3.j2.k1
p6.j1.k5.j1.k5
p6.j2.k1.j1.k3
```

In addition to set elements, it is also possible to permute numerical data in a GAMS parameter. Consider the following example where all permutations of the numerical values stored in the one-dimensional parameter `a(i)` are computed:

```
set i  / i1*i3 /
$eval pmax fact(card(i))
    p  / p1*p%pmax% /;
Parameter a(i) /i1 1, i2 2, i3 3/
          pall_a(p,i);
option pall_a > a;
option pall_a:0:1:1;
display pall_a;
```

The display statement results in:

```
----      8 PARAMETER pall_a

            i1          i2          i3

p1           1           2           3
p2           1           3           2
p3           2           1           3
p4           2           3           1
p5           3           1           2
p6           3           2           1
```

The permutation can also be extended to multi-dimensional parameters. Consider the following example where all permutations of the numerical values stored in the two-dimensional parameter `b(j,k)` are computed:

```
set j / j1*j2 /, k / k1*k5 /;
Parameter b(j,k) /j1.k3 1, j1.k5 2, j2.k1 3/;
$eval pmax fact(card(b))
set p / p1*p%pmax% /;
parameter pall_b(p,j,k);
option pall_b > b;
option pall_b:0:1:2;
display pall_b;
```

The display statement results in:

```
----      8 PARAMETER pall_b

         j1.k3       j1.k5       j2.k1

p1           1           2           3
p2           1           3           2
p3           2           1           3
p4           2           3           1
p5           3           1           2
p6           3           2           1
```

4.35 System Attributes

4.35.1 Introduction

System attributes give access to string constants in the GAMS system environment. We will refer to these string constants as system suffixes. Moreover, there are attributes that can be used as data elements for user symbols. This way one can get access e.g. to the solvers (as set elements) in the system. We will refer to these data elements as system data. Two special system data attributes, namely `powerSetLeft` and `powerSetRight`, for a three dimensional set `setName(n,s,b)` create data which can be interpreted as a numbering system with base b and s digits. Finally, a system attribute can provide access to a system function that was hidden by the compiler because of a name conflict with a user symbol.

4.35.2 System Suffixes

System suffixes contain information about the GAMS system environment during a run. There are two ways to reference them: `system.suffix` and `%system.suffix%`. Here `system` is a keyword and `.suffix` is the name of the system suffix. A full list is given in section List of all System Suffixes below. Note that `system.suffix` references the *execution-time* version of the system suffix and `%system.suffix%` references the *compile-time* version resulting in a textual replacement. The execution-time system suffix can only be used in the context of put files. Consider the following example where both versions are used:

```
file fx;
put fx 'lp:' system.LP;
option lp=gurobi;
put  / 'lp:' system.LP
put  / 'lp:%system.LP%';
```

Observe that the compile-time string `%system.LP%` is evaluated at compile time and does not change. However, the execution-time system suffix `system.LP` is evaluated at execution time and it does change. It is first `CPLEX`, since CPLEX is the default LP solver. Then it changes to `GUROBI` as a result of the execution-time command `option LP=gurobi;`. The resulting `put` file `fx.put` follows:

```
lp:CPLEX
lp:GUROBI
lp:CPLEX
```

In the following two subsection we will discuss execution-time suffix in more detail.

4.35.2.1 Execution-Time System Suffixes

Execution-time system suffixes are only used in the context of the put writing facility with commands like put , putclose and put_utility. Consider the following simple example. We have the files `x.gms` and `dummy.gms`, where `x.gms` contains just the following line:

```
file fx; put fx system.version;
```

The file `dummy.gms` contains the following:

```
$exit;
```

We run these files with the following calls:

```
> c:\gams\win64\24.7\gams x.gms action=c s=putVersion
```

Note that the value `c` for the command-line parameter action causes the first input file to be compiled only. The result is saved in the work file `putVersion`. Then we restart and execute `dummy.gms`. Now, depending on the GAMS release version, the `put` file `fx` will have a different content:

```
c:\gams\win64\24.7\gams dummy.gms r=putVersion // this produces a put file with content "GAMS Rev 247
c:\gams\win64\24.8\gams dummy.gms r=putVersion // this produces a put file with content "GAMS Rev 248
```

In contrast, the value of the compile-time system suffix `%system.version%` is determined at compile time. Suppose we change the file `x.gms` to contain the following line:

```
file fx; put fx "%system.version%";
```

The file `dummy.gms` is not changed and we use the same calls as above. Note that the `put` file will have the same content for both versions of GAMS:

```
c:\gams\win64\24.7\gams dummy.gms r=putVersion // this produces a put file with content "GAMS Rev 247
c:\gams\win64\24.8\gams dummy.gms r=putVersion // this produces a put file with content "GAMS Rev 247
```

4.35.2.2 Compile-Time System Suffixes

Compile-time system suffixes may be placed anywhere a compile-time variables makes sense, including code for conditional compilation. The quoting of these compile-time system suffixes depends on their use in the code. Consider the following example:

```
$set systemDATE %system.date%
display "system.date", "%system.date%";
$ifi %system.LP% == Xpress $log "LP solver is Xpress."
```

The resulting output generated by the display statement follows:

```
----      3 system.DATE
            11/07/16
```

4.35.2.3 List of all System Suffixes

In the following table all system suffixes are listed.

System Suffix	Description
BuildCode	System build code
CNS	CNS solver that is active for CNS model type
ComputerName	Operating system computer name
Date	Job date
Date1	Job date format
DirSep	File or directory separator in file names
DNLP	DNLP solver that is active for DNLP model type
Elapsed	Elasped time in seconds since start of job
EMP	EMP solver that is active for EMP model type
Error	Used to indicate unknown suffix
ErrorLevel	System Errorlevel
FE	File extension
FileSys	Name of the operating system (MSNT for Windows, UNIX for Linux and macOS)
FN	File name
FP	File path
GamsRelease	GAMS Release number
GamsReleaseMaint	GAMS Release number with maintenance number suffix
GamsVersion	GAMS version number
GdxFileNameIn	GDX file name for input, set by put_utility command gdxin
GdxFileNameOut	GDX file name for output, set by put_utility command gdxout
GString	GAMS system audit string (exact GAMS version being used)
HostPlatform	Host platform
IFile	Input file
ILine	Current source line number beeing executed
IncLine	Include file line
IncName	Include file name
IncParent	Include file parent
IncParentL	Include file parent line number

System Suffix	Description
IsAlphaBeta	Indicates an Alpha or Beta bulid
JobHandle	Job handle of last async call
LicenseFileName	The file name of the license file currently used
LicenseLevel	GAMS license level
Line	Line number in source code
ListLine	Line number on listing file
LP	LP solver that is active for LP model type
MACAddress	MAC address of the first network adapter
MaxInput	Max input line length that can be processed
MCP	MCP solver that is active for MCP model type
Memory	Memory (in Mb) in use
MINLP	MINLP solver that is active for MINLP model type
MIP	MIP solver that is active for MIP model type
MIQCP	MIQCP solver that is active for MIQCP model type
MPEC	MPEC solver that is active for MPEC model type
NLP	NLP solver that is active for NLP model type
NullFile	The null filename
OFile	Output (Listing) file
OPage	Current page number in output (listing) file
Page	Current page number
PFile	Current put file
Platform	Job platform (DAX for macOS on ARM64, DEX for macOS on x86_64, LEX for Linux, WEX for Windows)
PrLine	Line on listing page
Procname	Model types LP MIP etc
PrPage	Listing page number
PutFileName	The filename of the currently active PUT file
QCP	QCP solver that is active for QCP model type
RDate	Restart file date
ReDirLog	Append redirection string into the logfile
RFile	Restart file name
RMINLP	RMINLP solver that is active for RMINLP model type
RMIP	RMIP solver that is active for RMIP model type
RMIQCP	RMIQCP solver that is active for RMIQCP model type
RMPEC	RMPEC solver that is active for RMPEC model type
RTime	Restart file time
SFile	Save file name
SString	Subsystem (Solver) audit (last solver used)
Tab	Tab character
TClose	Time to save GAMS
TComp	Time to compile
TExec	Time to execute
Time	Job time
Title	Current listing title
TStart	Time to restart GAMS
UserConfigDir	User writable directory that is searched for gamsconfig.yaml
UserDataDir	User writable directory that is searched for gamslice.txt and others
UserName	Operating system user name

System Suffix	Description
VerID	GAMS version ID
Version	GAMS compiler version

A model that prints all system suffixes with its current values to a put file can be found in model [**SSUFFIX**].

4.35.3 System Data

Some of the system relevant information does not fit into a single string. Such system data can be stored in GAMS symbols. The system data for this symbol can be accessed in the data statement of the symbol, e.g. `set allSolvers / system.solverNames /;`. Now you can work with the solver names as you can work with any set. The system data can be viewed as an internal set, so it can be used to create more complex GAMS symbols with this information. For example:

```
set seq / 1*1000 /;
set solvermap(seq,*)   / set.seq:system.solverNames /;
parameter solverpar(*) / system.solverNames 1 /;
```

Here is the list and dimensionality of the system data:

System Data	Dimension	Description
dollarOptions	1	Dollar control options
empty	1	Empty label
gamsFunctions	1	Intrinsic functions
gamsParameters	1	Command line parameters
gamsParameterSynonymMap	2	Map between command line parameters and their synonyms
gamsParameterSynonyms	1	Synonyms for command line parameters
GUSSModelAttributes	1	Model attributes stored for GUSS scenarios
modelTypes	1	Model types, e.g. LP, MIP, ...
platforms	1	Platform code
powerSetLeft	3	Numbering system with base b and s digits
powerSetRight	3	Numbering system with base b and s digits
predefinedSymbols	1	Predefined symbols, e.g. SameAs, Diag, ...
setConstants	1	System data names (this list)
solverNames	1	Names of solvers and tools
solverPlatformMap	2	Map between solvers and platforms
solverTypePlatformMap	3	Map between solvers, model types and platforms
systemSuffixes	1	System suffixes

4.35.3.1 Power Set

The very special system attribute `powerSetLeft` and `powerSetRight` do not provide access to static data relevant to the system but produce data based on existing sets `b` and `s`. This system data produced can be interpreted as a numbering system with base b and s digits. The total number of tuples created is

power(card(b),card(s)). The first index of the resulting set has to have at least this size. The following small example demonstrates the functionality of powerSetRight:

```
$set digits 3
set s / s1*s%digits% /, b / 0,1 /;
$eval nMax power(card(b),card(s))
set n        / n1*n%nMax% /
    x(n,s,b) / system.powerSetRight /;
option x:0:0:%digits%; display x;
```

With 3 digits the display results in the following output:

```
----      6 SET x
n1.s1.0,    n1.s2.0,    n1.s3.0
n2.s1.0,    n2.s2.0,    n2.s3.1
n3.s1.0,    n3.s2.1,    n3.s3.0
n4.s1.0,    n4.s2.1,    n4.s3.1
n5.s1.1,    n5.s2.0,    n5.s3.0
n6.s1.1,    n6.s2.0,    n6.s3.1
n7.s1.1,    n7.s2.1,    n7.s3.0
n8.s1.1,    n8.s2.1,    n8.s3.1
```

Changing powerSetRight to powerSetLeft reassigns the order of the digits. So the lowest digit is on the left:

```
----      6 SET x
n1.s1.0,    n1.s2.0,    n1.s3.0
n2.s1.1,    n2.s2.0,    n2.s3.0
n3.s1.0,    n3.s2.1,    n3.s3.0
n4.s1.1,    n4.s2.1,    n4.s3.0
n5.s1.0,    n5.s2.0,    n5.s3.1
n6.s1.1,    n6.s2.0,    n6.s3.1
n7.s1.0,    n7.s2.1,    n7.s3.1
n8.s1.1,    n8.s2.1,    n8.s3.1
```

4.35.4 Access to Hidden Functions

Intrinsic functions are not keywords but they are stored in a list of predefined names. If a user program uses such a predefined name of a function for a user symbol, GAMS will hide the original function. For example,

```
set uniform 'School Uniform' / skirt, blouse, blazer, socks, shoes /;
scalar randomNumber;
randomNumber = uniform(0,1);
```

will result in a compilation error in the second line because GAMS expects uniform to be a set. The function uniform is hidden for this GAMS program. Renaming the user symbol that hides a function is often the best solution for this, but there are a few cases (including GAMS code already compiled and stored in a restart file) where renaming is not an option. A hidden function can be accessed via the system attribute system.functionName. So the following will successfully compile and execute:

```
set uniform 'School Uniform' / skirt, blouse, blazer, socks, shoes /;
scalar randomNumber;
randomNumber = system.uniform(0,1);
```

4.36 The Grid and Multi-Threading Solve Facility

4.36.1 Introduction

The GAMS Grid facility allows to take advantage of High Performance Computing Grids and systems with multiple CPUs. This language feature facilitates the management of asynchronous submission and collection of model solution tasks in a platform independent fashion. A simple architecture, relying on existing operating system functionality allows for rapid introduction of new environments and provides for an open research architecture.

A typical application uses a coarse grain approach involving hundreds or thousands of model solutions tasks which can be carried out in parallel. Examples include but are not limited to scenario analysis, Monte Carlo simulations, Lagrangian relaxation, decomposition algorithms and advanced solution approaches.

The grid features work on all GAMS platforms and have been tailored to many different environments, like the `Condor` Resource Manager, a system for high throughput computing from the University of Wisconsin-Madison. Researchers using Condor reported a delivery of 5000 CPU hours in 20 hours wall clock time.

Similarly, the GAMS Multi-Threading Solve Facility allows the asynchronous submission and collection of model solution tasks on a single, multi-threaded machine while using efficient in-memory communication between GAMS and the solver.

Disclaimer: The use of the term grid computing may be offensive to some purists in the computer science world. We use it very loosely to refer to a collection of computing components that allow us to provide high throughput to certain applications. One may also think of it as a resurrection of the commercial service bureau concept of some 30 years ago.

Caution: Although these features have been tested on all platforms and are part of our standard release we may change the approach and introduce alternative mechanisms in the future.

Acknowledgments: Prof. Monique Guignard-Spielberg and Antoine Sauré at that time at Wharton School at the University of Pennsylvania introduced us to parallel Lagrangian relaxation on the SUN Grid Environment. Prof. Michael Ferris from the University of Wisconsin-Madison adopted our original GAMS grid approach to the high throughput system Condor and helped to make this approach a practical proposition.

4.36.2 The Grid Facility: Basic Concepts

The Grid facility separates the solution process into several steps which then can be controlled separately. First we will review the steps taken during synchronous solution and then we will introduce the asynchronous or parallel solution steps.

When GAMS encounters a solve statement during execution it proceeds in three basic steps:

1. **Generation:** The symbolic equations of the model are used to instantiate the model using the current state of the GAMS data base. This instance contains all information and services needed by a solution method to attempt a solution. This representation is independent of the solver and computing platform.
2. **Solution:** The model instance is handed over to a solver and GAMS will wait until it terminates.
3. **Update:** The detailed solution and statistics are passed to GAMS from the solver to update the GAMS data base.

In most cases, the time taken to generate the model and update the data base with the solution will be much smaller than the actual time spent in a specific solver. The model generation will take a few seconds, whereas the time to obtain an optimal solution may take a few minutes to several hours or even longer. If sequential model solutions do not depend on each other, we can solve in parallel and update the data base in a random order. All we need is a facility to generate models, submit them for solution and continue. At a convenient point in our GAMS program we will then look for the completed solutions and update the data base accordingly. To summarize, solving in parallel entails two steps:

1. **Submission Loop:** In this phase we will generate and submit models for solutions that can be solved independently.
2. **Collection Loop:** The solutions of the previously submitted models are collected as soon as a solution is available. It may be necessary to wait for some solutions to complete by pausing the execution for some time.

Note that we have assumed that there will be no errors in any of these steps. Of course, this will not always be the case and elaborate mechanisms are in place to make the operation fail-safe.

Note

For scenario analysis the solver GUSS might be of particular interest. The model **[GUSSGRID]** demonstrates how GUSS is used together with the Grid facility.

4.36.3 The Grid Facility: A First Example

In this section we will illustrate the use of the basic grid facility with the model **[QMEANVAR]**. This model traces an efficiency frontier for restructuring an investment portfolio. Each point on the frontier requires the solution of independent quadratic mixed integer models. The *original solution loop* is shown below:

```
loop(p(pp),
   ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
   solve minvar min var using miqcp ;
   xres(i,p)         = x.l(i);
   report(p,i,'inc') = xi.l(i);
   report(p,i,'dec') = xd.l(i);
);
```

This loop will save the solutions of the model `minvar` for different returns `ret`. As the solutions do not depend on the order in which they are carried out, we can rewrite this loop to operate in parallel.

4.36.3.1 The Submission Loop

The first step for solving in parallel using the Grid facility is to write the submission loop:

```
Parameter h(pp)      'model handles';
minvar.solveLink = 3;
loop(p(pp),
   ret.fx = rmin + (rmax-rmin)/(card(pp)+1)*ord(pp) ;
   solve minvar min var using miqcp;
   h(pp) = minvar.handle;
);
```

The model attribute solveLink controls the behavior of the solve statement. The value of 3 (which is equivalent to the compile-time constant %solveLink.asyncGrid%) directs GAMS to generate the model and submit it for solution and then continue without waiting for the completion of the solution step. Thus with setting `minvar.solveLink` to 3 we activate grid computing.

A *handle* in the grid environment identifies the particular model and data instances available. The model attribute handle contains a unique identification of each submitted solution request and is typically stored in a parameter defined over a set that covers all model instances. The specific numerical values of handles are assigned by GAMS and may be used to recover solutions and manage models that are solved on the grid. In our example, the handle parameter is `h` and the set of all model instances is `pp`. The handle values that are stored in `h` are later used to collect the solutions once the solution processes are completed.

4.36.3.2 The Collection Loop

We collect the solutions with the following collection loop:

```
loop(pp$handleCollect(h(pp)),
   xres(i,pp)         = x.l(i);
   report(pp,i,'inc') = xi.l(i);
   report(pp,i,'dec') = xd.l(i);
);
```

Note that the dollar condition restricts the looping set `pp` to those elements which return a nonzero value to the function `handleCollect(h(pp))`. The function handleCollect tests the solution status for each element of the set `pp`. And if the solution is available, it is read into the GAMS data base. In this case the function returns a value of 1. If the solution is not ready to be retrieved, the value zero will be returned.

Observe that the collection loop above has one big flaw. If a solution has not been ready (that is if `handleCollect` equaled zero), it will not be retrieved. We need to call this loop several times until all solutions have been retrieved or we get tired of it and quit. We will use a repeat until construct and the handle parameter `h` to control the loop to look only for the solutions that have not been loaded yet. The code follows:

```
repeat
   loop(pp$handleCollect(h(pp)),
         xres(i,pp)         = x.l(i);
         report(pp,i,'inc') = xi.l(i);
         report(pp,i,'dec') = xd.l(i);
         display$handleDelete(h(pp)) 'trouble deleting handles' ;
         h(pp) = 0;
   ) ;
   display$readyCollect(h, 100) 'Problem waiting for next instance to complete';
until card(h) = 0 or timeelapsed > 100;
xres(i,pp)$h(pp) = na;
```

Once we have extracted a solution we will set the handle parameter `h` to zero. In addition, we want to remove the instance from the system by calling the function handleDelete which returns zero if successful. No harm is done if it fails but we want to be notified via the conditional display statement. Before running the collection loop again, we may want to wait a while to give the system time to complete more solution steps. This is done with the function readyCollect which waits until another model instance with a handle in `h` is ready to be collected (or the optionally defined number of seconds has passed). The final wrinkle is to terminate if all model instances have been deleted from the system since their solutions were retrieved or after 100 seconds have elapsed, even if we did not get all solutions. This is accomplished with the function timeElapsed and is important, because if one of the solution steps fails our program would never

terminate. Recall that the handle parameter `h` equals zero for all elements of the set `pp` whose related models have been solved and their solutions have been extracted. The last statement in the code above sets the results of the *missed* solves to `na` to signal the failed solve. The parameter `h` will now contain the handles of the failed solves for later analysis.

Alternatively, we could use the function handleStatus and collect the solutions that are stored in a GDX file. For example, we could write:

```
loop(pp$(handleStatus(h(pp)) = 2),
   minvar.handle = h(pp);
   execute_loadhandle minvar;
   xres(i,pp)         = x.l(i);
   report(pp,i,'inc') = xi.l(i);
   report(pp,i,'dec') = xd.l(i);
);
```

The function handleStatus tests the solution process and returns the value 2 if the solution process has been completed and the results can be retrieved. The solution is stored in a GDX file which can be loaded in a way similar to other GDX solution points. First, we need to specify which solution to retrieve by setting the the model attribute `minvar.handle` to the appropriate value. Then we can use the statement `execute_loadhandle minvar;` to load the solution for the model `minvar` back into the GAMS data base.

Note

Except for the requirement of a model with a previously specified handle, the command `execute_loadhandle` operates like the procedure execute_loadpoint.

Using the function `handleStatus` and the command `execute_loadhandle` instead of the simpler `handleCollect`, adds one more layer of control to the final collection loop. Now we need one additional if statement inside the collection loop above:

```
repeat
   loop(pp$h(pp),
      if(handleStatus(h(pp)) = 2,
         minvar.handle = h(pp);
         execute_loadhandle minvar;
         xres(i,pp)         = x.l(i);
         report(pp,i,'inc') = xi.l(i);
         report(pp,i,'dec') = xd.l(i);
         display$handleDelete(h(pp)) 'trouble deleting handles' ;
         h(pp) = 0;
      );
   ) ;
   display$readyCollect(h, 100) 'Problem waiting for next instance to complete';
until card(h) = 0 or timeelapsed > 100;
xres(i,pp)$h(pp) = na;
```

Finally, we are ready to run the modified model.

4.36.3.3 The Execution Log

The execution log will contain some new information that may be useful for more advanced applications:

```
--- LOOPS pp = p1
---   46 rows 37 columns 119 non-zeroes
---   311 nl-code 7 nl-non-zeroes
---   14 discrete-columns
--- Submitting model minvar with handle grid137000002
--- Executing after solve
...
--- GDXin=C:\answerv5\gams_srcdev\225j\grid137000003\gmsgrid.gdx
--- Removing handle grid137000003
```

Note that the log contains some additional information about the submission, retrieval and removal of the solution instance. In the following sections we will make use of this additional information.

The execution log does not cotain any solver log. Though, if a logOption value is set, that triggers writing to a file (i.e. `2` or `4`), the solver log of each job will be written to the corresponding folder in the Grid Directory. So, if the solver log should be inspected when the job is done, one needs to make sure, that these folders do not get cleaned automatically, e.g. by setting the gridDir command line parameter.

For a complete example for grid computing, see the grid enabled transport model [**TRNSGRID**].

Observe that we have made no assumptions about what kind of solvers and what kind of computing environment we will operate. The example above is completely platform and solver independent and it runs on a Windows laptop or on a massive grid network like the Condor system without any changes in the GAMS source code.

4.36.4 Advanced Use of Grid Features

In this section we will describe a few special application requirements and show how this can be handled with the current system. Some of those applications may involve thousands of model instances with solution times of many hours each. Some may fail and require resubmission. More complex examples require communication and the use of GAMS facilities like the Branch-and-Cut-and-Heuristic Facility (BCH), which submit other models from within a running solver.

Imagine a situation with thousands of model instances each taking between minutes and many hours to solve. We will break the master program into a *submitting* program, an *inquire* program and a final *collection* program. We will again use the model [**QMEANVAR**] to demonstrate the principle. We will split the code of the modified [**QMEANVAR**] GAMS code into three components: `qsubmit`, `qcheck` and `qreport`.

4.36.4.1 Very Long Job Durations: The Submitting Program

The file `qsubmit.gms` will include everything up to and including the new submission loop. To save the instances we will need a unique grid directory `gdir` and to restart the problem we will have to create a `save file`. For details on the save and restart facility in GAMS, see chapter The Save and Restart Feature. When running the first job, we will use the command line parameter save or its synonym `s` to create the required `save file`:

```
> gams qsubmit s=submit gdir=c:\test\grid
```

4.36.4.2 Very Long Job Durations: The Inquire Program

The solution of all the model instances may take hours. From time to time we may run a quick inquiry job to learn about the status. The following program `qcheck.gms` will list the current status:

```
Parameter status(pp,*);
Scalar    handle;
Acronym   BadHandle, Waiting, Ready;
loop(pp,
   handle = handleStatus(h(pp));
   if(handle=0,
      handle = BadHandle;
   elseif handle=2,
      handle = Ready;
      minvar.handle = h(pp);
      execute_loadhandle minvar;
      status(pp,'solvestat') = minvar.solvestat;
      status(pp,'modelstat') = minvar.modelstat;
      status(pp,'seconds') = minvar.resusd;
   else
      handle = Waiting;
   );
   status(pp,'status') = handle;
);
display status;
```

For details on the model attributes referenced in the code above, see handle, solveStat, modelStat and resUsd. To run the program above, we will restart from the previous `save file` by using the command line parameter restart or its synonym `r`.

```
> gams qcheck r=submit gdir=c:\test\grid
```

The output generated by the display statement may look like the following:

```
----   173 PARAMETER status

     solvestat   modelstat   seconds     status

p1       1.000       1.000     0.328      Ready
p2       1.000       1.000     0.171      Ready
p3                                      Waiting
p4                                      Waiting
p5       1.000       1.000     0.046      Ready
```

We may want to do some more detailed analysis on one of the solved model instances. The respective program, called `qanalyze.gms`, may inlcude the following lines of code:

```
$if not set instance $abort --instance is missing
if(not handleStatus(h('%instance%')),
     abort$yes 'model instance %instance% not ready');
minvar.handle = h('%instance%');
execute_loadhandle minvar;
display x.l,xi.l,xd.l;
...
```

For information on dollar control options, see chapter Dollar Control Options, especially the detailed descriptions of the options $if and $abort. Note that `instance` is a compile-time variable. The program may be called using a double dash paramter, which defines and sets a GAMS compile-time variable:

```
> gams qanalyze r=submit gdir=c:\test\grid --instance=p4
```

4.36.4.3 Very Long Job Durations: The Collection Program

Once all jobs are completed we are ready for the collection loop. For simplicity, we will not include the repeat loop, because we would not run the final collection program unless we were satisfied that we got most of the solutions that we wanted. The file `qreport.gms` could look like the following:

```
loop(pp$handleStatus(h(pp)),
   minvar.handle = h(pp);
   execute_loadhandle minvar;
   xres(i,pp)         = x.l(i);
   report(pp,i,'inc') = xi.l(i);
   report(pp,i,'dec') = xd.l(i);
   display$handleDelete(h(pp)) 'trouble deleting handles' ;
   h(pp) = 0;
);
xres(i,pp)$h(pp) = na;
...
```

We will restart the program above from the `save file` that was created earlier:

```
> gams qreport r=submit gdir=c:\test\grid
```

Note that it is not necessary to run the job from the same directory that we have used for the initial submission; it is even possible to use a different operating system.

4.36.5 Summary of Grid Features

We introduced several GAMS features to facilitate the asynchronous or parallel execution of the solve statement. These GAMS features are summarized in the following subsections.

In addition to the features described below, the option or command line parameter ThreadsAsync was introduced. `ThreadsAsync` controls the number of threads or CPU cores that are used in multi-threading computing.

4.36.5.1 Grid Handle Functions

The grid handle functions are listed in Table 1. For details on functions in GAMS in general and complete lists of all GAMS functions, see section Functions. Note that the desired return values - the return values that indicate that no error has occurred - are marked with bold letters.

Function	Description	Return Values
handleCollect(HANDLE)	Tests if the solve of the model instance identified by HANDLE is done: if so, it loads the solution into the GAMS data base. If the option asyncSolLst is active the solution listing is printed to the listing file. **Note** that handleCollect ignores the setting of the option SolveOpt and always uses the default value merge.	0: The model instance was not ready or could not be loaded. **>0: The model instance solution has been loaded**. (When using the Grid facility, this is always 1; when using the multi-threading option, this returns the thread ID used.)
handleStatus(HANDLE)	Tests if the solve of the model instance identified by HANDLE is done. Note that there are compile-time constants that are related to this function.	0: The model instance is not known to the system. 1: The model instance exists but the solution process is incomplete. **2: The solution process has terminated and the solution is ready for retrieval.** 3: The solution process signaled completion but the solution cannot be retrieved.
handleDelete(HANDLE)	Deletes the model instance identified by HANDLE and returns a numerical indicator of the status of the deletion. If the HANDLE given is not valid, an execution error is triggered.	**0: The model instance has been removed.** 1: The argument HANDLE is not a legal handle. 2: The model instance is not known to the system. 3: The deletion of the model instance encountered errors.
handleSubmit(HANDLE)	Resubmits the model instance identified by HANDLE for solution. In case of a nonzero return an execution error is triggered.	**0: The model instance has been resubmitted for solution.** 1: The argument HANDLE is not a legal handle. 2: The model instance is not known to the system. 3: The completion signal could not be removed. 4: The resubmit procedure could not be found. 5: The resubmit process could not be started.

Function	Description	Return Values
`readyCollect(HANDLES[,maxWait])`	Waits until a model solution is ready to be collected. `HANDLES` must be either a scalar or parameter containing one or more model handles or a model with its handle attribute. `MaxWait` specifies the maximum time to wait in seconds, the default value is `+inf`.	**0: One or more of the requested jobs is/are ready.** 1: There is neither an active job to wait for nor a job ready to be collected. 2: The handle symbol is empty. 3: The argument is not a legal handle. 4: User specified time-out (using a solveLink = 6 handle). 5: User specified time-out (using a solveLink = 3 handle). 8: Unknown error (should not happen).

Table 1: Grid Handle Functions

Note that GAMS might issue execution errors which could give additional information that may help to identify the source of problems. The function execError may be used to get and set the number of execution errors.

4.36.5.2 Grid Model Attributes

Model attributes are introduced in section Model Attributes. The following three model attributes are particularly relevant for grid computing:

Function	Description
solveLink	Specifies the solver linking conventions. The following values direct the solve statement to use grid computing or multi-thread computing: 3, 4, 6 and 7. Note that the default for this model attribute can be set as command line parameter and option statement. For more information, see the detailed description.
handle	Specifies the current instance handle. This is used to identify a specific model instance and to provide additional information needed for the process signal management (compare subsections The Submission Loop and The Collection Loop above).
number	Specifies the current instance number. Any time a solve is attempted for a model, the instance number is incremented by one and the handle is update accordingly. The instance number can be reset by the user which then resynchronizes the handle.

Table 2: Grid Handle Attributes

4.36.5.3 Grid Solution Retrieval

As an alternative to the function handleCollect a solution may be retrieved with the following statement:

```
execute_loadhandle mymodel;
```

This statement will update the GAMS data base with the status and solution for the current instance of `mymodel`. Note that the underlying mechanism is a GDX file. Except for the requirement of a model with a previously specified handle, this command operates like the procedure execute_loadpoint. If the option asyncSolLst is active the solution listing is printed to the listing file.

4.36.5.4 The Grid Directory

The instantiated (generated) models and their corresponding solutions are kept in unique directories that may be reached from the submitting system. Each GAMS job may have only one *grid directory*. By default, the grid directory is assumed to be the scratch directory. This may be overwritten by using the GAMS command line parameter GridDir, or short `GDir`. An example follows.

```
> gams myprogram ... GDir=gridpath
```

If `gridpath` is not a fully qualified name, the name will be completed using the current directory. If the grid path does not exist, an error will be issued and the GAMS job will be terminated. A related GAMS parameter is ScrDir or short `SD`.

Recall the following default mechanism: When a GAMS job is started a unique process directory is created in the current directory. These directories are named `225a` to `225zz`. When a GAMS job terminates, the system will remove the process directory at the completion of a GAMS job. Any file that has not been created by the GAMS core system will be flagged. If the call `gamskeep` instead of `gams` is used, another exit script will be activated that results in the process directory to be kept.

Note that if we do not specify a scratch directory, the scratch directory will be the same as the process directory. If we do not specify a grid directory, the grid directory will be the same as the scratch directory.

Observe that if we assume that some of the model instances may fail or we want to break the GAMS program into several pieces to run as separate jobs, we need to be careful not to remove the model instance we have not completely processed. In such cases we have to use the parameter GridDir, so that we may access previously created model instances.

4.36.6 The Grid Facility: Architecture and Customization

The current Grid facility relies on very basic operating system features and does not attempt to offer real and direct job or process control. The file system is used to signal the completion of a submitted task and GAMS has currently no other way to interact with the submitted process directly, like forcing termination or change the priority of a submitted task. This approach has its obvious advantages and disadvantages. There are a number of attempts to use grid computing to provide value added commercial remote computing services.

When GAMS executes a solve with the option solveLink set to 3 it will perform the following steps:

1. Create a subdirectory in the GridDir with the name `gridnnn`. Here `nnn` stands for the numeric value of the handle. The handle value is the internal symbol ID number x 1e6 + the model instance number. For example, in the **[QMEANVAR]** example the first grid subdirectory was `grid137000002`.
2. Remove the completion signal in case the file already exists. Currently the signal is a file called `finished`. For example, `grid137000002/finished`.
3. Create or replace a GDX file called `gmsgrid.gdx` which will contain a dummy solution with failed model and solver status. This file will be overwritten by the final step of the solution process and will be read when calling execute_loadhandle.
4. Place all standard GAMS solver interface files into the above instance directory.
5. Execute the submission wrapper called `gmsgrid.cmd` under Windows or `gmsgrid.run` under Unix. These submission scripts are usually located in the GAMS system directory, they may be located via the current path if they are not found in the GAMS system directory.

The grid submission script `gmsgrid.cmd` or `gmsgrid.run` is called with four arguments that are needed to make a standard GAMS solver call: the solver executable file name, the solver control file name, the solver scratch directory, and the solver name. The submission script then does the final submission to the operating system. This final script will perform the following steps:

1. call the solver,
2. call a utility that will create the final GDX file `gmsgrid.gdx`,
3. set the completion signal `finished`.

If we want to use the function handleSubmit we will also have to create the script `gmsrerun.cmd` or `gmsrerun.run`. This script could later be used to resubmit the job.

For example, the default submission script for Windows is shown below:

```
@echo off

: gams grid submission script
:
: arg1 solver executable
:    2 control file
:    3 scratch directory
:    4 solver name
:
: gmscr_nx.exe processes the solution and produces 'gmsgrid.gdx'
:
: note: %3 will be the short name, this is neeeded because
:       the START command cannot handle spaces or "...'
:       before we use %~3  will strip surrounding "..."
:       makes the name short
:
: gmsrerun.cmd will resubmit runit.cmd

echo @echo off                       > %3runit.cmd
echo %1 %2 %4                       >> %3runit.cmd
echo gmscr_nx.exe %2                >> %3runit.cmd
echo echo OK ^> %3finished ^& exit >> %3runit.cmd

echo @start /b /belownormal %3runit.cmd ^> nul  > %3gmsrerun.cmd

start /b /belownormal %3runit.cmd > nul

exit
```

4.36.6.1 Grid Submission Testing

The grid submission process can be tested on any GAMS program without having to change the source text. The option `solveLink=4` instructs the solve statement to use the grid submission process and then wait until the results are available. Note that the option solveLink may be set via a GAMS command line parameter, a GAMS option statement or via assignment to the model attribute. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the reslim seconds allocated for this optimization job and report a failure if this limit has been exceeded. After successful or failed retrieval of the solution, GAMS will remove the grid directory, unless we have used the call `gamskeep` or have set the GAMS command line parameter keep.

4.36.7 Multi-Threading

As we have described in this chapter, each solve is handled in its own process space with the Grid facility. Recall that the Grid facility is activated by setting the option or model attribute solveLink to 3 or 4. If `solveLink` is set to 6 (or the compile-time constant `%solveLink.asyncThreads%`) instead, a separate thread is used. This allows efficient in-memory communication between GAMS and the solver, like it is done if the option `solveLink` is set to 5 (or the compile-time constant `%solveLink.loadLibrary%`).

Apart from this, the multi-threading facility works in the same way as the Grid facility. The solve statement generates the model and passes it to the solver in a separate thread, then a handle of the model instance may be stored using the model attribute handle and the grid handle functions may be used to collect the solution and deal with the model instance, namely handleCollect, handleDelete, handleStatus and readyCollect.

Note that the option or command line parameter ThreadsAsync sets the maximum number of threads that should be used for the asynchronous solves.

The following matrix shows which solvers may be used with `solveLink = 6` on which platform:

Solver	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
CONOPT3	×	×	×	×
CONOPT4	×	×	×	×
CONVERT	×	×	×	×
COPT	×	×	×	×
CPLEX	×	×	×	×
GUROBI	×	×	×	×
HIGHS		×	×	×
IPOPT	×	×	×	×
MOSEK	×	×	×	×
ODHCPLEX	×	×		
SCIP	×	×	×	×
SHOT	×	×	×	×
SNOPT		×		
Soplex	×	×	×	×
XPRESS	×	×	×	

If a solver is selected for which `solveLink = 6` is not supported on the corresponding platform, `solveLink = 3` will be used instead and it will be noted in the log.

4.36.7.1 Multi-threading Submission Testing

The multi-threading submission process may be tested on any GAMS program without having to change the source text. The option solveLink = 7 (or, equivalently, the compile-time constant `%solveLink.threadsSimulate%`) instructs the solve statement to use the multi-threading submission process, wait until the results are available and then load the solution into the GAMS data base. Once the model instance has been submitted for solution, GAMS will check if the job has been completed. It will keep checking twice the reslim seconds allocated for this optimization job and report a failure if this limit has been exceed. After successful or failed retrieval of the solution GAMS will remove the thread handle.

4.37 Special Features for Mathematical Programs

4.37.1 Introduction

In this chapter we will introduce special GAMS features that are useful for specific model types. The features include model scaling, conic programming and features that facilitate mixed integer as well as indicator constraints, a feature that does not translate across solvers.

4.37.2 Special Mixed Integer Programming (MIP) Features

Some special features have been added to GAMS to help simplifying the modeling of Mixed Integer Programming (MIP) problems. In GAMS MIP is the model type for mixed integer *linear* programs, this section used MIP more generally, we consider model with discrete variables, including non-linear expressions and pure discrete problem. We will first present details on discrete variables in GAMS, then we will discuss how to customize priorities for the branching process. Next, we will cover the model attributes that are important for MIPs and we will conclude with some hints that will make mixed integer programming with GAMS easier.

4.37.2.1 Types of Discrete Variables

Variables and variable types are introduced in chapter Variables. GAMS provides six discrete variable types: `binary`, `integer`, `sos1`, `sos2`, `semicont` and `semiint`. In the following subsections we will present details and examples for each of these discrete variable types. Note that if any discrete variables feature in a model, it has to be a mixed integer model or one of the related model types, like MINLP or MIQCP. See section Classification of Models for a full listing of all GAMS model types.

Binary Variables

Binary variables can take values of 0 (zero) and 1 (one) only. They are declared as follows:

```
Binary Variable var_name [(index_list)] [text];
```

The keyword `binary` indicates that this is a binary variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as `binary` later. Consider the following code snippets from the orthogonal Latin Square model **[LATIN]**:

```
Sets k  "rows"      / row1*row4 /
     l  "columns"   / col1*col4 /
     v  "values"    / val1*val4 /;
alias (i,j,v);

Variables x(i,j,k,l) "pairs (i,j) allocated to cell(k,l)"
          z          "some objective";
Binary Variable x;

Equations c1(i,j)    "for each cell pick only one item pair";
c1(i,j).. sum((k,l), x(i,j,k,l)) =e= 1;
```

Note that the binary variable `x` is used in equation `c1` to model the restriction that in each cell only one item pair is allowed. Binary variables are often used to model logical conditions such as imposing mutual exclusivity or complementarity.

Note that the default lower bound is 0 (zero) and the default upper bound is 1 (one). If the relaxed versions of the discrete models is solved, binary variables are treated like positive variables with the upper bound of 1. In addition, an infinite priority may be used to override binary specifications, see section Setting Priorities for Branching below for more information.

Even though the only possible values are 0 and 1, a solver might return a value for binary variable that is only *close* to 0 or 1. Every solver works with tolerances and also uses a tolerance to determine if a value is *close enough* to an integer values. So it is unwise to use code as `a(i)$(b.l(i)=1) = yes;` because one will potentially miss some elements. A safe way to write such code is: `a(i)$(b.l(i)>0.5) = yes;`. Rounding the level of a binary variable after the solve is also possible, but it is not done by the solver or the solver link because even small rounding can lead to infeasibilities.

A binary variable can also have a truely fractional value after a solver if the model status does not indicate a feasible integer solution (model status 1 or 8).

Integer Variables

Integer variables are discrete variables that can take only values between their bounds. The user may change both bounds from the default value. The default lower bound is 0 (zero) and the default upper bound inside GAMS is `+inf`, and the same upper bound is passed on to the solver.

Note that in relaxed model types the integrality requirement is relaxed. In addition, an infinite priority may be used to override integer specifications, see section Setting Priorities for Branching below for more information. Integer variables are declared as follows:

```
Integer Variable var_name [(index_list)] [text];
```

The keyword `integer` indicates that this is an integer variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as `integer` later. Consider the following code snippets from the power scheduling model [**MAGIC**]:

```
Sets t  "demand blocks" / 12pm-6am, 6am-9am, 9am-3pm, 3pm-6pm, 6pm-12pm /
     g  "generators"    / type-1, type-2, type-3 /;

Variables  x(g,t)  "generator output (1000mw)"
           n(g,t)  "number of generators in use"
           cost    "total operating cost (1)";
Integer Variable n;
```

The integer variable `n` models the number of generators of various types that are in use at any of the time blocks.

Special Order Sets of Type 1 (SOS1)

SOS1 variables are a set of variables, such that at most one variable within the group may have a nonzero value. This variable may take any positive value. Special ordered sets of type 1 are defined as follows:

```
SOS1 Variable var_name (index_list) [text];
```

The keyword SOS1 indicates that this is a SOS1 variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as SOS1 later. Consider the following example:

```
SOS1 Variable s1(i), t1(k,j), w1(i,j,k) ;
```

Note that the members of the innermost (the right-most) index belong to the same SOS set. For example in the sets defined above, s1 represents one special ordered set of type 1 with i elements, t1 defines k sets with j elements each and w1 defines (i,j) sets with k elements each.

The default bounds for SOS1 variables are zero and +inf. As with any other variable, the user may change these bounds. Further, the user may explicitly provide whatever convexity row that the problem may need through an equation that requires the members of the SOS1 set to be less than a certain value. Any such convexity row will implicitly define bounds on each of the variables.

Consider the following example:

```
SOS1 Variable s1(i); Equation defsoss1;
defsoss1.. sum(i,s1(i)) =l= 3.5 ;
```

The equation defsoss1 implicitly defines the nonzero value that one of the elements of the SOS1 variable s1 may take as equal to or smaller than 3.5. Note that it is also possible that all variables s1 equal zero.

A special case arises when one of the elements of the set has to be nonzero and equal to a number, say 3.5. In this case equation defsoss1 will be:

```
defsoss1.. sum(i,s1(i)) =e= 3.5 ;
```

Frequently the nonzero value equals 1. As a result, the SOS1 variable is effectively a binary variable. It is only treated differently by the solver at the level of the branch and bound algorithm. For example, consider the following example where we want to model that one out of n options has to be selected. This is expressed as:

```
SOS1 Variable x(i); Equation defx;
defx.. sum(i, x(i)) =e= 1 ;
```

The variable x can be made binary without any change in meaning and the solution provided by the solver will be indistinguishable from the SOS1 case.

The use of special ordered sets may not always improve the performance of the branch and bound algorithm. If there is no natural *order* the use of binary variables may be a better choice. A good example of this is the classical assignment problem (see [H.P. Williams (2013) Model Building in Mathematical Programming], Wiley, Section 9.3.

Note that any model with SOS1 variables requires a MIP solver, because the solution process needs to impose the restrictions of at most one nonzero level values may be present.

For an example where SOS1 variables are used, see the production scheduling model **[PRODSCHX]**.

Special Order Sets of Type 2 (SOS2)

SOS2 variables are a set of variables, such that at most two variables within the set may have nonzero values and these variables have to be adjacent. This requirement implies that the set is *ordered*, see chapter Sets as Sequences: Ordered Sets for details on ordered sets in GAMS. Note that the nonzero variables may take any positive value. Special ordered sets of type 2 are defined as follows:

```
SOS2 Variable var_name [(index_list)] [text];
```

The keyword SOS2 indicates that this is a SOS2 variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as SOS2 later. Consider the following example:

```
Set i  / i1*i5 /;
SOS2 Variable s2(i), t2(k,j), w2(i,j,k);
```

The members of the innermost (the right-most) index belong to the same set. For example, in the sets defined above, s2 represents one special ordered set of type 2 with elements for each member of the set i. At most two variables s2 may be nonzero and they must reference adjacent elements of the set i. Note that the variables s2('i1') and s2('i2') are adjacent, but the variables s2('i1') and s2('i3') are not. Further, t2 defines k sets of SOS2 variables with j elements each and the adjacency requirement refers to the set j which must be ordered. Similarly, w2 defines (i,j) sets with k elements each and the adjacency requirement refers to the set k which must be ordered.

The default bounds for SOS2 variables are zero and +inf. As with any other variable, the user may change these bounds. SOS2 variables are most often used to model piece-wise linear approximations to nonlinear functions. The production scheduling model [**PRODSCHX**] shows SOS type formulations with binary, SOS1 and SOS2 sets.

Note that any model with SOS2 variables requires a MIP solver, because the solution process needs to impose the restrictions of adjacency and that no more than two nonzero level values may be present.

Semi-Continuous Variables

Semi-continuous variables are either zero or above a given minimum level. This can be expressed algebraically as: either $x = 0$ or $L \leq x \leq U$. By default, the lower bound L is 1 and the upper bound U is +inf. As usual, these bounds may be changed with the variable attributes .lo and .up. Semi-continuous variables are defined as follows:

```
SemiCont Variable var_name [(index_list)] [text];
```

The keyword semicont indicates that this is a semi-continuous variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as semicont later. Consider the following example:

```
SemiCont Variable x;
x.lo = 1.5;  x.up = 23.1;
```

The slice of code above declares the variable x to be a semi-continuous variable that may either be zero or behave as a continuous variable between 1.5 and 23.1.

Note that any model with semi-continuous variables requires a MIP solver, because the solution process needs to impose the discontinuous jump between zero and the threshold value.

Note

- Not all MIP solvers allow semi-continuous variables. We recommend users to verify how the solver they are interested in handles semi-continuous variables by checking the relevant section of the respective solver manual.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than zero, otherwise GAMS will report an error.
- The variable solution listing might show the level outside the lower and upper bound which for other variables indicates an infeasible variable, but not so for semi-continuous variables.
- Semi-continuous variables are especially helpful if the upper bound is `+inf` and no implicit bound can be easily derived. If a finite upper bound is available it can be computational more efficient to replace the semi-continuous variable `sc` with lower bound `scLow` by a continuous variable `x` and binary variable b and the following equations:

```
Equation xForceLowerBnd   "Force x to be greater than scLow if b is 1"
         xForceZero       "Force x to be zero if b is zero";
xForceLowerBnd.. x =g= scLow*b;
xForceZero..     x =l= x.up*b;
```

Semi-Integer Variables

Semi-integer variables are either zero or integer and above a given minimum value. This can be expressed algebraically as: either $x = 0$ or $x \in \{L, \ldots, U\}$. By default, the lower bound L is 1 and the upper bound U inside GAMS is `+inf` and the same values are passed on to the solver. As usual, these default bounds may be changed with the variable attributes `.lo` and `.up`. Note that in relaxed model types the integrality requirement is relaxed. In addition, an infinite priority may be used to override integer specifications, see section Setting Priorities for Branching below for more information.

Semi-integer variables are defined as follows:

```
SemiInt Variable var_name [(index_list)] [text];
```

The keyword `semiint` indicates that this is a semi-integer variable and then the usual conventions for variable declarations are followed. Alternatively, the variable may be declared first and specified as `semiint` later. Consider the following example:

```
SemiInt Variable x;
x.lo = 2; x.up = 25;
```

The slice of code above declares the variable `x` to be a semi-integer variable that may either be zero or take any integer value between 2 and 25. Note that the bounds for `semiint` variables have to take integer values, otherwise GAMS will flag an error during model generation. Note further, that any model with semi-integer variables requires a MIP solver.

Note

- Not all MIP solvers allow semi-integer variables. We recommend users to verify how the solver they are interested in handles semi-integer variables by checking the relevant section of the respective solver manual.
- The lower bound has to be less than the upper bound, and both bounds have to be greater than zero, otherwise GAMS will report an error.
- The variable solution listing might show the level outside the lower and upper bound which for other variables indicates an infeasible variable, but not so for semi-integer variables.

- Semi-integer variables are especially helpful if the upper bound is `+inf` and no implicit bound can be easily derived (together with the appropriate IntVarUp setting). If a finite upper bound is available, it can be computationally more efficient to replace the semi-integer variable `si`, with lower bound `siLow`, by an integer variable `i` and a binary variable b and the following equations:

```
Equation iForceLowerBnd   "Force i to be greater than siLow if b is 1"
         iForceZero       "Force i to be zero if b is zero";
iForceLowerBnd.. i =g= siLow*b;
iForceZero..     i =l= i.up*b;
```

4.37.2.2 Setting Priorities for Branching

By setting priorities users may specify an order for choosing variables to branch on during a branch and bound search for MIP models. Without priorities the MIP algorithm will internally determine which variable is the most suitable to branch on. Priorities for individual variables may be used only if the model attribute `.prioropt` is set to 1; the respective GAMS statement is:

```
mymodel.prioropt = 1;
```

Here `mymodel` is the name of the model specified in the model statement. The default value is NA.

If the model attribute `.prioropt` is set to 1, the variable attribute `.prior` may be used to set the priorities of individual discrete variables. Note that there is one `.prior` value for each individual component of a multidimensional variable. Priorities may be set to any real value; the default value is 1. As a general rule, the most important variables should be given the highest priority. The highest priority is denoted by the lowest nonzero value in the `.prior` attribute. Functionally, the attribute `.prior` establishes in what order variables are to be branched on in the branch-and-bound algorithm while searching for a solution. Variables with a specific `.prior` value will branched on earlier until all fractional variables with higher `.prior` values have been branched on.

Note

> The variable attribute `.prior` of a *discrete* variable may be used to relax the discrete restriction on that variable: setting the `.prior` value to `+inf` will relax a variable permanently (or until `.prior` gets a finite value assigned). This relaxation is done independently of the model attribute `.prioropt`.

Consider the following example:

```
z.prior(i,'small')   = 3;
z.prior(i,'medium')  = 2;
z.prior(i,'large')   = 1;
```

In this example the variables `z(i,'large')` are branched on before the variables `z(i,'medium')`, which in turn are branched on before the variables `z(i,'small')`.

Note that knowledge about the problem may help to determine which variables should be considered first. For example, consider a problem with a binary variable `u` representing a yes/no decision whether to build a factory and other binary variables representing equipment selections within that factory. We would naturally want to explore whether or not the factory should be built before considering what specific equipment to be purchased within the factory. Therefore we would set the priority values lower for `u`. By assigning a higher priority - a lower value of the attribute `.prior` - to the build/nobuild decision variable `u`, we can force this logic into the tree search and thus speed up computation time since uninteresting portions of the tree are left unexplored.

Note

- The lower the value given to the `.prior` suffix, the higher the priority for branching.
- All members of any `SOS1` or `SOS2` set should be given the same priority value since it is the set itself which is branched upon rather than the individual members of the set.
- While any value is accepted for `.prior` many solvers scale all giving priorities in the integer range of $0, ..., 1000$.
- Branching priorities were a very important feature in the early days of mixed integer programming. Nowadays, it is not easy to find branching priorities that improve on the solvers default selection.
- Global non-linear optimization solvers branch on continuous variables too (see, for example BARON). In GAMS one cannot set the branching priority of a continuous variable. Such branching priorities need to be communicated via a solver option file.

4.37.2.3 Miscellaneous Hints

We will conclude the discussion of mixed integer models in GAMS with this section where we offer a variety of hints that are meant to make special facilities of mixed integer programming solvers more accessible.

Model Attributes for Mixed Integer Programming in GAMS

GAMS offers several model attributes that may be used to influence MIP solver performance or report on results of MIPs. These model attributes include Cheat, CutOff, NodLim, ObjEst, OptCA, OptCR, PriorOpt and TryInt.

The Branch and Cut and Heuristic Facility

Hard MIP problems can be solved faster with the help of user supplied routines that generate cutting planes and good integer feasible solutions. The GAMS Branch-and-Cut-and-Heuristic (BCH) automates all major steps necessary to define, execute and control the use of user defined routines within the framework of general purpose branch-and-cut codes. It is documented in Branch-and-Cut-and-Heuristic Facility (BCH).

Branch and Bound Output

While the log output for each solver differs, some key figures are usually displayed for branch-and-bound based solvers. For example, solving a linear mixed integer model with CPLEX will yield output like the following:

```
       Nodes                                          Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Bound    ItCnt     Gap

*     0+    0                           0.0000   5.05646e+08             ---
Found incumbent of value 0.000000 after 0.01 sec. (0.73 ticks)
      0     0   2.23031e+07    12        0.0000   2.23031e+07       17     ---
      0     0   2.23031e+07     7        0.0000       Cuts: 8       23     ---
*     0+    0                       2.23031e+07   2.23031e+07             0.00%
Found incumbent of value 2.2303094e+07 after 0.02 sec. (1.08 ticks)
 ...

Fixing integer variables, and solving final LP...
```

```
...

Solution satisfies tolerances.

MIP Solution:     22303093.628684    (23 iterations, 0 nodes)
Final Solve:      22303093.628684    (0 iterations)

Best possible:    22303113.950091
Absolute gap:           20.321407
Relative gap:            0.000001
```

A brief explanation of the columns follows:

- `Node` is the number of branch and bound nodes so far.
- `Nodes Left` is the number of problems created during the branching process that are yet to be examined.
- `Objective` gives the current objective function value of the relaxes node problem.
- `IInf` gives the number of discrete variables with fractional solution levels.
- `Best Integer` gives the incumbent solution. Note the last solution in that column is not necessarily the global best solution.
- `Cuts/Best Bound` gives the current lower bound on the solution.
- `ItCnt` gives the accumulated LP iteration count
- `Gap` gives the maximum percentage difference from the theoretical optimum.

Note that it is common that solves of mixed integer models end with a gap between the solution found and the best possible solution. This may be controlled by limits (e.g. time), solver options and model attributes like .OptCR and .OptCA.

Nonlinear MIPs

Modelers may wish to impose integer restrictions on nonlinear formulations combining two hard model types: MIP and NLP. Such MINLP models can be solved with a selection of solvers. Many solvers, e.g. DICOPT and SBB, provide a local optimum where others, e.g. ANTIGONE and BARON provide a global optimum. In most cases both types of MINLP solver make use of MIP and NLP solvers to calculate a solution. Such *subsolvers* need to be licensed for the solver to succeed.

Model Termination Conditions and Recommended Actions

Recall that the termination condition of the model after the solution process has been completed is stored in the model attribute .modelStat. A list of all possible model statuses is given in section Model Status. We can easily check for the existence of a feasible solution (status 1 and 8). Linear models and MINLP problems solved with global solvers can achieve the "OPTIMAL" status, given sufficient resources (time) and a setting of OptCR and OptCA to 0. All other cases do not yield a feasible integer solution. If a problem is reported as infeasible (status 4,5,10, and 19), it might be a good idea to see if the relaxed version of the model is already infeasible. Debugging models that are relaxed feasible but integer infeasible is very difficult.

Frequent Problems

There are some problems users frequently encounter either due to GAMS settings or problem characteristics:

Default bounds

One needs to be aware that while the GAMS upper bound for integer and semi-integer variables is +inf, the bound that is passed to the solver can be different, namely 100 (see the discussion about integer variables). This can lead to unexpected results (e.g. infeasibilities or suboptimal solutions declared as optimal).

Ending with a gap – large default for `optCR` (10%)

MIP solves often end with a gap between the solution found and the best possible solution. This is controlled by OptCR or OptCA or by non-convergence. Note that the default value of 0.1 for `optCR` is relatively large. Users may want to reduce this to a smaller value. We will discuss the other cause of a gap next.

The nonending quest

Integer programming is a quite desirable formulation technique. However, integer problems are theoretically hard and the solution process (in the worst case) of exponential complexity. There are many ways that focus on improving the solution time of the solver. As with all models scaling is important (especially when using bigM formulations). For particular problems many different formulations exist and the literature about a particular problem together with the ability of GAMS to rapidly prototype and experiment is the best constellation to get the best results for the problem at hand. For fine tuning, some MIP solvers provide automated tuning tools (see e.g. Cplex tuning) that tweak the solver options to get the best performance.

4.37.3 Model Scaling - The Scale Option

The rules for good scaling are exclusively based on algorithmic needs. GAMS has been developed to increase the efficiency of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as *natural* as possible. The units of measurement are one part of this natural notation. However, there is a potential conflict between what the modeler thinks is a good unit and what constitutes a well-scaled model.

4.37.3.1 The Scale Option

To facilitate the translation between a natural model and a well scaled model, GAMS has introduced the concept of a *scale factor*, both for variables and equations. The notations and definitions are quite simple. Scaling is turned off by default. Setting the model attribute `.scaleopt` to 1 turns on the scaling feature. For example,

```
model mymodel /all/ ;
mymodel.scaleopt = 1 ;
solve mymodel using nlp maximizing dollars ;
```

The statement should be inserted somewhere after the `model` statement and before the `solve` statement. To turn scaling off again, `mymodel.scaleopt` has to be set to zero before the next solve.

In most respects GAMS scaling is hidden from the user. The solution values reported back from a solution algorithm are always reported in the notation of the user. The algorithm's internal representation of the equations and variables are only reflected in the derivatives in the equation and column listings in the GAMS output if the values of the options limrow and limcol are positive. In addition, the internal representations will appear in the debugging output from the solution algorithm if the option sysout is set to `on`.

4.37.3.2 Scaling Variables

The scale factor of a variable is defined using the variable attribute `.scale` in the following way:

```
myvar.scale(i,j) = c;
```

The scale factor `c` is a number or a numerical expression that evaluates to a number. Note that the default scale factor is 1.

Note that there is one scale value for each individual component of a multidimensional variable.

Assume that c is the scale factor of a variable V_u. Assume further, that the variable seen by the algorithm is V_a. Then we have: $\mathbf{V_a} = \mathbf{V_u}/\mathbf{c}$. This means that each variable as seen by the user is *divided by* the scale factor.

For example, consider the following code snippet:

```
Positive Variables x1, x2;
Equation eq;
eq.. 200*x1 + 0.5*x2 =l= 5;
x1.up = 0.01;
x2.up = 10;
x1.scale = 0.01;
x2.scale = 10;
```

By setting `x1.scale` to 0.01 and `x2.scale` to 10, the model seen by the solver is:

```
Positive Variables xPrime1, xPrime2;
Equation eq;
eq.. 2*xPrime1 + 5*xPrime2 =l= 5;
xPrime1.up = 1;
xPrime2.up = 1;
```

Note that the solver does not see the variables `x1` or `x2`, but rather the scaled (and better-behaved) variables `xPrime1` and `xPrime2`. Note further, that upper and lower bounds on variables are automatically scaled in the same way as the variable itself.

Attention

- Discrete variables cannot be scaled.
- **Expert Note**. Internally, GAMS stores with each variable and equation one additional attribute or field (besides fields for level, marginal and lower and upper bound). Depending on the type of variable and sometimes even model type or solver, this field has different names in the GAMS language. For continuous variables, the field is called *scale*, while for discrete variable it is called *prior*. For stochastic 2-stage linear program models solved with DECIS, this field is called *stage*. The field `.stage` can lead to confusing results. Consider the following example:

  ```
  Variable x;
  x.scale = 0.1;
  display x.stage;
  ```

  The output is:

  ```
  ----      3 VARIABLE x.scale = 0.100
  ```

  The field `.scale` has to be in a certain range (>1e-20 and no special value), but this is only checked at model generation time. The field `.prior` can be any number and even `+inf` (but no other special values). For further information on `.prior`, see section Setting Priorities for Branching. For an introduction to variable and equation fields, see sections Variable Attributes and Equation Attributes respectively.

4.37.3.3 Scaling Equations

The scale factor of an equation is defined using the equation attribute `.scale` in the following way:

```
mzeqn.scale(i,j) = d;
```

The scale factor `d` is a number or a numerical expression that evaluates to a number. Note that the default scale factor is 1.

Assume that d is the scale factor of an equation G_u. Assume further, that the equation seen by the algorithm is G_a. Then we have: $\mathbf{G_a} = \mathbf{G_u}/\mathbf{d}$. This means that each equation as seen by the user is *divided by* the scale factor.

For example, consider the following equations:

```
Positive Variables y1, y2;
Equations eq1, eq2;
eq1..   200*y1 + 100*y2 =l= 500;
eq2..       3*y1 - 4*y2 =g= 6;
```

By setting `eq1.scale` to 100, the model seen by the solver is:

```
Positive Variables y1, y2;
Equations eqPrime1, eq2;
eqprime1..   2*y1 + 1*y2 =l= 5;
eq2..        3*y1 - 4*y2 =g= 6;
```

Note

> The user may have to perform a combination of equation and variable scaling to obtain a well-scaled model.

Consider the following example:

```
Positive variables x1, x2;
Equations  eq1, eq2;
eq1..   100*x1 + 5*x2 =g= 20;
eq2..   50*x1 - 10*x2 =l= 5;
x1.up = 0.2;
x2.up = 1.5;
```

Setting the following scale values:

```
x1.scale  = 0.1;
eq1.scale = 5;
eq2.scale = 5;
```

will result in the solver seeing the following well-scaled model:

```
Positive Variables xPrime1, x2;
Equations eqPrime1, eqPrime2;
eqPrime1..   2*xPrime1 + x2 =g= 4;
eqPrime2..   xPrime1 - 2*x2 =l= 1;
xPrime1.up = 2;
x2.up      = 1.5;
```

4.37.3.4 Scaling Derivatives

In nonlinear models the derivatives also need to be well-scaled. Assume that the derivatives in the model of the user are denoted by $d(G_u)/d(V_u)$. Assume further, that the derivatives in the scaled model seen by the algorithm are denoted by $d(G_a)/d(V_a)$. Then we have: $\mathbf{d(G_a)/d(V_a) = d(G_u)/d(V_u) \cdot c/d}$, where c is the scale factor for the variable and d is the scale factor for the equation.

The user may affect the scaling of derivatives by scaling both the equation and variable involved.

4.37.3.5 Scaling Data

Scaling input data is independent of the model attribute `.scaleopt` and may contribute considerably towards achieving a well-scaled model. We recommend users to try to define the units of the input data such that the largest values expected for decision variables and their marginals is under a million, if possible.

For example, in US agriculture about 325 million acres are cropped and the corn crop is 9-10 billion bushels per year. When defining production data, we could enter land in 1000's of acres and all other resources in 1000's of units. We could also define the corn crop in millions of bushels. The data will be simultaneously scaled, hence if resource endowments are quoted in 1000's, corn yields are divided by 1000. This scaling results in a corn production variable in the units of millions. Consumption statistics would need to be scaled accordingly. Money units could also be in millions or billions of dollars. Such data scaling generally greatly reduces the disparity of coefficients in the model.

4.37.4 Conic Programming in GAMS

Conic programming models minimize a linear function over the intersection of an affine set and the product of nonlinear cones. The problem class involving second order (quadratic) cones is known as Second Order Cone Programs (SOCP). These are nonlinear convex problems that include linear and (convex) quadratic programs as special cases.

Conic programs allow the formulation of a wide variety of application models, including problems in engineering and financial management. Examples are portfolio optimization, Truss topology design in structural engineering, Finite Impulse Response (FIR) filter design and signal processing, antenna array weight design, grasping force optimization, quadratic programming, robust linear programming and norm minimization problems.

For more information, see References and Links at the end of this section.

4.37.4.1 Introduction to Conic Programming

Conic programs can be thought of as generalized linear programs with the additional nonlinear constraint $x \in C$, where C is required to be a convex cone. The resulting class of problems is known as *conic optimization* and has the following form:

$$\begin{array}{rl} \text{minimize} & c^T x \\ \text{subject to} & Ax \leq r^c, \\ & x \in [l^x, u^x] \\ & x \in C \end{array}$$

where $A \in \mathbb{R}^{m \times n}$ is the constraint matrix, $x \in \mathbb{R}^n$ the decision variable and $c \in \mathbb{R}^n$ the objective function cost coefficients. The vector $r^c \in \mathbb{R}^m$ represents the right-hand side and the vectors $l^x, u^x \in \mathbb{R}^n$ are lower and upper bounds on the decision variable x.

Now partition the set of decision variables x into sets $S^t, t = 1, ..., k$, such that each decision variables x is a member of at most one set S^t. For example, we could have

$$S^1 = (x_1, x_4, x_7) \quad \text{and} \quad S^2 = (x_6, x_5, x_3, x_2).$$

Let x_{S^t} denote the variables x belonging to set S^t. Then define

$$C := \{x \in \mathbb{R}^n : x_{S^t} \in C_t, t = 1, ..., k\},$$

where C_t must have one of the following forms:

- **Quadratic cone** (also referred to as Lorentz or ice cream cone):

$$C_t = \left\{ x \in \mathbb{R}^{n^t} : x_1 \geq \sqrt{\sum_{j=2}^{n^t} x_j^2} \right\}.$$

- ***Rotated*** **quadratic cone** (also referred to as hyperbolic constraints):

$$C_t = \left\{ x \in \mathbb{R}^{n^t} : 2x_1 x_2 \geq \sum_{j=3}^{n^t} x_j^2,\ x_1, x_2 \geq 0 \right\}.$$

These two types of cones allow the formulation of quadratic, quadratically constrained and many other classes of nonlinear convex optimization problems.

4.37.4.2 Implementation of Conic Constraints in GAMS

The recommended way to write conic constraints is by using a quadratic formulation. Many solvers have the capability to identify the conic constraints in a QCP model even if it is not in perfect form but can be easily reformulated to fit in the described form. However, some solvers (namely MOSEK) expect the conic constraints to be precisely in the form given above. In earlier versions this form was enforced by a special type of equation, the `=c=` equation type. Moreover, such solvers have other requirements (e.g. disjunctive cones) that can be easily fulfilled by simple reformulation steps. Much progress is expected on the solver side in the coming years, so we don't go into much detail here.

Observe that we could formulate conic problems as regular NLPs using the following constraints:

- **Quadratic cone:**

```
x('1') =g= sqrt[ sum(i$[not sameas(i,'1')], sqr[x(i)]) ];
```

- **Rotated quadratic cone:**

```
2*x('1')*x('2') =g= sum(i$[not sameas(i,'1') and not sameas(i,'2')], sqr[x(i)]);
```

  Here `x('1')` and `x('2')` are positive variables.

The following example illustrates the different formulations for conic programming problems. Note that a conic optimizer usually outperforms a general NLP method for the reformulated (NLP) cone problems.

4.37.4.3 Example

Consider the following example, which illustrates the use of rotated conic constraints. We will give reformulations of the original problem in regular NLP form and in conic form (with conic constraints).

The original problem is:

$$\text{minimize} \sum_{i=1}^{n} \frac{d_i}{x_i} \tag{4.2}$$

$$\text{subject to } a\,x \leq b \tag{4.3}$$

$$x_i \in [l_i, u_i], \qquad i = 1, \ldots, n \tag{4.4}$$

where $x \in \mathbb{R}^n$ is the decision variable, $d, a, l, u \in \mathbb{R}^n$ are parameters with $l_i > 0$ and $d_i \geq 0$ and $b \in \mathbb{R}$ is a scalar parameter. The original model may be written in GAMS using the equations:

```
defobj..  sum(n, d(n)/x(n)) =e= obj;
e1..      sum(n, a(n)*x(n)) =l= b;
Model orig /defobj, e1/;
x.lo(n) = l(n);
x.up(n) = u(n);
```

We can write an equivalent QCP formulation by using the substitution $t_i = 1/x_i$ in the objective function and adding a constraint. As we are dealing with a minimization problem, $d_i \geq 0$ and $x_i \geq l_i > 0$, we can relax the equality $t_i x_i = 1$ into an inequality $t_i x_i \geq 1$, which results in an equivalent problem with a convex feasible set:

$$\text{minimize} \sum_{i=1}^{n} d_i t_i \tag{4.5}$$

$$\text{subject to } a\,x \leq b \tag{4.6}$$

$$t_i x_i \geq 1, \qquad i = 1, \ldots, n \tag{4.7}$$

$$x \in [l, u], \tag{4.8}$$

$$t \geq 0, \tag{4.9}$$

$$\tag{4.10}$$

where $t \in \mathbb{R}^n$ is a new decision variable. The GAMS formulation of this QCP is:

```
defobjc..     sum(n, d(n)*t(n)) =e= obj;
e1..          sum(n, a(n)*x(n)) =l= b;
coneqcp(n)..  t(n)*x(n)         =g= 1;

Model cqcp /defobjc, e1, coneqcp/;
t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);
```

Note that the constraints $t_i x_i \geq 1$ are almost in rotated conic form. If we introduce a variable $z \in \mathbb{R}^n$ with $z_i = \sqrt{2}$, then we can reformulate the problem using conic constraints as:

$$\text{minimize} \sum_{i=1}^{n} d_i t_i \tag{4.11}$$

$$\text{subject to } a\,x \leq b \tag{4.12}$$

$$z_i = \sqrt{2}, \qquad i = 1, \ldots, n \tag{4.13}$$

$$2 t_i x_i \geq z_i^2, \qquad i = 1, \ldots, n \tag{4.14}$$

$$x \in [l, u], \tag{4.15}$$

$$t \geq 0, \tag{4.16}$$

$$\tag{4.17}$$

The GAMS formulation using conic equations is:

```
defobjc..        sum(n, d(n)*t(n)) =e= obj;
e1..             sum(n, a(n)*x(n)) =l= b;
e2(n)..          z(n) =e= sqrt(2);
coneperfect(n).. 2*x(n)*t(n) =g= sqr(z(n));

Model cperfect  /defobjc, e1, e2, coneperfect/;
t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);
```

The complete model is listed below:

```
Set       n / n1*n10 /;
Parameter d(n), a(n), l(n), u(n);
Scalar    b;

d(n) = uniform(1,2);
a(n) = uniform (10,50);
l(n) = uniform(0.1,10);
u(n) = l(n) + uniform(0,12-l(n));

Variables x(n);
x.l(n) = uniform(l(n), u(n));
b      = sum(n, x.l(n)*a(n));

Variables t(n), z(n), obj;
Equations defobjc, defobj, e1, e2(n), coneqcp(n), coneperfect(n), conenlp(n);

defobjc..          sum(n, d(n)*t(n)) =e= obj;
defobj..           sum(n, d(n)/x(n)) =e= obj;
e1..               sum(n, a(n)*x(n)) =l= b;
coneqcp(n)..       t(n)*x(n) =g= 1;
e2(n)..            z(n) =e= sqrt(2);
coneperfect(n)..   2*x(n)*t(n) =g= sqr(z(n));

Model cqcp     /defobjc, e1, coneqcp/;
Model cperfect /defobjc, e1, e2, coneperfect/;
Model orig     /defobj, e1/;

t.lo(n) = 0;
x.lo(n) = l(n);
x.up(n) = u(n);

Option qcp=cplex;
Solve cqcp min obj using qcp;
Option qcp=mosek;
Solve cperfect min obj using qcp;
Solve orig min obj using nlp;
```

4.37.4.4 Sample Conic Models in GAMS

Conic models in the GAMS model library include:

- [**EMFL**]: A multiple facility location problem,
- [**FDESIGN**]: Linear Phase Lowpass Filter Design,
- [**IMMUN**]: Financial Optimization: Risk Management,
- [**PMEANVAR**]: Mean-Variance Models with variable upper and lower Bounds,
- [**QP7**]: A portfolio investment model using rotated quadratic cones (quadratic program using a Markowitz model),
- [**ROBUSTLP**]: Robust linear programming as an SOCP,
- [**SPRINGCHAIN**]: Equilibrium of System with Piecewise Linear Spring,
- [**TRUSSM**]: Truss Toplogy Design with Multiple Loads

4.37.4.5 References and Links

- A. Ben-Tal and A. Nemirovski, Lectures on Modern Convex Optimization: Analysis, Algorithms, and Engineering Applications, MPS/SIAM Series on Optimization, SIAM Press, 2001.
- M. Lobo, L. Vandenberghe, S. Boyd and H. Lebret, Applications of Second-Order Cone Programming, Linear Algebra and its Applications, 284:193-228, November 1998, Special Issue on Linear Algebra in Control, Signals and Image Processing.
- MOSEK ApS, MOSEK Modeling Cookbook, 2015.
- G. Pataki G and S. Schmieta, The DIMACS Library of Semidefinite-Quadratic-Linear Programs. Tech. rep., Computational Optimization Research Center, Columbia University, 2002.
- Seventh Dimacs Implementation Challenge on Semidefinite and Related Optimization Problems.

4.37.5 Indicator Constraints

An indicator constraint is a way of expressing relationships between variables by specifying a binary variable to control whether or not a constraint takes effect. For example, indicator constraints are useful in problems where there are fixed charges to express only if a given variable comes into play.

So-called Big M formulations require to estimate an upper bound of an expression in a model. In most cases the model data can be used to determine relatively small number for such coefficients. In some cases it is not possible to find small Big M values and a resulting solution may exhibit trickle flow and have other unwanted side-effects. The main purpose of indicator constraints is to overcome these limitations of Big M formulations. Generally, the use of indicator constraints is not warranted when the unwanted side-effects of Big M formulations are not present.

Consider the following example:

```
constr01.. x1 + x2 + x3 =l= 1e+9*y; // may cause problems
```

Here we use a Big M formulation, that relies on the x values to sum to less than one billion. Note that this formulation may cause numeric instability or undesirable solutions in some situations. Alternatively, we could use an indicator constraint:

```
constr01$(y=0).. x1 + x2 + x3 =l= 0; // alternative
```

Note that y is a binary variable; the logical condition makes sure that the constraint is only active if $y = 0$. Unfortunately, the `$()` expressions do not allow endogenous variables and we need a different way to specify such implications. Indicators are supported by CPLEX, GUROBI, SCIP, and XPRESS.

We should mention a formulation of indicators without the explicit indicator constraints. We can use a SOS1 constraint to express that a constraint holds or not based on a binary variable by having a SOS1 set of a slack variable and the binary variable `SOS1(slack,1-y)`:

```
Positive variable slack;
constr01.. x1 + x2 + x3 =l= slack; // alternative
```

Recall that only one of the variables in a SOS1 constraint can be non-zero, so if `1-y` is non-zero (i.e. `y=0`) then `slack` must be zero, i.e. the constraint holds. If `1-y=0` is non-zero, i.e. `y=1` then `slack` can be used to make the constraint feasible for any setting of `x1`, `x2`, and `x3`. Here two unrelated variable, `slack` and `y` are part of an SOS1 constraint and some tricks are required to formulate this properly in GAMS:

```
Set oo / slack, yExpr /;
SOS1 variable indic(oo);
Equation iOn, iOff;
Positive variable slack;
constr01.. x1 + x2 + x3 =l= slack;
iOn..      indic('slack') =e= slack;
iOff..     indic('yExpr') =e= 1-y;
```

4.37.5.1 Indicator Constraints with GAMS

In the remainder of this section we will describe how to specify indicator constraints when using GAMS. Please consult the corresponding solver manual for information on whether indicator constraints are supported and possible differences in their specification.

The example from above will be implemented with a constraint in combination with additional information in a solver option file, e.g. CPLEX: `cplex.opt`:

```
constr01.. x1 + x2 + x3 =l= 0;
```

and the following entry in the option file:

```
indic constr01$y 0
```

This has the following effect: equation `constr01` will become an indicator constraint and becomes active in a solution where the binary variable takes the value 0. If the value of y in a solution is 1, the constraint is not active.

Note that this way of entering an indicator constraint is dangerous since the option files changes the model (usually an option file has some effect on the performance of the algorithm). Therefore, the solver will abort if there is a problem processing the indicator options in the solver option file.

Attention

> If the model is given to a solver without the option file containing the indicator mapping (or to a solver that does not understand the `indic` keyword, a very different model will be solved. The current implementation of indicator constraints requires a significant amount of caution from the user.

There are two ways of entering the equation/binary variable mapping in a solver option file: with an indexed format and using labels.

The *indexed format* is a convenient shorthand notation which borrows its syntax from the GAMS syntax. It requires that the indices for the binary variable are already present in the index set of the equation.

Consider the following example of an invalid GAMS syntax with an endogenous variable in the dollar condition:

```
equ1(i,j,k)$(ord(i) < ord(j) and bin1(i,k)=1).. lhs =l= rhs;
```

This may be specified with the following equation in the GAMS file:

```
equ1(i,j,k)$(ord(i) < ord(j)).. lhs =l= rhs;
```

plus a solver option file with the following entry:

```
indic equ1(i,j,k)$bin1(i,k) 1
```

The *label format* is used in cases where the binary variable indices are not present in the equation indices or the binary variable is adjusted with lags or leads. In these cases the mapping of all individual equations and variables of the indicator constraints need to be specified. An example follows.

```
Set             i /i1*i3/, j /j1*j2/;
Binary variable bin1(j);
Equation        equ1(i,j);

equ1(i,j)$(bin1(j++1)=0)..   lhs =e= 0;
```

Note that the example above is not valid GAMS code. Instead, we will combine a valid GAMS equation and a solver option file using the label format as follows:

```
equ1(i,j).. lhs =e= 0;
```

and the solver option file

```
indic equ1('i1','j1')$bin1('j2') 0
indic equ1('i1','j2')$bin1('j1') 0
indic equ1('i2','j1')$bin1('j2') 0
indic equ1('i2','j2')$bin1('j1') 0
indic equ1('i3','j1')$bin1('j2') 0
indic equ1('i3','j2')$bin1('j1') 0
```

Note that the lines in such option files need not be entered manually. They may be easily generated using the GAMS The Put Writing Facility. For example, the lines above may be generated as follows:

```
file fcpx / cplex.opt /;
fcpx.pc=8;
loop((i,j), put fcpx 'indic' equ1(i,j) '$' bin1(j++1) '0'/);
putclose fcpx;
```

Attention

Mixing and matching between indexed format and label format is not possible. While expressions like `eq(i,j,k)$x(i,k)` and `eq('i1','j2','k3')$x('i1','k6')` are valid, combinations such as `eq(i,'j2',k)$x(i,k)` are not supported.

There are situations where the indicator binary variable exist in the indicator constraint only and hence will *not* be *generated* by GAMS to be passed on to the solver. In such cases the solver will issue the following error message:

```
Error: Column is not of type Variable
```

There is an easy way to fix this problem: adding the binary indicator variable artificially to the model. For example, it may be added with the coefficient `eps` to the objective:

```
defobj.. z =e= ... + eps*sum(j, bin1(j));
```

4.37.5.2 An Example for Indicator Constraints with GAMS

In this subsection we will comment on parts of the model [**TRNSINDIC**]. This model uses big M formulations and indicator constraints to solve the same problem. In addition, a formulation that makes it easy to switch between these two is presented. It is a fixed-charge network example based on the well-known model [**TRNSPORT**].

Recall that `i` is the set of canning plants and `j` is the set of markets where cases of some product are to be shipped. First, the basic model is reformulated to a MIP by introducing the binary variable `use(i,j)` and two new equations:

```
Binary Variable use(i,j)     is 1 if arc is used in solution;

Equations       minship(i,j) ensure minimum shipping
                maxship(i,j) ensure zero shipping if use variable is 0;

minship(i,j)..  x(i,j) =g= minshipping*use(i,j);
maxship(i,j)..  x(i,j) =l= bigM*use(i,j);
```

Note that `minshipping` is a scalar denoting the minimum amount of cases that may be shipped and `bigM` is a sufficiently large number, as usual.

Next, the same problem is solved with indicator constraints: the two new equations are reformulated and a CPLEX option file with information on indicator constraints is added:

```
Equations  iminship(i,j) ensure minimum shipping using indicator constraints
           imaxship(i,j) ensure zero shipping if use variable is 0 using indicator constraints;

iminship(i,j)..   x(i,j) =g= minshipping;
imaxship(i,j)..   x(i,j) =e= 0;

Model indicatorModel /cost, supply, demand, iminship, imaxship/ ;

file fcpx Cplex Option file / cplex.opt /;
putclose fcpx 'indic iminship(i,j)$use(i,j) 1' / 'indic imaxship(i,j)$use(i,j) 0';
indicatorModel.optfile = 1;

Solve indicatorModel using mip minimizing z ;
```

Note that the option file contains an entry for each of the two equations. Note further, that the binary variable `use` moved from the equations to the option file. However, it also features in the objective equation of the model, therefore this is not problematic. Observe that the indexed format for the equation/binary variable mapping was used in the option file. Alternatively, the label format may be used:

```
loop((i,j),
     put fcpx 'indic ' iminship.tn(i,j) '$' use.tn(i,j) yes
            / 'indic ' imaxship.tn(i,j) '$' use.tn(i,j) no / );
  putclose fcpx;
```

In a final step the model is reformulated again such that the same problem may be solved with and without indicator constraints. This can be especially useful for debugging a model with indicator constraints. For details, see [**TRNSINDIC**].

Another example of a model with indicator constraints is the model [**BILINEAR**] where various formulations to represent bilinear product terms are demonstrated.

4.38 GAMS Output

4.38.1 Introduction

The output from GAMS contains many components in support for checking and comprehending a model. This chapter discusses the components of the GAMS output file generated from a GAMS run as well as ways to control the amount of diagnostic output produced. A GAMS run also generates a log which serves as a useful first step in analysing a GAMS run. The details regarding the log can be found in the GAMS log chapter.

The output file generated from a GAMS run is called *listing file*. The listing file has the file extension `.lst` and can be read using any text editor. By default the listing file has the same file name as the input file, but this can be changed using the command line parameter Output. See chapter The GAMS Call and Command Line Parameters for more information. The main components in the listing file are:

1. Compilation. The compilation output contains an echo print of the input file, possibly error messages, along with lists of GAMS objects and cross reference maps.
2. Execution. The execution output contains the results of display statements and possibly execution error messages.
3. Model Generation. The output generated during model generation contains listings of equations and variable listings as well as model statistics and possibly generation execution error messages.
4. Solution. The output generated when an external solver program processes the model is the solution report including the solve summary, the solver report, the solution listing and the report summary.
5. Post-Solution. The final components added to the listing file are the final execution summary and the file summary.

A small nonlinear program [**ALAN**] will be used to illustrate every component of the listing file. The possibilities for extension to large models with voluminous output (which is when the diagnostics are really useful) should be apparent. After covering all the components of the listing file we will discuss error reporting and conclude the chapter with a compiled list of how to customize the output file.

This chapter does not cover the output that can be generated by the user through display statements and put statements. This output is covered in the chapters The Display Statement and The Put Writing Facility respectively.

4.38.2 An Illustrative Model

[**ALAN**] is a portfolio selection model whose objective is to choose a portfolio of investments whose expected return meets a target while minimizing the variance. We will discuss a simplified version of this model. The input file is listed for reference.

```
$Title A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)

$onsymlist onsymxref onuellist onuelxref

$Ontext
This is a mini mean-variance portfolio selection problem described in
'GAMS/MINOS:Three examples' by Alan S. Manne, Department of Operations
 Research, Stanford University, May 1986.
$Offtext

* This model has been modified for use in the documentation

Set i  securities   /hardware, software, show-biz, t-bills/;
alias (i,j);

Scalar target     target mean annual return on portfolio % /10/,
       lowyield   yield of lowest yielding security,
       highrisk   variance of highest security risk ;

Parameters  mean(i)  mean annual returns on individual securities (%)
      / hardware   8
        software   9
        show-biz  12
        t-bills    7 /

Table v(i,j)  variance-covariance array (%-squared annual return)
                  hardware  software  show-biz  t-bills
       hardware      4         3        -1        0
       software      3         6         1        0
       show-biz     -1         1        10        0
       t-bills       0         0         0        0 ;

lowyield = smin(i, mean(i)) ;
highrisk = smax(i, v(i,i)) ;
display lowyield, highrisk ;

Variables  x(i)       fraction of portfolio invested in asset i
           variance   variance of portfolio
Positive Variable x;

Equations  fsum    fractions must add to 1.0
           dmean   definition of mean return on portfolio
           dvar    definition of variance;

fsum..     sum(i, x(i))                      =e=  1.0;
dmean..    sum(i, mean(i)*x(i))              =e=  target;
dvar..     sum(i, x(i)*sum(j,v(i,j)*x(j)))   =e=  variance;

Model portfolio  / fsum, dmean, dvar / ;
Solve portfolio using nlp minimizing variance;
display x.l, variance.l;
```

4.38.3 Compilation Output

The compilation output is the output produced during the initial check of the program, often referred to as compilation. It contains the following parts: the echo print of the input file, the symbol reference map, the symbol listing map, the unique element listing map, and the include file summary.

By default only the echo print and the include file summary is shown, the other parts are suppressed and may be turned on with dollar control options. See also section Customizing the Output File on how to control the appearance and amount of detail in the output file produced by the GAMS compiler.

4.38.3.1 The Echo Print of the Input File

The *echo print* of the program is always the first part of the output file. It is a listing of the input with added line numbers. It is possible to control the listing output using $offlisting to suppress the echo print and $onlisting to turn the echo print on again. The first few lines of the echo print of the Illustrative Model follow:

```
A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
C o m p i l a t i o n

   2
   4
      This is a mini mean-variance portfolio selection problem described in
      'GAMS/MINOS:Three examples' by Alan S. Manne, Department of Operations
       Research, Stanford University, May 1986.
  10
  11  * This model has been modified for use in the documentation
```

The default header of the GAMS listing file is `General Algebraic Modeling System`. This may be replaced using the $title at the start of a line. The text on that line will be the new header, as demonstrated above. After the header the compilation output is announced with the title `Compilation`.

The first three line numbers refer to empty lines in the input and line 11 is a comment line. If the lines on the input file are counted, it can be seen that this comment line appears after ten lines of dollar directives, comments and empty lines.

The dollar control options that follow in the model are used to display more information in the output file than the default and will be discussed in the subsections below.

Note

> By default, dollar control option lines are not listed in the echo print. The $onDollar and $offDolar controls the echoing of Dollar Control Options lines in the listing file. Dollar control option lines are also listed if they contain errors.

The block comment enclosed by $ontext and $offtext is listed without line numbers, while single line comments starting with asterisks (*) are listed with the respective line numbers. Observe that line numbers always refer to the physical line number in the input file. The remainder of the echo print follows:

```
12
13  Set i  securities   /hardware, software, show-biz, t-bills/;
14  alias (i,j);
15
16  Scalar target     target mean annual return on portfolio % /10/,
17         lowyield   yield of lowest yielding security,
18         highrisk   variance of highest security risk ;
19
20  Parameters  mean(i)  mean annual returns on individual securities (%)
21        / hardware   8
22          software   9
23          show-biz  12
24          t-bills    7 / ;
25
26  Table v(i,j)  variance-covariance array (%-squared annual return)
27                    hardware  software  show-biz  t-bills
28         hardware      4         3         -1        0
29         software      3         6          1        0
30         show-biz     -1         1         10        0
31         t-bills       0         0          0        0 ;
32
33  lowyield = smin(i, mean(i)) ;
34  highrisk = smax(i, v(i,i)) ;
35  display lowyield, highrisk ;
36
37  Variables  x(i)       fraction of portfolio invested in asset i
38             variance   variance of portfolio ;
39  Positive Variable x ;
40
41  Equations  fsum    fractions must add to 1.0
42             dmean   definition of mean return on portfolio
43             dvar    definition of variance;
44
45  fsum..     sum(i, x(i))                      =e=  1.0;
46  dmean..    sum(i, mean(i)*x(i))              =e=  target;
47  dvar..     sum(i, x(i)*sum(j,v(i,j)*x(j)))   =e=  variance;
48
49  Model portfolio  / fsum, dmean, dvar / ;
50  Solve portfolio using nlp minimizing variance;
51  display x.l, variance.l;
```

Lines in the echo print can be set to double space using $double and then reset to single space with $single. If errors were detected, the explanatory messages would be found at the end of the echo print. All discussions of error messages have been grouped together in the section Error Reporting below.

4.38.3.2 The Symbol Reference Map

The other parts of the compilation outputs are maps. They are extremely useful for users looking at a model written by someone else or trying to make changes in their own model after spending some time away from it. By default all maps are suppressed and can be turned on with dollar control options that are specified below.

The first map is the *symbol cross reference*. It can be turned on with the dollar control option $onSymXRef at the beginning of the program. The symbol cross reference map lists all identifiers (symbols) from the model in alphabetical order, identifies them as to data type, shows the line numbers where the symbols appear and classifies each appearance. The map that was generated as part of the output of our Illustrative Model is shown below:

```
Symbol Listing


SYMBOL      TYPE   REFERENCES

dmean        EQU   declared       42  defined       46 impl-asn       50      ref       49
dvar         EQU   declared       43  defined       47 impl-asn       50      ref       49
fsum         EQU   declared       41  defined       45 impl-asn       50      ref       49
highrisk     PARAM declared       18 assigned       34      ref       35
i            SET   declared       13  defined       13      ref       14       20       26       33
37     45     2*46     2*47  control       33       34       45       46       47
j            SET   declared       14      ref       26     2*47  control       47
lowyield     PARAM declared       17 assigned       33      ref       35
mean         PARAM declared       20  defined       21      ref       33       46
portfolio    MODEL declared       49  defined       49 impl-asn       50      ref       50
target       PARAM declared       16  defined       16      ref       46
v            PARAM declared       26  defined       26      ref       34       47
variance     VAR   declared       38 impl-asn       50      ref       47       50       51
x            VAR   declared       37 impl-asn       50      ref       39       45       46     2*47
```

In the first two columns the name and type of each identifier are given. For example, the last symbol listed is `x` which is defined to be of type `VAR`, a shorthand symbol for variable. The complete list of data types and their shorthand symbols is given below:

Shorthand Symbol	GAMS Data Type
ACRNM	acronym
EQU	equation
FILE	put file
MODEL	model
PARAM	parameter
SET	set
VAR	variable

For further details on data types in GAMS, see section Data Types and Definitions.

The symbol `highrisk` was declared as a `scalar`, but is listed as a `parameter` in the map above. Recall that `parameters`, `scalars`, and `tables` are three *input formats* for the data type `parameter`. For more information, see the overview Parameters, Scalars and Tables.

After the name and type of the identifier, a list of references to the symbol is given. References are grouped by reference type and identified by the line number in the output file. The actual reference can be found by referring to the echo print of the program, which has line numbers on it. In the case of the symbol `x` in the example above, the list of references as shown in the symbol reference map are as follows:

```
declared       37
impl-asn       50
ref            39       45       46     2*47       51
```

This means that `x` is declared on line 37, implicitly assigned through a `solve` statement on line 50 and referenced on lines 39, 45, 46, 47, and 51. The entry `2*47` means that there are two references to `x` on line 47 of the input file.

The complete list of reference types and their shorthand symbols in GAMS Output is given below:

Shorthand Symbol	Reference Type	Description
`declared`	Declaration	The identifier is declared as to type. This must be the first appearance of the identifier.
`defined`	Definition	An initialization (for a table or a data list between slashes) or symbolic definition (for an equation) starts for the identifier.
`assigned`	Assignment	Values are replaced because the identifier appears on the left-hand side of an assignment statement.
`impl-asn`	Implicit Assignment	An equation or variable will be updated as a result of being referred to implicitly in a solve statement.
`control`	Control	A set is used as (part of) the driving index in an assignment, equation, loop or indexed operation.
`ref`	Reference	The symbol has been referenced on the right-hand side of an assignment or in a display, equation, model, solve statement or put statetement.
`index`	Index	Like `control`, but used only for set *labels*. Appears only in the cross reference map of unique elements. See section The Unique Element Listing Map for details.

Note that the symbol reference map may be useful in model development, documentation preparation, and to make sure that all declared items are actually used in the model (by checking that they all have reference symbols in addition to `declared` and `defined`).

4.38.3.3 The Symbol Listing Map

The second optional map is the *symbol listing*. In the symbol listing map all identifiers are grouped alphabetically by data type and listed with their explanatory texts. This is another very useful aid for trying to understand a large model prepared by someone else. Note that expressive explanatory text is particularly helpful here. The symbol listing map can be turned on by entering a line containing the dollar control option $onSymList at the beginning of the program. The symbol listing map generated from our Illustrative Model follows:

```
SETS

i             securities
j             Aliased with i

PARAMETERS

highrisk      variance of highest security risk
lowyield      yield of lowest yielding security
mean          mean annual returns on individual securities (%)
target        target mean annual return on portfolio %
v             variance-covariance array (%-squared annual return)

VARIABLES

variance      variance of portfolio
x             fraction of portfolio invested in asset i

EQUATIONS

dmean         definition of mean return on portfolio
```

```
dvar          definition of variance
fsum          fractions must add to 1.0

MODELS

portfolio
```

4.38.3.4 The Unique Element Listing Map

The last optional map is the *Unique Element* (UEL) Listing. The unique element listing can be turned on by entering a line containing the dollar control option $onUELList at the beginning of the program. All unique elements are first grouped in entry order and then in sorted order with their explanatory texts:

```
Unique Element Listing


Unique Elements in Entry Order

1  hardware    software    show-biz    t-bills

Unique Elements in Sorted Order

1  hardware    show-biz    software    t-bills
```

In addition, a map with references for every label is given if the dollar control option $onUELXRef appears at the beginning of the program.

```
ELEMENT   REFERENCES

hardware  declared       13      ref       21       27       28
show-biz  declared       13      ref       23       27       30
software  declared       13      ref       22       27       29
t-bills   declared       13      ref       24       27       31
```

The unique element list is important in the context of ordered sets. For details see section Ordered and Unordered Sets.

4.38.3.5 The Include File Summary

Often the GAMS program includes a text file specified by the dollar control option $include. Consider the following example:

```
$include 'file1.inc'
Set i / i1*i10 /;
Scalar a; a = 7;
$include 'file2.inc'
```

The file `file1.inc` contains the following two lines:

```
Set j  / j1*j5 /;
Parameter k(j);
```

The file `file2.inc` contains the following line:

```
a = a+3;
```

The echo print follows:

```
INCLUDE    C:\models\file1.inc
   2  Set j  / j1*j5 /;
   3  Parameter k(j);
   4  Set i / i1*i10 /;
   5  Scalar a; a = 7;
INCLUDE    C:\models\file2.inc
   7  a = a+3;
```

Note that the contents of the two include files are expanded in the echo print and the include file names and their path is echoed in the lines with the dollar control option $include. If we insert the dollar control option $offInclude in the first line of the code, the echo print will not include the lines with the names of the include files any more, but the contents will still be echoed:

```
   3  Set j  / j1*j5 /;
   4  Parameter k(j);
   5  Set i / i1*i10 /;
   6  Scalar a; a = 7;
   8  a = a+3;
```

If the program contains include files, there will be an `Include File Summary` at the end of the compilation output. The include file summary of the simple example above (without the line $offInclude) follows. If `$offInclude` is active, it will be suppressed.

```
Include File Summary

   SEQ   GLOBAL TYPE      PARENT   LOCAL  FILENAME

     1        1 INPUT          0       0  C:\models\test.gms
     2        1 INCLUDE        1       1  .C:\models\file1.inc
     3        6 INCLUDE        1       4  .C:\models\file2.inc
```

In the first column the sequence number of the input files is given. Note that the parent file called by the GAMS call is always listed first. The second column gives the global (expanded) line number which contains the respective option `include` or one of its variants. The third column indicates the type of the respective file. An overview of the file types is given below:

File Type	Description
INPUT	The GAMS input file that was called with the GAMS call.
EXIT	Exit from compilation
INCLUDE	A file that was inserted with the dollar command option $include.
BATINCLUDE	A file that was inserted with the dollar command option $batinclude.
LIBINCLUDE	A file that was inserted with the dollar command option $libinclude.

File Type	Description
SYSINCLUDE	A file that was inserted with the dollar command option $sysinclude.
CALL	A program that was called with the dollar command option $call
CALL.ASYNC	A program that was called asynchronously with the dollar command option $call.Async
CALLTOOL	A GAMS tool was called with the dollar command option $callTool
GDXIN	A GDX file that was opened for input with the dollar command option $gdxIn
GDXOUT	A GDX file that was opened for output with the dollar command option $gdxOut
IF EXIST	Operator used to test whether a given file name exists
IF DEXIST	Operator used to test whether a given directory name exists
FUNCLIBIN	A file that was inserted with the dollar command option $funcLibIn
TERMINATE	Terminate compilation and execution
STOP	Stop compilation

The next column, named PARENT, provides the sequence number of the respective parent file. The column with the header LOCAL gives the local line number in the parent file where the file was included. The last column shows the path and the name of the respective file.

Note that compilation error messages in include files have additional information about the name of the include file and the local line number. For information on compilation errors, see section Compilation Errors below.

4.38.4 Execution Output

The execution output follows the compilation output in the GAMS listing file. It is introduced with the title Execution. The execution output is the output generated while GAMS is performing data manipulations and results from display statements. The output from the display statement on line 33 of our Illustrative Model is shown below.

```
E x e c u t i o n


----     33 PARAMETER lowyield             =        7.000  yield of lowest yielding security
            PARAMETER highrisk             =       10.000  variance of highest security risk
```

If errors are detected because of illegal data operations, a brief message indicating the cause and the line number of the offending statement will appear. For further information on execution errors, see section Execution Errors below.

Note

In case there is no display statement before The Solve Statement in the model and there are no execution errors, there will be no execution output in the GAMS listing file.

4.38.5 Model Generation Output

The model generation output, the solution report, and the post-solution Output are produced when a solve statement is executed. The actions that are initiated by a solve statement are presented in section Actions Triggered by the Solve Statement. All output generated as a result of a solve is labeled with a subtitle identifying the model, its type and the line number of the solve statement.

The output generated during model generation includes equations listing, variable listing, and model statistics.

4.38.5.1 The Equation Listing

The *equation listing* is the first part of the output generated by a solve statement. It is marked with the subtitle `Equation Listing`. By default, the first three equations in every block are listed. This can be modified with the option limrow. If there are three or fewer single equations in any equation block, then all the single equations are listed.

Note that by studying the equation listing the user may determine whether the model generated by GAMS is the the model that the user has intended - an extremely important question. This component of the output shows the specific equations generated within the model when the current values of the sets and parameters are plugged into the general algebraic form of the model.

The equation listing from our Illustrative Model is given below. In our case we have three equation blocks, each producing one single equation.

```
A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
Equation Listing    SOLVE portfolio Using NLP From line 50


---- fsum  =E=  fractions must add to 1.0

fsum..  x(hardware) + x(software) + x(show-biz) + x(t-bills) =E= 1 ; (LHS = 0, INFES = 1 ****)


---- dmean  =E=  definition of mean return on portfolio

dmean..  8*x(hardware) + 9*x(software) + 12*x(show-biz) + 7*x(t-bills) =E= 10 ; (LHS = 0, INFES = 10


---- dvar  =E=  definition of variance

dvar..  (0)*x(hardware) + (0)*x(software) + (0)*x(show-biz) - variance =E= 0 ; (LHS = 0)
```

Note

> The equation listing is an extremely useful debugging aid. It shows the variables that appear in each constraint. In addition, it shows what the individual coefficients and right-hand side values evaluate to after the data manipulations have been done.

Each equation block is marked with four dashes which are useful for mechanical searching. The name, type of equation and explanatory text is shown, followed by the individual equations.

Note

> All the terms that depend on variables are collected on the left-hand side and all the constant terms are combined into one number on the right-hand side, any necessary sign changes being made.

Four places of decimals are shown if necessary, but trailing zeroes following the decimal point are suppressed. E-format is used to prevent small numbers to be displayed as zeros.

Note

Nonlinear equations are treated differently. If the coefficient of a variable in the equation listing is enclosed in parentheses, then the corresponding constraint is nonlinear, and the value of the coefficient depends on the activity levels of one or more of the variables. The listing is not algebraic, but shows the partial derivative of each variable evaluated at their current level values.

Note that in the equation listing from our example the equation `dvar` is nonlinear. A simpler example will help to clarify the point. Consider the following equation and associated level values.

```
eq1.. 2*sqr(x)*power(y,3) + 5*x - 1.5/y =e= 2; x.l = 2; y.l = 3 ;
```

This equation will appear in the equation listing as:

```
eq1.. (221)*x + (216.1667)*y  =E=  2 ; (LHS = 225.5 ***)
```

The coefficient of x is determined by first differentiating the equation above with respect to x. This results in $2 \times (2 \times x.l) \times (y.l)^3 + 5$, which evaluates to 221, given that $x.l$ equals 2 and $y.l$ equals 3. Similarly, the coefficient of y is obtained by differentiating the equation above with respect to `y`, which results in $2 \times (x.l)^2 \times 3 \times (y.l)^2 + \frac{1.5}{(y.l)^2}$, giving 216.1667. Observe that the coefficient of y could not have been determined if its level had been left at zero. The attempted division by zero would have produced an error and premature termination. For further information on modeling NLP problems with GAMS, see the tutorial Good NLP Formulations.

The result of evaluating the left-hand side of the equation at the initial point is shown at the end of each individual equation listing. In the example above it is 225.5, and the three asterisks(***) are a warning that the constraint is infeasible at the starting point.

Note

The order in which the equations are listed depends on how the model was defined. If it was defined with a list of equation names, then the listing will be in the order of that list. If it was defined as `/all/`, then the list will be in the order of declaration of the equations. The order of the entries for the individual constraints is determined by the label entry order.

4.38.5.2 The Column Listing

The *column listing* or *variable listing* is the next part of the output. It is marked with the title `Column Listing` and contains a list of the individual coefficients sorted by column rather than by row (like in the equation listing). By default the first three entries for each variable are shown, along with their lower bound `.lo`, upper bound `.up` and current level values `.l`. Note that the default number of entries shown may be modified with the option limcol.

The format for the coefficients is exactly as in the equation listing, with the nonlinear coefficients enclosed in parentheses and the trailing zeroes dropped. The column listing from our Illustrative Model follows.

```
A Quadratic Programming Model for Portfolio Analysis (ALAN,SEQ=124a)
Column Listing      SOLVE portfolio Using NLP From line 50


---- x  fraction of portfolio invested in asset i

x(hardware)
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
        1       fsum
        8       dmean
       (0)      dvar

x(software)
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
        1       fsum
        9       dmean
       (0)      dvar

x(show-biz)
                (.LO, .L, .UP, .M = 0, 0, +INF, 0)
        1       fsum
       12       dmean
       (0)      dvar

REMAINING ENTRY SKIPPED

---- variance  variance of portfolio

variance
                (.LO, .L, .UP, .M = -INF, 0, +INF, 0)
       -1       dvar
```

Note

The order in which the variables appear is the order in which they were declared.

4.38.5.3 The Range Statistics

This block shows a statistic about the range of values seen in the model. Namely, it prints the range of absolute non-zero values for the bounds, right hand sides and matrix coefficients. Zero values are marked explicitly. This is the example output when running the model [**TRNSPORT**]:

```
Range Statistics    SOLVE transport Using LP From line 71


RANGE STATISTICS (ABSOLUTE NON-ZERO FINITE VALUES)

RHS       [min, max] : [ 2.750E+02, 6.000E+02] - Zero values observed as well
Bound     [min, max] : [        NA,        NA] - Zero values observed as well
Matrix    [min, max] : [ 1.260E-01, 1.000E+00]
```

Note the `NA` range for the bounds: For this model, the bounds are either `0` or `+INF`. Both values are excluded in this range calculation. Therefore we see the `NA` here.

4.38.5.4 The Model Statistics

The final information generated while a model is being prepared for solution is the statistics block. It is marked with the title `Model Statistics`. Its most obvious use is to provide details on the size and nonlinearity of the model. The model statistics of our Illustrative Model follow:

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Model Statistics    SOLVE portfolio Using NLP From line 50


MODEL STATISTICS

BLOCKS OF EQUATIONS           3     SINGLE EQUATIONS            3
BLOCKS OF VARIABLES           2     SINGLE VARIABLES            5
NON ZERO ELEMENTS            12     NON LINEAR N-Z              3
CODE LENGTH                  25     CONSTANT POOL              17


GENERATION TIME      =        0.004 SECONDS      4 MB


EXECUTION TIME       =        0.005 SECONDS      4 MB
```

The `BLOCK` counts indicate the number of GAMS symbols (`equations` and `variables`, respectively) appearing in the problem generated. The `SINGLE` counts indicate the number of individual rows and columns in the problem generated. The `NON ZERO ELEMENTS` entry refers to the number of nonzero coefficients in the problem matrix. To understand the remaining elements, some background will be useful.

For linear models, the model can be completely specified by the problem matrix and vectors for the RHS, etc. More is needed for noninear models. GAMS passes the nonlinear expressions defining the model algebra to the solvers in the form of *NL code* - opcode/address pairs in a simple stack-based language. For example, the expression `exp(x*y-1.675)` results in code similar to this:

```
PUSHV  column-index-of-x
MULV   column-index-of-y
SUBC   index-of-1.675
FUNC1  index-of-exp-function
STOR   index-of-row
```

In addition to containing indices refering to the rows and columns of the model, the NL code contains indices of constants used to define the model algebra. These constants are not contained in the NL code directly but rather in a separate pool or list of constants used for the model.

There are three entries that provide additional information about nonlinear models. The `NON LINEAR N-Z` entry refers to the number of nonlinear matrix entries in the model. Nonlinear models may differ in the complexity or size of the nonlinear expressions defining them. For example, `x*y` is a simpler form of nonlinearity than `exp(x*y)` and will require a shorter list of instructions to define, even though both of these terms result in two nonlinear entries in the matrix (one for x and one for y). The `CODE LENGTH` entry indicates the length of the NL code (i.e. number of instuctions) required to define the model algebra, while the `CONSTANT POOL` indicates the length of the constant pool that is used by the NL code. In general, the more nonlinear a problem is, the more difficult it tends to be to solve.

The times that follow statistics are also useful. The `GENERATION TIME` is the time used since compilation (syntax check) is finished. This includes the time spent in generating the model. The measurement units are given and represent ordinary clock time on personal computers or central processor usage (CPU) time on other machines. Memory use is given in megabytes.

4.38.6 The Solution Report

The *solution report* is the next part of the output. It is marked with the title `Solution Report` and includes the solve summary, the solver report, the solution listing, and the report summary.

4.38.6.1 The Solve Summary

The *solve summary* contains details about the solution process and is marked with the title `SOLVE SUMMARY`. The first part of the solve summary is common to all solvers and is discussed in this subsection. The second part of the solve summary is solver specific, it is covered in the subsection Solver Report below. The first part of the solve summary for our Illustrative Model follows.

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Solution Report     SOLVE portfolio Using NLP From line 50


               S O L V E      S U M M A R Y

     MODEL   portfolio           OBJECTIVE  variance
     TYPE    NLP                 DIRECTION  MINIMIZE
     SOLVER  MINOS               FROM LINE  50

**** SOLVER STATUS     1 Normal Completion
**** MODEL STATUS      2 Locally Optimal
**** OBJECTIVE VALUE                2.8990

 RESOURCE USAGE, LIMIT          0.188      1000.000
 ITERATION COUNT, LIMIT         5    2000000000
 EVALUATION ERRORS              0             0
```

The entry `MODEL` contains the name of the model being solved, `TYPE` provides the model type of the model, `SOLVER` shows the name of the solver used to solve the model, `OBJECTIVE` gives the name of the objective variable being optimized, `DIRECTION` shows the direction of optimization being performed and the entry `FROM LINE` provides the line number of the solve statement the solve summary refers to.

Note that the four asterisks make it easy to find the solve summary if it is searched for mechanically.

The entries `SOLVER STATUS` and `MODEL STATUS` contain the solver and model status for the problem respectively. Their possible values are given in Model Status table and Solver Status tables below. The entry `OBJECTIVE VALUE` provides the value of the objective function at the termination of the solve. This value is the optimum value for the problem provided the solver and model have the right status.

The entry `RESOURCE USAGE, LIMIT` reports the amount of wall clock time (in seconds) taken by the solver and the upper limit allowed for the solver. Note that the solver will stop as soon as the limit on time usage has been reached. The default limit may be changed with the option reslim. Observe that the option statement `option reslim = x;` - where `x` is the desired limit in wall clock seconds - must be entered in the model *before* the solve statement.

The entry `ITERATION COUNT, LIMIT` provides the number of iterations used by the solver and the upper limit allowed for the solver. The solver will stop as soon as this limit is reached. The default limit on iterations used is practically infinity. This limit may be changed with the option iterlim. Observe that the option statement `option iterlim = n;` - where `n` is the desired limit of the iterations used - must be entered in the model *before* the solve statement.

Finally, the entry `EVALUATION ERRORS` reports the number of numerical errors encountered by the solver and the upper limit allowed for the solver. These errors result from numerical problems like division by zero. Note that this is suppressed for LP, RMIP and MIP models since evaluation errors are not applicable for these model types. The default limit on evaluation errors used is zero. This limit may be changed with the option domlim. Observe that the option statement `option domlim = n;` - where `n` is the desired limit of the evaluation errors allowed - must be entered in the model *before* the solve statement.

Model Status

Value	Message	Description
1	OPTIMAL	The solution is optimal, that is, it is feasible (within tolerances) and it has been proven that no other feasible solution with better objective value exists. Note that the latter criterion is not influenced by the optcr and optca options.
2	LOCALLY OPTIMAL	A local optimum for an NLP has been found. That is, a solution that is feasible (within tolerances) and it has been proven that there exists a neighborhood of this solution in which no other feasible solution with better objective value exists.
3	UNBOUNDED	The solution is unbounded. This message is reliable if the problem is linear, but occasionally it appears for difficult nonlinear problems that are not truly unbounded, but that lack some strategically placed bounds to limit the variables to sensible values.
4	INFEASIBLE	The problem has been proven to be infeasible. If this was not intended, something is probably misspecified in the logic or the data.
5	LOCALLY INFEASIBLE	No feasible point could be found for the NLP problem from the given starting point. It does not necessarily mean that no feasible point exists.
6	INTERMEDIATE INFEASIBLE	The current solution is not feasible, but the solver stopped, either because of a limit (for example, iteration or resource) or because of some sort of difficulty. The solver status will give more information.
7	FEASIBLE SOLUTION	A feasible solution to a problem without discrete variables has been found.
8	INTEGER SOLUTION	A feasible solution to a problem with discrete variables has been found. There is more detail following about whether this solution satisfies the termination criteria (set by options optcr and optca).
9	INTERMEDIATE NON-INTEGER	An incomplete solution to a problem with discrete variables. A feasible solution has not yet been found. See section Model Termination Conditions for MIPs for more information.
10	INTEGER INFEASIBLE	It has been proven that there is no feasible solution to a problem with discrete variables. See section Model Termination Conditions for MIPs for more information.
11	LIC PROBLEM - NO SOLUTION	The solver cannot find the appropriate license key needed to use a specific subsolver.
12	ERROR UNKNOWN	After a solver error the model status is unknown.
13	ERROR NO SOLUTION	An error occurred and no solution has been returned. No solution will be returned to GAMS because of errors in the solution process.
14	NO SOLUTION RETURNED	A solution is not expected for this solve. For example, the CONVERT solver only reformats the model but does not give a solution.

Value	Message	Description
15	SOLVED UNIQUE	Indicates the solution returned is unique, i.e. no other solution exists. Used for CNS models. Examples where this status could be returned include non-singular linear models, triangular models with constant non-zero elements on the diagonal, and triangular models where the functions are monotone in the variable on the diagonal.
16	SOLVED	Indicates the model has been solved: used for CNS models. The solution might or might not be unique. If the solver uses status 17 - SOLVED SINGULAR wherever possible then this status implies that the Jacobian is non-singular, i.e. that the solution is at least locally unique.
17	SOLVED SINGULAR	Indicates the CNS model has been solved, but the Jacobian is singular at the solution. This can indicate that other solutions exist, either along a line (for linear models) or a curve (for nonlinear models) including the solution returned.
18	UNBOUNDED - NO SOLUTION	The model is unbounded and no solution can be provided.
19	INFEASIBLE - NO SOLUTION	The model is infeasible and no solution can be provided.

Note that the model status is stored in the model attribute modelStat. For details on model attributes, see section Model Attributes. Observe that there are compile-time constants that are related to this model attribute.

Solver Status

Value	Message	Description
1	NORMAL COMPLETION	The solver terminated in a normal way: it was not interrupted by a limit (resource, iterations, nodes or other) or by internal difficulties. The model status describes the characteristics of the accompanying solution.
2	ITERATION INTERRUPT	The solver was interrupted because it used too many iterations. The option iterlim may be used to increase the iteration limit if everything seems normal.
3	RESOURCE INTERRUPT	The solver was interrupted because it used too much time. The option reslim may be used to increase the time limit if everything seems normal.
4	TERMINATED BY SOLVER	The solver encountered some difficulty and was unable to continue. More details will appear following the message.
5	EVALUATION INTERRUPT	Too many evaluations of nonlinear terms at undefined values. We recommend to use variable bounds to prevent forbidden operations, such as division by zero. The rows in which the errors occur are listed just before the solution.
6	CAPABILITY PROBLEMS	The solver does not have the capability required by the model. For example, some solvers do not support certain types of discrete variables or support a more limited set of functions than other solvers.
7	LICENSING PROBLEMS	The solver cannot find the appropriate license key needed to use a specific subsolver.
8	USER INTERRUPT	The user has sent a message to interrupt the solver via the interrupt button in the IDE or sending a Control+C from a command line.

Value	Message	Description
9	ERROR SETUP FAILURE	The solver encountered a fatal failure during problem set-up time.
10	ERROR SOLVER FAILURE	The solver encountered a fatal error.
11	ERROR INTERNAL SOLVER FAILURE	The solver encountered an internal fatal error.
12	SOLVE PROCESSING SKIPPED	The entire solve step has been skipped. This happens if execution errors were encountered and the GAMS parameter ExecErr has been set to a nonzero value or the property MaxExecError has a nonzero value.
13	ERROR SYSTEM FAILURE	This indicates a completely unknown or unexpected error condition.

Note that the solver status is stored in the model attribute solveStat. For details on model attributes, see section Model Attributes. Observe that there are compile-time constants that are related to this model attribute.

4.38.6.2 Solver Report

The next section in the listing file is the part of the solve summary that is particular to the solver program that has been used. This section normally begins with a message identifying the solver and its authors: MINOS and QUADMINOS was used in the example here. There will also be diagnostic messages in plain language if anything unusual was detected and specific performance details, some of them probably technical. The solver manual of the respective solver will help explain these. In case of serious trouble, the GAMS listing file will contain additional messages printed by the solver. This may help identify the cause of the difficulty. If the solver messages do not help, a perusal of the solver documentation or help from a more experienced user is recommended. The solver report from our Illustrative Model follows.

```
GAMS/MINOS
M I N O S  5.6      (Nov 2014)

    GAMS/MINOS 5.6, Large Scale Nonlinear Solver
    B. A. Murtagh, University of New South Wales
    P. E. Gill, University of California at San Diego,
    W. Murray, M. A. Saunders, and M. H. Wright,
    Systems Optimization Laboratory, Stanford University

 Work space allocated           --     0.77 Mb

 EXIT - Optimal Solution found, objective:        2.899038
```

Note that the line `Work space allocated -- 0.77 MB` provides the amount of memory used by the solver for the problem. The solver estimates the amount of memory it will need to solve the problem. If this amount is not available on the machine used, GAMS will return a message saying that not enough memory was allocated. In addition, GAMS will return the maximum amount of memory available on the machine. The user may direct the solver to use less memory with the model attribute `.workspace`:

```
mymodel.workspace = x;
```

Here `mymodel` is the name of the model being solved as specified by the model statement and `x` is the amount of memory in Megabytes. Note that the solver will attempt to solve the problem with `x` MB of memory. However, it is not guaranteed to succeed since the problem may require more memory.

Note that more information for a successful run may be obtained using the option sysout. As usual, the respective option statement should be placed *before* the solve statement.

4.38.6.3 The Solution Listing

The solution listing is a row-by-row then column-by-column listing of the solutions returned to GAMS by the solver program. Each individual equation and variable is listed with four pieces of information. The solution listing may be suppressed with the option solPrint:

```
option solprint = off ;
```

This option statement should be placed *before* the solve statement. The solution listing generated from our Illustrative Model is shown below.

```
                            LOWER          LEVEL          UPPER         MARGINAL

---- EQU fsum                1.0000         1.0000         1.0000       -13.5288
---- EQU dmean              10.0000        10.0000        10.0000         1.9327
---- EQU dvar                  .              .              .           -1.0000

  fsum  fractions must add to 1.0
  dmean  definition of mean return on portfolio
  dvar  definition of variance

---- VAR x  fraction of portfolio invested in asset i

                LOWER          LEVEL          UPPER         MARGINAL

hardware           .            0.3029         +INF             .
software           .            0.0865         +INF     6.217249E-15
show-biz           .            0.5048         +INF             .
t-bills            .            0.1058         +INF           EPS

                            LOWER          LEVEL          UPPER         MARGINAL

---- VAR variance          -INF            2.8990         +INF             .

  variance  variance of portfolio
```

The order of the equations and variables is the same as in the symbol listing map.

The four columns associated with each entry correspond to the equation and variable attributes and have the following meaning:

`LOWER` lower bound (`.lo`)

`LEVEL` level value (`.l`)

`UPPER` upper bound (`.up`)

`MARGINAL` marginal (`.m`)

For variables, the values in the `LOWER` and `UPPER` columns refer to the lower and upper bounds. For equations, they are obtained from the (constant) right-hand side value and from the relational type of the equation. For details see section Variable Attributes and Equation Attributes. Note that instead of `level` values, `slack` values may be shown in equations. For more information, see section Customizing the Output File below.

Note

The LEVEL and MARGINAL values have been determined by the solver and the values shown are used to update the GAMS values. In the list they are shown with fixed precision, but the values are returned to GAMS with full machine accuracy. The single dots '.' on the list represent zeros.

EPS is the GAMS extended value used for a stored zero. It is common to see a marginal value given as EPS, since GAMS uses the convention that marginals are necessarily zero for basic variables and typically non-zero for other variables.

Note

EPS is used to indicate non-basic variables whose marginal values are at or close to zero, and for nonlinear problems to indicate superbasic variables whose marginals are at or close to zero. A superbasic variable is one between its bounds at the final point but not in the basis.

Note that in the Glossary there are brief explanations of technical terms that were used in this section.

For models that are not solved to optimality or for models types without an objective, some constraints may additionally be marked with certain flags. The list of these flags and their description is given in the following table.

Shorthand Symbol	Description
INFES	The row or column is **infeasible**. This mark is made for any entry where the level value is not between the upper and lower bounds.
NOPT	The row or column is **non-optimal**. This mark is made for any non-basic entries for which the marginal sign is incorrect, or superbasic ones for which the marginal value is too large.
UNBND	The row or column can increase without limit (e.g. in an unbounded ray) or has an excessively large magnitude.
REDEF	The equation type has been ignored or redefined for this solution point, e.g. because this is an MCP
REDIR	The equation has been flipped (i.e. negated or reoriented) in the model statement: useful for MCP and EMP
DEPND	The row or column is dependent on other rows or columns in the system: see CNS

4.38.6.4 Report Summary

The final section of the solution report is the report summary, marked with four asterisks (as are all important components of the output) followed by the title REPORT SUMMARY. It shows the count of rows or columns that have been marked INFES, NOPT or UNBND in the solution listing. For model types like MCP where REDEFs and REDIRs are possible, the counts for these markers may also be shown. The sum of infeasibilities will be shown if the reported solution is infeasible. The domain error count is only shown if the problem is nonlinear. If there are variables or equations where the levels were projected to one of the bounds, the count of those is also shown here.

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
Solution Report     SOLVE portfolio Using NLP From line 50


**** REPORT SUMMARY :        0     NONOPT
```

```
    0 INFEASIBLE
    0  UNBOUNDED
    0     ERRORS
   42  PROJECTED
```

4.38.7 Post-Solution Output

The final part of the listing file is the post-solution output. It contains Final Execution Summary and the File Summary.

4.38.7.1 Final Execution Summary

The final execution summary is marked with the title `Execution` and contains the output from display statements that were placed *after* the solve statement in the model, allowing simple reporting. In addition, it shows the final exection time and memory use. The respective ouput from our Illustrive Model follows.

```
A Quadratic Programming Model for Portfolio Analysis ALAN,SEQ=124a)
E x e c u t i o n


----     51 VARIABLE x.L  fraction of portfolio invested in asset i

hardware 0.303,    software 0.087,    show-biz 0.505,    t-bills  0.106

----     51 VARIABLE variance.L            =        2.899  variance of portfolio

EXECUTION TIME       =        0.001 SECONDS      3 MB
```

For further information on this output and ways to customize it, see chapter The Display Statement.

4.38.7.2 File Summary

The file summary is the very last part of the output file. If output has been written to put files, there will be a `REPORT FILE SUMMARY` that is marked with four asterisks. The report will list the put files with their internal names and the full paths of their external names.

All listing files have a `FILE SUMMARY` that is marked with four asterisks and that reports the names of the input and output (listing) files.

```
**** FILE SUMMARY

Input      C:\PROGRAM FILES\\gamsIDE\ALAN.GMS
Output     C:\PROGRAM FILES\\gamsIDE\ALAN.LST
```

If work files (save or restart) have been used, they will be listed in the file summary as well.

4.38.8 Error Reporting

All comments and descriptions about errors have been collected in this section for easy reference when disaster strikes.

Effective error detection and recovery are important parts of any modeling system. GAMS is designed around the assumption that the *error state* is the normal state of modeling. Experience shows that most compilations during the early stages of development will produce errors. Not to worry! The computer is much better at checking details than the human mind and should be able to provide positive feedback and suggestions about how to correct errors or avoid ambiguities. Developing a model is like writing a paper or an essay; many drafts and rewrites are required until the arguments are presented in the most effective way and meet all the requirements of proper English. GAMS acts like a personal assistant with knowledge of mathematical modeling and of the syntactic and semantic details of the language.

Errors are detected at various stages in the modeling process. Most of them are caught at the compilation stage, which behaves like the proofreading stage of the modeling process. Once a problem has passed through the rigorous test of this stage, the error rate drops to almost zero. Most of the execution runs, which are much more expensive than compilation, proceed without difficulties because GAMS *knows* about modeling and has anticipated problems. Many of the typical errors made with conventional programming languages are associated with concepts that do not exist in GAMS. Those error sources – like address calculations, storage assignment, subroutine linkages, input-output and flow control – create problems at execution time, are difficult to locate, often lead to long and frustrating searches, and leave the computer user intimidated. GAMS takes a radically different approach. Errors are spotted as early as possible, they are reported in a way that is comprehensible to the user, including clear suggestions for how to correct the problem and a presentation of the source of the error in terms of the problem of the user.

Note

All errors are marked with four asterisks '****' at the beginning of a line in the output listing.

As soon as an error is detected, processing will be stopped at the next convenient opportunity. A model will never be solved after an error has been detected. The only remedy is to fix the error and repeat the run.

Errors are grouped into the three phases of GAMS modeling: compilation, execution and model generation (which includes the solution that follows). In the following subsections we will discuss the errors that may occur in each of these phases.

4.38.8.1 Compilation Errors

Compilation errors are discussed in some detail in the tutorial A GAMS Tutorial by Richard E. Rosenthal and the tutorial Fixing Compilation Errors. Note that there is some overlap between the material in the tutorials and this section. Several hundred different types of errors can be detected during compilation and can often be traced back to just one specific symbol in the GAMS input. Most of the errors are caused by simple mistakes: forgetting to declare an identifier, putting indices in the wrong order, leaving out a necessary semicolon or misspelling a label. For errors that are not caused by such simple mistakes, the explanatory error message text will help diagnose the problem and correct it.

Note

When a compilation error is discovered, a dollar symbol and error number are printed below the offending symbol (usually to the right) on a separate line that begins with four asterisks.

If more than one error is encountered on a line (possibly because the first error caused a series of other spurious errors) the dollar signs may be suppressed and error number squeezed. GAMS will not list more than 10 errors on any one line.

Note

A list of all error numbers encountered and a description of the probable cause of each error is printed at the end of the echo print of the program. The error messages are self-explanatory. For further information on compilation errors and advice on how to resolve them, see the tutorial Fixing Compilation Errors.

It is worth noting that it is easy to produce a model that does not do what you want it to do, but does not contain errors in the sense that the term is being used in this section. The best precaution is to check your work carefully and build in as many automatic consistency checks as possible.

Note that if a GAMS reserved word is accidentally used for the name of a label or an identifier it is impossible to provide helpful error messages for technical reasons. This may cause confusion. We recommend users to familiarize themselves with the reserved words.

Attention

In some cases an error may not be detected until the statement following its occurrence, where it may produce a number of error conditions whose explanations seem quite silly. We recommend to always check carefully for the cause of the first error in such a group. If nothing obvious is wrong, look at the previous statement and particularly watch out for missing semicolons.

The following example illustrates the general reporting format for compilation errors.

```
   1  Set c crops / wheat, corn, wheat, longaname /
****                                  $172
   2  Parameter price(c) / wheat 200, cotton 700 /
****                                       $170
   3
Error Messages

170  Domain violation for element
172  Element is redefined

**** 2 ERROR(S)   0 WARNING(S)
..
**** USER ERROR(S) ENCOUNTERED
```

4.38.8.2 Compilation Time Errors

The reporting format for errors found while analyzing solve statements is more complicated than for normal compilation errors, mainly because many things must be checked. All identifiers referenced must be defined or assigned, the mathematics in the equations must match the model class, and so on. More elaborate reporting is required to accurately describe any problems found. The solve statement is only checked if the model has been found to be error free up to this point. This is not only because the check is comparatively expensive, but also because many erroneous and confusing messages can be produced while checking a solve in a program containing other errors.

Attention

Compiler error messages related to a solve statement are reported in two places and in two formats:

1. They are shown immediately after the solve statement with a short text including the name of any offending identifier and the type of model involved. This will be sufficient in most cases.
2. A longer message with some hints appears with the other error messages at the end of the compilation.

The example below illustrates the general reporting format for compiler errors associated with a solve statement.

```
   1  Variables x, y, z ;
   2  Equations eq1 , eq2 ;
   3
   4  eq1.. x**2 - y =e= z ;
   5  eq2.. min(x,y) =l= 20 ;
   6
   7  Model silly / all / ;
   8  solve silly using lp maximizing z ;
****                                  $54,51,256
**** THE FOLLOWING LP ERRORS WERE DETECTED IN MODEL SILLY:
**** 54 IN EQUATION EQ1         .. ENDOG OPERANDS FOR **
**** 51 IN EQUATION EQ2         .. ENDOG ARGUMENT(S) IN FUNCTION
   9

Error Messages

 51  Endogenous function argument(s) not allowed in linear models
 54  Endogenous operands for ** not allowed in linear models
256  Error(s) in analyzing solve statement. More detail appears
     Below the solve statement above

**** 3 ERROR(S)   0 WARNING(S)
**** USER ERROR(S) ENCOUNTERED
```

4.38.8.3 Execution Errors

Execution time errors are usually caused by illegal arithmetic operations such as division by zero or taking the log of a negative number. GAMS prints a message on the output file with the line number of the offending statement and continues execution. A GAMS program should never abort with an unintelligible message from the computer's operating system if an invalid operation is attempted. GAMS has rigorously defined an extended algebra that contains all operations including illegal ones. Note that the model **[CRAZY]** contains all non-standard operations and should be executed to study for its exceptions. For advice on detecting and resolving execution errors, see the tutorial Finding and Fixing Execution Errors and Performance Problems.

Recall that the GAMS arithmetic is defined over the closed interval [-INF,+INF] and contains the extendedn range arithmetic values EPS (small but not zero), NA (not available) and UNDF (the result of an illegal operation). The results of illegal operations are propagated through the entire system and can be displayed with standard display statements. However, observe that, if errors have been previously detected, a model cannot be solved and a work file cannot be saved.

4.38.8.4 Solve Errors

The execution of a `solve` statement can trigger additional errors called *matrix errors*. They report on problems encountered during transformation of the model into a format required by the solver. Problems are most often caused by illegal or inconsistent bounds, or by an extended range value that is used as a matrix coefficient. The example below shows the general format of these errors:

```
   1  Variable x;
   2  Equation eq1;
   3
   4  eq1.. x =l= 10 ;
   5  x.lo = 10 ;
   6  x.up = 5 ;
   7  Model wrong /eq1/;
   8  solve wrong using lp maximizing x ;
   9

**** Matrix error - lower bound > upper bound
x   (.LO, .L, .UP = 10, 0, 5)
...
**** SOLVE from line 8 ABORTED, EXECERROR = 1
...
**** USER ERROR(S) ENCOUNTERED
```

Some `solve` statements require the evaluation of nonlinear functions and the computation of derivatives. Since these calculations are not carried out by GAMS but by other subsystems not under the direct control of GAMS, errors associated with these calculations are reported in the solution report. Note that by default the subsystems will interrupt the solution process if arithmetic exceptions are encountered. This may be changed with the option domlim. They are then reported on the listing as shown in the following example:

```
   1  Variable x, y;
   2  Equation one;
   3
   4  one.. y =e= sqrt(10/x);
   5  x.l = 10;
   6  x.lo = 0;
   7
   8  Model divide /  all / ;
   9  solve divide maximizing y using nlp;

               S O L V E      S U M M A R Y

     MODEL   divide              OBJECTIVE  y
     TYPE    NLP                 DIRECTION  MAXIMIZE
     SOLVER  MINOS               FROM LINE  9

**** SOLVER STATUS     5 Evaluation Interrupt
**** MODEL STATUS      7 Feasible Solution
**** OBJECTIVE VALUE    3.1622776602E+0149

 RESOURCE USAGE, LIMIT          0.182      1000.000
 ITERATION COUNT, LIMIT         0    2000000000
 EVALUATION ERRORS              2             0

EXIT - Function evaluation error limit exceeded.
```

```
**** ERRORS/WARNINGS IN EQUATION one
     2 error(s): div: FUNC SINGULAR: x/y, |y| <= 1e-150 (RETURNED 1E299)

**** REPORT SUMMARY :         1     NONOPT ( NOPT)
                              0 INFEASIBLE
                              0  UNBOUNDED
                              1     ERRORS ( ****)
```

Note that the solver status returned with a value of 5, meaning that the solver has been interrupted because more than `domlim` evaluation errors have been encountered. The type of evaluation error and the equation causing the error are also reported.

In case the solver returns an intermediate feasible solution because of evaluation errors, the following solve will still be attempted. The only fatal GAMS error that can be caused by a solver program, is the failure to return any solution at all. If this happens all possible information is listed on the GAMS output file and any solves that follow will not be attempted.

4.38.9 Customizing the Output File

This section reviews the most commonly used dollar control options, options, and command line parameters to customize output in the listing file. Table 1 lists dollar control options, options, and command line parameters that are used in the input file to control the amount of detail in the output file produced by the GAMS compiler. Table 2 lists dollar control options and command line parameters that can change the layout and appearance of the output. The first column of the two tables shows the purpose of customizing the output that differs from the GAMS default behavior.

The output generated by display statements can also be customized. This topic is covered in section Display Controls from chapter The Display Statement. See also the complete list of dollar control options that affect the output format in section Dollar Control Options Affecting the Output Format.

Table 1: ***Customization of output to be included in the listing file***

Customization	Method	Further Details
suppress the echo print	$offlisting	$onlisting restores the default behavior. Any lines between $offlisting and $onlisting will not appear in the echo print.
suppress include files in the echo print	$offinclude	$oninclude restores the default behavior.
activate the symbol reference map	$onSymXRef	Maps are most often turned on or off at the beginning of the program and left as initially set. $onSymXRef activates the symbol reference map and $offSymXRef restores the default behavior.
activate the symbol listing map	$onSymList	Maps are most often turned on or off at the beginning of the program and left as initially set. $onSymList activates the symbol listing map and $offSymList restores the default behavior.
activate the unique element listing	$onUELList	Maps are most often turned on or off at the beginning of the program and left as initially set. $onUELList activates the unique element listing and $offUELList restores the default behavior.

Customization	Method	Further Details
activate the unique element reference map	$onUELXRef	Maps are most often turned on or off at the beginning of the program and left as initially set. $onUELXRef activates the unique reference map $offUELXRef restores the default behavior.
suppress or expand the equation listing	option limrow	The statement `option limrow = 0;` will suppress the equation listing; the statement `option limrow = n;` will expand it to `n` equations. The statement must be placed *before* The Solve Statement.
suppress or expand the column listing	option limcol	The statement `option limcol = 0;` will suppress the column listing; the statement `option limcol = n;` will expand it to `n` columns. The statement must be placed *before* The Solve Statement.
suppress the solution listing	option solPrint	The statement `option solPrint = off;` will suppress the solution listing. The statement must be placed *before* The Solve Statement.
restrict output to just a few displays	save restart feature	With this strategy the listing file will contain only the output generated by the desired display statements. This facilitates concentrating on a narrow set of output while remaining capable of generating a lot more output. For details see section Generating Concise Listing Files in chapter The Save and Restart Feature.
show slack variables in the solution listing	option solslack	The statement `option solslack = 1;` causes the equation output in the solution listing to show slack (`.slack`) variable values instead of level (`.l`) values. The statement must be placed *before* The Solve Statement.

Table 2: ***Customization that changes the output layout***

Customization	Method	Further Details
change the default header	$title	The directive $title causes every page on the output to have the header specified in `text`. The header may be reset later by using another line starting with $title . Currently, the `text` may have up to 80 characters.
add a subheader	$stitle	The statement `$stitle text` causes every page on the output to have the subheader specified in `text`. The header may be reset later by using another line starting with $stitle. Currently, the `text` may have up to 80 characters.

Customization	Method	Further Details
start a new page in the echo print	$eject	
start a new page in LST file if less than n lines are left	$lines	The directive is $lines, where n is the number of lines left on the current page.
change width and length of a page	command line parameters PageWidth and PageSize	The general syntax is: gams mymodel pw=n ps=m. Here n is desired page width and the desired page length is m. Note that pw and ps are synonyms of PageWidth and PageSize respectively.
change the echo print to upper case only	$onUpper	The directive $offUpper restores the default.
change lines in the echo print to be double spaced	$double	The directive $single restores the default beahvior.
change the marker ****	$stars	

4.39 GAMS Log

4.39.1 Introduction

The GAMS log provides the log of a GAMS run including the GAMS job description, defined command line parameters, license information, solver log, and GAMS compilation and execution output. Looking at the log gives a brief idea about the success of a GAMS run and is usually a useful first step in debugging a GAMS model. The detailed output from GAMS is displayed in the output file. Log output can be obtained in the form of a text file (typically with .log extension) that is generated with each GAMS run based on the command line parameters logOption and logFile. The GAMS log consists of three major sections: header, compilation log, and execution log.

4.39.2 Header

The header of the GAMS log consists of information that is independent of the content of the executed GAMS file. It contains the job description, parameter log, and license log.

4.39.2.1 Job Description

In this line, the name of the GAMS file being run along with a timestamp, the GAMS version and operating system information is provided.

```
--- Job trnsport.gms Start 08/30/21 17:29:15 36.1.0 r2c0a44a WEX-WEI x86 64bit/MS Windows
```

4.39.2.2 Parameter Log

The default parameters for the operating system and configuration are obtained via parameter file (gmsprmNT.txt on Windows, gmsprmun.txt on Linux, gmsprmun.txt on macOS) and gamsconfig.yaml files. The parameter log explicitly lists all additional parameters in the following lines. For example, input displays the full path of the GAMS input file. In the example log provided below, the output file option PageSize is set to 0. ScrDir is the full path to the scratch directory; SysDir is the full path to the systems directory, logOption is defined that controls the location of the output log.

```
--- Applying:
    C:\GAMS\36\gmsprmNT.txt
    C:\Users\jsmith\Documents\GAMS\gamsconfig.yaml
--- GAMS Parameters defined
    Input C:\test\trnsport.gms
    PageSize 0
    ScrDir C:\test\225a\
    SysDir C:\GAMS\36\
    LogOption 3
    ErrMsg 1
    ErrorLog 99
    IDE 1
    LstTitleLeftAligned 1
```

4.39.2.3 License Log

This section provides information about the license such as licensee, the organization, and the license number is provided on the second line. The full path to the license file that is being read is provided on the third line. The last line in the image shows the GAMS license type being used.

```
Licensee: John Smith                                    G201001/0001CS-GEN
          The ACME Corporation, USA                                 DC0000
          C:\Users\jsmith\Documents\GAMS\gamslice.txt
          Other time limited license
```

4.39.2.4 Processor Information and GAMS version

This section provides information about the processor and GAMS version being used.

```
Processor information: 1 socket(s), 8 core(s), and 16 thread(s) available
GAMS 36.1.0   Copyright (C) 1987-2021 GAMS Development. All rights reserved
```

4.39.3 Compilation Log

A GAMS run consists of two phases: compilation and execution. The following lines provide details about the compilation, memory usage, the time elapsed in compilation. The number in parentheses refers to the line number in the echo print of the input file. In addition to the above-mentioned output, this part of the log also contains information about the output such as as GDX files, the files included using $include, and outputs of commands such as $call as shown in the following lines.

```
--- Starting compilation
--- clad.gms(27) 2 Mb
--- $echo File C:\test\includefilename.gms
--- clad.gms(30) 2 Mb
--- . includefilename.gms(1) 2 Mb
--- clad.gms(32) 2 Mb
--- call echo "hello!"
"hello!"
--- clad.gms(45) 3 Mb
--- GDXin=C:\test\claddat.gdx
--- GDX File ($gdxIn) C:\test\claddat.gdx
--- clad.gms(164) 3 Mb
```

4.39.4 Execution Output

The beginning of the execution phase of the GAMS run is clearly indicated in the GAMS log:

```
--- Starting execution: elapsed 0:00:00.010
```

In general, the execution log output indicates the progress of the GAMS job, including line numbers, memory usage, elapsed time for the run, and time required for long-running statements. Several statements also result in specific log output.

Output of GDX files generated using execute_unload is shown in the log as follows.

```
--- GDX File (execute_unload) path\to\outputfile.gdx
```

The output for reference files generated is shown in the log as follows.

```
--- RefFile  path\to\trnsport.ref
```

Similarly, the information about a file written using the Put writing facility is mentioned in the log as follows.

```
--- Putfile results  path\to\results.txt
```

The output of the program flow control features such as abort is shown as follows.

```
*** Error at line 61: Execution halted: abort$1 'model not solved as expected'
```

If there are loops in the program, the log shows information about the current iteration as follows.

```
--- LOOPS scen = s1
```

Similar to the output of call command shown during compilation phase, the output for execute command is shown in the execution phase. The following line of GAMS code

```
execute "echo output of execute command";
```

will result in the following output in the log.

```
--- trnsport.gms(67) 4 Mb
output of execute command
```

For each solve statement, there will also be a solver log embedded in the execution output (see below for details regarding the solver log). After a solver log, the execution output continues. It ends with the status of the GAMS Job (here `*** Status:  Normal completion`), a timestamp and the total run time of the GAMS job.

```
--- Executing after solve: elapsed 0:00:02.892
--- trnsport.gms(66) 4 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 08/30/21 17:29:18 elapsed 0:00:02.892
```

4.39.4.1 Solver Log

One of the most important GAMS statements is the solve statement: a solve results in a substantial amount of log output.

The first line in the execution output is about model generation. Model generation can take a long time in which case the log also shows progress and equations that take long to generate.

```
--- Generating LP model transport
--- trnsport.gms(64) 4 Mb
---   6 rows  7 columns  19 non-zeroes
```

This is followed by some statistics about the range of values seen in the model:

```
--- Range statistics (absolute non-zero finite values)
--- RHS       [min, max] : [ 2.750E+02, 6.000E+02] - Zero values observed as well
--- Bound     [min, max] : [        NA,        NA] - Zero values observed as well
--- Matrix    [min, max] : [ 1.260E-01, 1.000E+00]
```

After model generation, the solver name and solveLink are displayed.

```
--- Executing CPLEX (Solvelink=2): elapsed 0:00:02.066
```

The log output following this line is specific to the solver and will usually display the objective function value.

```
Optimal solution found
Objective:          153.675000
```

4.40 The GAMS Call and Command Line Parameters

There are multiple ways to trigger the run of a GAMS model: Running a model via `F9` from GAMS Studio, executing `GAMSJob.Run` method in the object oriented APIs or calling the `gams` executable from a command line. In all cases the same GAMS engine runs the user's model. Several options are available to customize the GAMS run. Depending on the particular way GAMS is triggered these options are supplied in different ways. For example, in Studio via the parameter editor in the toolbar, in the object oriented API through the GAMSOptions class, and from the command line via command line arguments. In order to avoid confusion with the often used word *option* we refer to these entities as *command line parameters* and demonstrate their use from the command line. Although details will vary with the type of computer and operating system used, the general operating principles are the same on all machines.

In this chapter we will introduce how GAMS is called from the command line and how parameters may be specified on the command line or in other ways. In addition, we will discuss user-defined command line parameters, compile-time variables and compile-time constants, which are GAMS specialties, and environment variables in GAMS. Moreover, most of the chapter is dedicated to the detailed description of all GAMS options (i.e. options available via command line parameters, option statement, and model attribute.

In almost all cases such a run continues from start to end without any user interaction. A GAMS run usually consists of two phases: the compilation and execution phase.

4.40.1 The Generic GAMS Call

The simplest way to start GAMS from a command shell is to enter the following command from the system prompt:

```
> gams myfile
```

GAMS will compile and execute the GAMS statements in the file `myfile`. If a file with this name cannot be found, GAMS will look for a file with the extended name `myfile.gms`. During the run GAMS will print a log to the console and create a listing file that is written by default to the file `myfile.lst`. For example, the following statements retrieves and runs the model [**TRNSPORT**] from the GAMS model library with the responds from the GAMS system:

```
> gamslib trnsport
Copy ASCII : trnsport.gms
> gams trnsport
--- Job trnsport Start 06/21/17 06:23:45 24.8.4 r60966 WEX-WEI x86 64bit/MS Windows
GAMS 24.8.4   Copyright (C) 1987-2017 GAMS Development. All rights reserved
Licensee: GAMS Development Corporation, Washington, DC   G871201/0000CA-ANY
          Free Demo,  202-342-0180,  sales@gams.com,  www.gams.com   DC0000
--- Starting compilation
--- trnsport.gms(69) 3 Mb
--- Starting execution: elapsed 0:00:00.010
--- trnsport.gms(45) 4 Mb
--- Generating LP model transport
--- trnsport.gms(66) 4 Mb
---   6 rows  7 columns  19 non-zeroes
--- Executing CBC: elapsed 0:00:00.018

COIN-OR CBC      24.8.4 r60966 Released Apr 10, 2017 WEI x86 64bit/MS Windows

COIN-OR Branch and Cut (CBC Library 2.9)
written by J. Forrest

Calling CBC main solution routine...
0  Obj 0 Primal inf 900 (3)
4  Obj 153.675
Optimal - objective value 153.675
Optimal objective 153.675 - 4 iterations time 0.052

Solved to optimality.
--- Restarting execution
--- trnsport.gms(66) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(68) 3 Mb
*** Status: Normal completion
--- Job trnsport.gms Stop 06/21/17 06:23:46 elapsed 0:00:01.249
```

Note

GAMS can also run source files that have been compressed with Posix utility `gzip` or have been encrypted with tool ENDECRYPT, for example

```
> gamslib trnsport
> gzip trnsport.gms
> gams trnsport.gms.gz
```

results in

- retrieval of the [**TRNSPORT**] model from the **GAMS Model Library**
- compression the source file `trnsport.gms`
- GAMS running the compressed file `trnsport.gms.gz`.

Observe that some GAMS options may be specified on the command line as part of the GAMS call. In addition, command line parameters may be set by specifying a secondary customization parameter file and by modifying the GAMS system parameter file. As command line parameters may be specified in several different ways, there are rules of precedence in case of conflicting instructions. We will discuss these topics in the next four subsections.

4.40.1.1 Specifying Options Through the Command Line

GAMS permits certain options to be passed through the command line. The syntax of the simple GAMS call is extended as follows:

```
> gams myfile key1=value1 key2=value2 ...
```

Here `key1` is the name of the option that is being set on the command line and `value1` is the value to which the option is set. Depending on the option, `value1` could be a character string, an integer number or a real number. Note that the options that may be set on the command line are called *GAMS command line parameters*.

For example, consider the following commands to run the transportation model [**TRNSPORT**]:

```
> gams trnsport  o myrun.lst  logOption 2
> gams trnsport -o myrun.lst -logOption 2
> gams trnsport /o myrun.lst /logOption 2
> gams trnsport  o=myrun.lst  logOption=2
> gams trnsport -o=myrun.lst -logOption=2
> gams trnsport /o=myrun.lst /logOption=2
```

All six commands above are equivalent: each directs the output listing to the file `myrun.lst`. Note that `o` is the synonym of the command line parameter output and it is set to the value `myrun.lst`. In addition, the parameter LogOption is set to 2, which has the effect that the log output is redirected to the file `trnsport.log`. Please also note that the option name can be specified without consideration of the casing of the option. Hence `logoption` works as well as `LogOption` or any other casing. The way options are specified can vary with each option. So a mixed alternative, e.g. `gams trnsport -o=myrun.lst /logoption 3` is valid too.

In addition to predefined command line parameters, GAMS also allows user-defined command line parameters which work in tandem with compile-time variables. User-defined parameters, also called *double dash parameters*, and compile-time variables are introduced in section Double Dash Parameters, Compile-Time Variables and Environment Variables below.

Option Values with Spaces

One may want to set string valued options including double dash parameters to an empty string or a string containing spaces, even trailing or leading spaces. All this requires the use of quotes. Quotes may be handled differently, depending on the operating system. Users should be attentive when using such string values. This is also important if GAMS is called from within a GAMS program with $call [=]gams ... and execute [=]gams ... because platform independence might be violated.

A platform independent way to set a string value with spaces is to use the variant that separates the option name and option value by a space, e.g. `opt ""`. The variant without space, e.g. `opt=""` might cause issues on some Unix shells. Consider the following example where `spacy.gms` contains only the following line to write the content of the double dash parameter dd to the log:

```
$log value of dd: >%dd%<
```

Here is a sequence of GAMS runs with a variety of string values

```
> gams spacy --dd normal
> gams spacy --dd "normal"
> gams spacy --dd ""
> gams spacy --dd " leading"
> gams spacy --dd "trailing "
> gams spacy --dd " leading and trailing "
> gams spacy --dd "   "
```

that result on all platforms in the following log output (only relevant part):

```
value of dd: >normal<
value of dd: >normal<
value of dd: ><
value of dd: > leading<
value of dd: >trailing <
value of dd: > leading and trailing <
value of dd: >   <
```

An even better way to pass arguments to a GAMS run without the system shell interfering with interpretation of some characters is described in the next section.

Attention

> GAMS options specifying file names may also contain leading and trailing blanks. Please be aware that while leading blanks in a file name are significant on all platforms, trailing blanks are only significant on some platforms (e.g. Linux). Hence `output "mymodel.lst "` and `output "mymodel.lst"` refer to the same file on Windows and to different files on Linux.

4.40.1.2 Specifying Options Through a Secondary Parameter File

Command line parameters may also be set by specifying a secondary customization parameter file. For example, we will create a file with the following two lines, call it `moreOptions.txt` and save it in the current working directory.

```
limCol=0
limRow=0
```

Note that setting limcol and limrow to zero will suppress the column listing and equation listing respectively. Moreover, note that we can specify the option in different way in this secondary parameter file, even have multiple options on one line. In the next step we will use the following command to run the transportation model [**TRNSPORT**]:

```
> gams trnsport parmFile=moreOptions.txt
```

Note that this call has the same effect as the following specification:

```
> gams trnsport limCol=0 limRow=0
```

Observe that command line parameter include files are particularly useful if modelers wish to use the same set of command line parameters repeatedly.

If options are listed multiple times on the command line, the last specification sets the option. This is important in particular in combination with a secondary parameter file. In the following example, GAMS will operate with `limRow=0`:

```
> gams trnsport limRow=10 parmFile=moreOptions.txt
```

while with the order reversed, GAMS operates with `limrow=10`:

```
> gams trnsport parmFile=moreOptions.txt limRow=10
```

4.40.1.3 Specifying Options Through the GAMS System Parameter File

A third way to specify command line parameters is by modifying the GAMS system parameter file, which is part of the GAMS system. It has different names depending on the operating system: `gmsprmnt.txt` for Windows and `gmsprmun.txt` for UNIX/Linux and macOS. The parameter file may be modified in the following way:

```
*****************************************************************
* GAMS 2.50 default Unix parameter file                         *
* Gams Development Corp.                                        *
* Date : 4 May, 1998                                            *
*****************************************************************
* entries required by CMEX, put in by the gams script
* SYSDIR
* SCRDIR
* SCRIPTNEXT
* INPUT
PageWidth 95
ParmFile "c:\some\central\location\moreOptions.txt"
```

Note that the last two lines were added to the standard GAMS system parameter file. As a consequence, each GAMS run will have a print width of 95 columns on the pages of the listing file. In addition, the command line parameters specified in the file `moreOptions.txt` will now apply to each GAMS run.

4.40.1.4 The GAMS Configuration in YAML Format

The GAMS configuration file `gamsconfig.yaml` written in the YAML format is easy to process by humans and computers. The GAMS Configuration Editor can be used to view and edit the GAMS configuration file but any other text editor can be used as well. This file has three configuration sections: command line parameters (`commandLineParameters`), operating system environment variables (`environmentVariables`), and external solver configuration (`solverConfig`). The latter is only relevant if you need to make a solver available to GAMS that is not shipped with the GAMS distribution. The COIN-OR GAMSlinks project has solver (links), e.g. Couenne, that could be added this way.

The first two sections are most easily described by an example:

```
commandLineParameters:
  - intVarUp:
      value: 1
  - eolCom:
      value: '//'
      maxVersion: 31
  - eolCom:
      value: '#'
      minVersion: 32
environmentVariables:
  - GDXCOMPRESS:
      value: 1
      maxVersion: 31
  - GDXCOMPRESS:
      value: 0
      minVersion: 32
  - PATH:
      value: /my/path/to/R/bin:/my/path/to/R/site-bin
      minVersion: 31.1.0
      maxVersion: 31.1.0
      pathVariable: True
```

In this example, the `gamsconfig.yaml` initializes some command line parameters and some environment variables. It initializes intVarUp unconditionally to 1. It also sets eolCom to the character sequence `//` for GAMS version 31 and earlier, while it uses the character `#` for GAMS version 32 and later. The keywords `minVersion` and `maxVersion` allow to have different entries for the same command line parameter for different versions. Also the file `gamsconfig.yaml` allows the repetition of entries. Hence, when GAMS runs it will filter all entries with appropriate `minVersion` and `maxVersion` and applies the setting of entries in sequence, so the last entry wins. Similarly, GAMS sets the environment variables `GDXCOMPRESS` to 0 or 1 depending on the actual version of GAMS. The last entry to set the `PATH` environment variable will be only applied for the GAMS version 31.1.0. Moreover, the indicator `pathVariable` set to `True` will result in prefixing the environment variable `PATH` with `/my/path/to/R/bin:/my/path/to/R/site-bin`.

The YAML syntax for the `gamsconfig.yaml` file is straightforward. Each section starts with a fixed name (`commandLineParameters` or `environmentVariables`) and is followed by a list of entries. Each entry starts with `-` and the name of the command line parameter or the name of the environment variable followed by a colon `:`. The next lines start with a keyword (`value`, `minVersion`, `maxVersion` and for environment variables also `pathVariable`) followed by a colon `:` and a value. The syntax of YAML files is very simple but it is crucial to use a consistent alignment. So all entries of a section must be perfectly aligned. Please be careful when mixing spaces and the tab character. If GAMS complains during the processing of a `gamsconfig.yaml` file with e.g. `System problems in yaml_parser_parse: mapping values are not allowed in this context` most likely your alignment is off as in the following erroneous example:

```
commandLineParameters:
  - eolCom:
      value: '//'
        maxVersion: 31
```

The lines with `value` and `maxVersion` need to start in the same column. There are plenty of online syntax checkers for YAML files. Use your favorite search engine and search for `yaml validator` (check the popular YAML validator) and copy and paste your problematic `gamsconfig.yaml` to get more insight into the YAML syntax problem. Moreover, the `gamsconfig.yaml` file has some required fields and some requirements for the value of some keys. In fact there is a schema file that validates that a syntactically correct `gamsconfig.yaml` fulfills the GAMS schema for configuration files. The schema file is part of the distribution and can be found in the system directory (`gamsconfig_schema.json`). One can validate a `gamsconfig.yaml` against this schema via a Python program (you might need to install additional Python packages `jsonschema` and `ruamel.yaml`):

```
import jsonschema
from ruamel.yaml import YAML
import json

with open('gamsconfig_schema.json') as f:
  schema = json.load(f)

yaml = YAML(typ="safe")
with open('gamsconfig.yaml') as f:
  config = yaml.load(f)

print(jsonschema.validate(instance=config, schema=schema))
```

Besides the required fields like `value` the schema also imposes restrictions on some value types (e.g. `True` and `False` for `pathVariable` and a proper release number for `minVersion` and `maxVersion`). The form of the release number can be just the major release (e.g. `31`), the major and minor release number (e.g. `31.1`) or the full detailed release number, e.g. `31.1.0`. So if `maxVersion` is set to e.g. `31` then all releases of `31` fulfill this version requirement.

Some parts of the GAMS system react on environment variables, e.g. the environment variable GMSPYTHONLIB decides about the Python library and installation used for embedded Python code. At the start of a GAMS run environment variables are set to the values given in the `gamsconfig.yaml` file. This is a convenient way to manage GAMS-related environment variables without utilizing OS-specific ways to set environment variables.

GAMS will process a sequence of `gamsconfig.yaml` files. It starts with the file in the GAMS system directory and then searches through some system-wide and user-specific standard locations.

Note

With GAMS 32 new defaults for several options were introduced. We have collected these options and their old or legacy default values in the file `gamsLegacyConfig.yaml` located in the GAMS system directory. Copy this file as `gamsconfig.yaml` to a location searched by GAMS to continue to work with the legacy defaults. The `minVersion` keyword in this file indicates when the new default became active.

Adding Solvers via gamsconfig.yaml

The third section in the GAMS configuration file gamsconfig.yaml allows to make a solver not shipped with GAMS available to the GAMS system. For that, an interface between an instantiation of a GAMS model and a solver needs to be implemented by use of libraries GMO and GEV. The COIN-OR GAMSlinks project provides examples of such solver interfaces. For the remainder of this section assume that we want to introduce a new version of the MINLP/MIQCP solver SHOT. GAMS knows about available solvers from the sub-system configuration file gmscmpNT.txt (gmscmpun.txt for non-Windows platforms). Here is a section from this file for the MINLP/MIQCP solver SHOT on Windows:

```
SHOT 1001 5 000102030405 1 0 2 MINLP MIQCP
gmsgennt.cmd
gmsgennx.exe
shtcclib64.dll sht 1 1
```

The gamsconfig.yaml file provides the same information in a more descriptive way:

```
solverConfig:
  - SHOT2:
      minVersion: 33
      fileType: 1001
      dictType: 5
      licCodes: 000102030405
      defName: optshot.def
      scriptName: gmsgennt.cmd
      executableName: gmsgennx.exe
      library:
        libName: D:\Users\janeDoe\Downloads\gamslink\GamsShot-20.dll
        auditCode: sht
        solverInterfaceType: 1
        threadSafeIndic: True
      modelTypes:
        - MINLP
        - MIQCP
```

The new solver will be called SHOT2 and has almost the identical information as the solver SHOT that comes with GAMS. We could also use the name SHOT again. This would hide the original solver SHOT and use the one described by gamsconfig.yaml. In addition to the solver information, this section allows the keywords min/maxVersion that instructs whether to include the solver depending on the GAMS version.

The fileType (line 1 second item in the section of gmscmpNT.txt) determines the type of the scratch files that contain the model instance. The one-digit determines binary (1) or text (0) format of the scratch files. The ten-digit determines whether GAMS puts the initial marginal information into the scratch files (1 for writing, 0 for not writing). The scratch file with the constraint matrix is stored column wise, the hundred-digit instructs GAMS to write (if at 1) the row count per column, so the solver can layout memory in advance for storing the constraint matrix. With GAMS' current libraries gmo and gev this information is not used and hence the hundred-digit should be at 0. The thousand-digit is about scaling. A non-zero value instructs GAMS to write the information stored in the .scale/.prior/.stage variable and equation attributes into the scratch files. With a value of 1 this is only done if scaleOpt/priorOpt are on. With a value of 2 this information is written independently of scaleOpt/priorOpt. With a value of 3 for models with variable and matching information (e.g. MCP) the reported marginals will be scaled by GAMS. The ten thousand-digit is always 0 and the hundred thousand-digit (if set to 1) instructs GAMS to write the variable/equation matching information to the scratch files for models that need this information (e.g. MCP). The leading zeros of the value for key fileType should be omitted.

The dictType (line 1 third item in the section of gmscmpNT.txt) decides about the type of the dictionary scratch file that contains the mapping information between the GAMS names and the variable and equation indexes seen by the solver. For example, this helps the solver to form good variable and equation names for error messages. Four values are allowed:

- 0: no dictionary information
- 5: dictionary with all labels that are used in the model instance
- 15: dictionary with all labels that are used in GAMS program
- 25: dictionary with all labels that are used in GAMS model plus an extra GDX file with the entire GAMS database (similar to `execute_unload "everything";` prior to the solve statement)

The `licCode` (line 1 fourth item in the section of `gmscmpNT.txt`) gives a guess about what license codes are required to run the solver. This is not the actual license information but allows other programs like e.g. "GAMS Studio" to list the solvers available with a particular license. In most cases solvers that are added this way should list the GAMS BASE license codes "000102030405".

The `defName` (line 2 second item in the section of `gmscmpNT.txt`) points to a file that contains all the solver options for the solver. This key does not need to be provided (same in `gmscmpNT.txt`) if the `defName` is `opt` + solvername + `.def` and the file resides in the GAMS system directory. Since we use the name `SHOT2` GAMS would expect the solver options to be in `optshot2.def`. If there were an `optshot.def` in the GAMS system directory for the original `SHOT`, we could reuse that file and list `optshot.def` for the `defName` key. The `defName` provided here is used by other programs like "GAMS Studio" that can assist with building solver options files.

The next three items on line 1 in the section of `gmscmpNT.txt` are for internal purposes and not discussed here in detail. They can be set via `gamsconfig.yaml` via the keys `defaultOkFlag` and `hiddenFlag` with values `True` or `False`.

GAMS has different ways of calling a solver controlled by the solveLink option. The `scriptName` (line 2 in the section of `gmscmpNT.txt`) contains the name of the script that calls the solver (in case of `solveLink=0` (chainScript) or 1 (callScript)). For solvers that are capable of `solveLink=5` (loadLibrary) a generic script `gmsgennt.cmd` (or `gmsgenus.run`) can be used. The `executableName` (line 3 in the section of `gmscmpNT.txt`) provides the name of the solver executable. Again for solvers capable of `solveLink=5` (loadLibrary) a generic executable `gmsgennx.exe` (or `gmsgenux.out`) can be used.

The section library is necessary only for solvers capable of `solveLink=5` (loadLibrary). The `libName` contains the name of the solver library file (line 3 first item in the section of `gmscmpNT.txt`). The `auditCode` (line 3 second item in the section of `gmscmpNT.txt`) is the prefix of the functions in the solver library (e.g. `shtCallSolver`). The `solverInterfaceType` (line 3 third item in the section of `gmscmpNT.txt`) defines the API implemented by the solver library and loaded by GAMS. Currently, there are only two types: 1 (`xyzReadyApi` and `xyzCallSolver`) and 2 (`xyzCallSolver` only). The `threadSafeIndic` (line 3 third item in the section of `gmscmpNT.txt`) indicates whether the solver library is thread-safe and can be used in combination with `solveLink=6` (aSyncThreads).

The final section `modelTypes` is about the model types the solver can handle. This information can be found in line 1 after item 7 in `gmscmpNT.txt`.

4.40.1.5 Order of Precedence for Options

The order of precedence for command line parameters including customization specifications in the GAMS IDE is as follows:

1. Command line parameters that are specified on the command line or in the Studio/IDE command line field.
2. Command line specifications in the IDE options window `Execute` (this is reached through File|Options|Execute).

3. Command line specifications in the specific IDE dialogs, e.g. `Output` (this is reached through File|Options|...).
4. Entries in the GAMS configuration files `gamsconfig.yaml`.
5. Entries in the GAMS system parameter file.

Many command line parameters initialize the `default` for a GAMS Option also accessible inside the GAMS program. For example, the system default for option MIQCP is `SBB`. If the command line parameter `MIQCP` has been set, e.g. to `DICOPT`, `DICOPT` will be new default MIQCP solver. Inside the GAMS program, one can reset the MIQCP via `option MIQCP=SHOT;` but setting it back to the default will result in `DICOPT` (not `SBB`) being the default solver: `option MIQCP=default;`.

4.40.2 Double Dash Parameters, Compile-Time Variables and Environment Variables

In this section we will cover double-dash parameters that enable users to set values for specific variables on the command line. These variables are substituted with their respective specified values at *compile* time. We will also discuss how operating system specific environment variables may be accessed and modified from a GAMS program.

4.40.2.1 Double Dash Parameters and Compile-Time Variables: A Simple Example

We will introduce double dash parameters and compile-time variables using as an example the well-known transportation model [**TRNSPORT**]. Assume we wish to explore how the solution changes if the demand assumes various values. To model this, we will introduce the compile-time variable `DMULT`, a multiplier for the demand `b`:

```
$set DMULT 1
...
demand(j) ..   sum(i, x(i,j))  =g=  %DMULT% * b(j) ;
```

Note that `DMULT` is defined with the dollar control option $set and its value is set to 1, which corresponds to the base line. Observe that the compile-time variable is referenced in the equation with the `%...%` notation. `%DMULT%` will be replaced at compile time by its value, in this case 1. We are now in the position to run the program multiple times by just changing the value of `DMULT` in the first line, which will automatically change the multiplier in the equation to the respective value.

GAMS offers a more convenient way for setting a compile-time variable like `DMULT` to a variety of values:

```
$if not set DMULT $set DMULT 1
...
demand(j) ..   sum(i, x(i,j))  =g=  %DMULT% * b(j) ;
```

Note that we set a default value for `DMULT` using conditional compilation. We may change this default on the command line when calling GAMS by specifying `DMULT` as a double dash parameter as follows:

```
> gams trnsport.gms --DMULT=0.9
```

Observe that with the specification `--DMULT=0.9` the compile-time variable `DMULT` is set and therefore the default does not apply. Thus the double dash parameter facilitates changing the value of a compile-time variable directly on the command line as part of the GAMS call while the respective GAMS file remains unchanged.

Assume that the model contains a second compile-time variable called `METHOD` that acts as a switch for various methods of solving the model and it may take the values 1, 2 and 3. In this case both compile-time variables may be set on the command line as follows:

```
> gams trnsport.gms --DMULT=1.12 --METHOD=3
```

In the next two subsections we will discuss double dash parameters and compile-time variables in more detail.

4.40.2.2 Double Dash Parameters

Double dash parameters are user-defined command line parameters that are used to define scoped compile-time variables or to assign values to scoped compile-time variables. The general syntax is:

```
> gams myfile --NAME=value
```

Here `NAME` is the name of the double dash parameter and `value` is its assigned value that may be any string. If the string contains spaces or other token terminating characters, the string value should be quoted. Consider the following simple example:

```
> gams myfile --keycity=Boston --myvalue=7.6 --dothis="display x;"
```

Suppose that `myfile.gms` contains the following lines with the scoped compile-time variables `keycity`, `myvalue` and `dothis`:

```
x("%keycity%")=%myvalue%;
%dothis%
```

Note that the compile-time variables are referenced using the notation `%...%`. The GAMS call above has the effect that at compile time the compile-time variables are substituted with the values specified through the double dash parameters resulting in the following:

```
x("Boston")=7.6;
display x;
```

GAMS offers three alternative syntax variants for defining double dash parameters:

```
> gams myfile //NAME=value
> gams myfile /-NAME=value
> gams myfile -/NAME=value
```

Note that the four syntax variants may be used interchangeably and have the same effect.

Double dash parameters are particularly useful for specifying granularity when modeling a discretization of time and or space. For example, in the model **[CHAIN]** the problem is to find the chain with minimal potential energy, assuming the chain has uniform density, is suspended between two points and has a given length `L`. Consider the following code snippet:

```
$if not set nh $set nh 50
Set nh / i0*i%nh% /;
```

The first two lines use conditional compilation to set a default value for the compile-time variable `nh`. Note that the value of `nh` determines the cardinality of the set `nh`, which is used for the discretization. The value of `nh` may be easily set on the command line to any desired value using `nh` as a double dash parameter:

```
> gams chain --nh=100
```

Note that the double dash functionality supersedes the command line parameters user1, ..., user5 which are accessible in the source file via `%gams.user1%, ..., %gams.user5%`. The example above would work with `user1` in the following way:

```
$set nh 50
$if not "%gams.user1"=="" $set nh %gams.user1%
Set nh / i0*i%nh% /;
```

The modified code could be called with:

```
> gams chain user1=100
```

Note that the double dash parameters facilitate using meaningful names instead of the generic names `user1, ..., user5`. This is especially useful if there are multiple parameters to pass on to the GAMS program.

Observe that the dollar control option $setDDList may be used to ensure that a model can only be run with the listed double dash parameters:

```
$setDDList nh
$if not errorFree $log *** Only allowed double dash options is: --nh=value
```

4.40.2.3 Compile-Time Variables

Compile-time variables are special variables that are substituted with their values at compile-time. They are not declared and defined with regular declaration statements like standard symbols (sets, parameters, ...), but they are defined with the dollar control option $set and its variants. There are three kinds of compile-time variables that differ in their scope level: local, scoped and global. An overview is given in Table 1.

Scope	Availability	Defined with	Removed from the system with
Local	Available only in the input file where they are defined	$setLocal.	$dropLocal
Scoped	Available in the input file where they are defined and in all include files of the input file.	$set	$drop
Global	Available in the input file where they are defined, in all parent files and in all include files.	$setGlobal	$dropGlobal

Table 1: Scope Levels for Compile-Time Variables in GAMS

Note that *scoped* compile-time variables may also be defined on the command line with double dash parameters. For example, in the example above the compile-time variable `DMULT` may be referenced in the equation `demand` without being defined with the dollar control option `$set` as long as it is defined and set on the command line. Note further, that *global* compile-time variables are saved in work files.

While the scope of a compile-time variable cannot be directly changed, but dropping and adding variables in different scopes accomplishes the same. Consider the following example:

```
$set MYVAR xxx

* From scoped to global
$ifThen set MYVAR
$  setGlobal MYVAR %MYVAR%
$  show
$  drop MYVAR
$  show
$endIf

* From global to local
$ifThen setGlobal MYVAR
$  setLocal MYVAR %MYVAR%
$  show
$  dropGlobal MYVAR
$  show
$endIf
```

Note that the compile-time variable MYVAR is first defined as a scoped variable with the value xxx. The dollar control option ifThen tests whether MYVAR was defined with $set and since this is TRUE the next four dollar control statements are processed: a new *global* compile-time variable called MYVAR is defined and is set to the value of the *scoped* compile-time variable MYVAR, the resulting compile-time variables are shown, the *scoped* compile-time variable is removed from the system with the option $drop and the resulting compile-time variables are shown again. A similar procedure is followed to change the *global* compile-time variable MYVAR to a *local* compile-time variable with the same name and value.

The output generated by the four dollar control options $show follows:

```
Level SetVal                            Type       Text
-------------------------------------------------------
    0 MYVAR                             SCOPED     xxx
    0 MYVAR                             GLOBAL     xxx

Level SetVal                            Type       Text
-------------------------------------------------------
    0 MYVAR                             GLOBAL     xxx

Level SetVal                            Type       Text
-------------------------------------------------------
    0 MYVAR                             LOCAL      xxx
    0 MYVAR                             GLOBAL     xxx

Level SetVal                            Type       Text
-------------------------------------------------------
    0 MYVAR                             LOCAL      xxx
```

Observe that this report is called *Environment Report*. This name is unfortunate, since the variables reported are in fact compile-time variables. Environment variables in GAMS are discussed in section Environment Variables in GAMS below.

Note that if a compile-time variable is referenced with %MYVAR%, it could reference a global, scoped or local compile-time variable called MYVAR. GAMS will always access the compile-time variable MYVAR with the most local scope. Thus if all three scopes are defined, the local compile-time variable is accessed first, then the scoped and then the global, as demonstrated in the following example:

```
$set XXX scoped
$setLocal XXX local
$setGlobal XXX global
$log %XXX%
$dropLocal XXX
$log %XXX%
$drop XXX
$log %XXX%
$dropGlobal XXX
$log %XXX%
```

The resulting log output will be:

```
local
scoped
global
%XXX%
```

Note that how %XXX% will be handled if no compile-time variable XXX is defined, is determined by the command line parameter stringChk.

For a full list of dollar control options that affect compile-time variables, see section Dollar Control Options for Compile-Tim

4.40.2.4 Environment Variables in GAMS

GAMS programs have access to operating system environment variables via %sysEnv.NAME%. Operating system environment variables may be modified or new environment variables may be defined with the dollar control option $setEnv. Consider the following artificial example:

```
$log %sysEnv.GEORGE%
$setEnv GEORGE Dantzig
$log %sysEnv.GEORGE%
$dropEnv GEORGE
$log %sysEnv.GEORGE%
```

Note that the dollar control option $dropEnv removes an environment variable. The log output follows:

```
%sysEnv.GEORGE%
Dantzig
%sysEnv.GEORGE%
```

There are two environment variables in GAMS that are specific to the GDX facility: GDXCONVERT and GDXCOMPRESS. Since GDX is used by utilities and other programs some general customization can be achieved via these environment variables. Their values determine the type of GDX files that are written. These environment variables may be overwritten through the command line parameters gdxConvert and gdxCompress or inside the GAMS file with the dollar control option $setEnv. Consider the following example that uses the latter functionality:

```
Scalar x /1/;
$log %sysEnv.GDXCONVERT%
$log %sysEnv.GDXCOMPRESS%
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxdump x.gdx -v | grep "File format\|Compression"

$setEnv GDXCONVERT v6
$setEnv GDXCOMPRESS 1
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxdump x.gdx -v | grep "File format\|Compression"

$setEnv GDXCONVERT v7
$setEnv GDXCOMPRESS 0
$gdxOut x.gdx
$unLoad x
$gdxOut
$call gdxdump x.gdx -v | grep "File format\|Compression"
```

Note that `gdxdump` is a GDX utility that writes the contents of a GDX file as a GAMS formatted text file. The switch `-v` lets `gdxdump` print the file version information. With the `grep` utility we filter the lines that contain either `File format` or `Compression`. This code is run with the following call that initializes `GDXCOMPRESS` and `GDXCONVERT`:

```
> gams gdxenv.gms gdxCompress=0 gdxConvert=v6 lo=3
```

The output follows:

```
--- Starting compilation
v6
0
--- gdxenv.gms(7) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
*  File format    :    6
*  Compression    :    0
--- gdxenv.gms(14) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
*  File format    :    6
*  Compression    :    1
--- gdxenv.gms(21) 2 Mb
--- call gdxdump x.gdx -v | grep "File format\|Compression"
*  File format    :    7
*  Compression    :    0
--- gdxenv.gms(22) 2 Mb
--- Starting execution - empty program
```

4.40.3 Compile-Time Constants

Compile-time constants are constants that are related to some functions, model attributes or options. They have a fixed value and are referenced as `%prefix.constant%`. Here `prefix` is the name of the respective function, model attribute or option and `constant` is the name of the constant.

For example, the function handleStatus is used in the context of grid computing. Typically, a collection loop may take the following form:

```
loop(pp$(handleStatus(h(pp)) = 2), ... );
```

Alternatively, the following formulation may be used:

```
loop(pp$(handleStatus(h(pp)=%handleStatus.ready%), ... );
```

Observe that the compile-time constant `%handleStatus.ready%` equals the value of 2. See the table below for other compile-time constants that are related to the function `handleStatus`.

Note

Compile-time constants are replaced at compile time and cannot be manipulated or reassigned.

Though compile-time constants are most often used in the context of the function, model attribute or option indicated with the prefix, they are in fact context free and may be used anywhere where an integer is expected. Consider the following example:

```
Scalar x / %solPrint.on% /; display x;
```

A complete list of the compile-time constants is given in Table 2.

Table 2: Compile-Time Constants

Compile-Time Constant	value
%handleStatus.unknown%	0
%handleStatus.running%	1
%handleStatus.ready%	2
%handleStatus.failure%	3
%modelStat.optimal%	1
%modelStat.locallyOptimal%	2
%modelStat.unbounded%	3
%modelStat.infeasible%	4
%modelStat.locallyInfeasible%	5
%modelStat.intermediateInfeasible%	6
%modelStat.feasibleSolution%	7
%modelStat.integerSolution%	8
%modelStat.intermediateNonInteger%	9
%modelStat.integerInfeasible%	10
%modelStat.licensingProblem%	11
%modelStat.errorUnknown%	12
%modelStat.errorNoSolution%	13
%modelStat.noSolutionReturned%	14
%modelStat.solvedUnique%	15
%modelStat.solved%	16
%modelStat.solvedSingular%	17
%modelStat.unboundedNoSolution%	18
%modelStat.infeasibleNoSolution%	19

Compile-Time Constant	value
%platformCode.unknown%	0
%platformCode.DEX%	1
%platformCode.LEX%	2
%platformCode.WEX%	3
%platformCode.DAX%	4
%solPrint.off%	0
%solPrint.on%	1
%solPrint.silent%	2
%solPrint.summary% (deprecated)	0
%solPrint.report% (deprecated)	1
%solPrint.quiet% (deprecated)	2
%solveLink.chainScript%	0
%solveLink.callScript%	1
%solveLink.callModule%	2
%solveLink.asyncGrid%	3
%solveLink.asyncSimulate%	4
%solveLink.loadLibrary%	5
%solveLink.aSyncThreads%	6
%solveLink.threadsSimulate%	7
%solveOpt.replace%	0
%solveOpt.merge%	1
%solveOpt.clear%	2
%solveStat.normalCompletion%	1
%solveStat.iterationInterrupt%	2
%solveStat.resourceInterrupt%	3
%solveStat.terminatedBySolver%	4
%solveStat.evaluationInterrupt%	5
%solveStat.capabilityProblems%	6
%solveStat.licensingProblems%	7
%solveStat.userInterrupt%	8
%solveStat.setupFailure%	9
%solveStat.solverFailure%	10
%solveStat.internalSolverFailure%	11
%solveStat.solveProcessingSkipped%	12
%solveStat.systemFailure%	13

4.40.3.1 Command Line Parameters as Compile-Time Constants

The value of a command line parameter that was passed to the model can be accessed at compile time using the prefix `'gams'` and the name of the command line parameter: `%gams.parameter%`. Example:

```
> gams trnsport gdx=output.gdx
```

Trnsport.gms:

```
$setNames "%gams.input%" filepath filename fileextension
$log The input file '%filename%' is located in the directory %filepath%
$log All results are written to %gams.gdx%
```

Which results in the following log output:

```
The input file 'trnsport' is located in the directory C:\Users\robin\Documents\GAMS\Studio\workspace
All results are written to C:\Users\robin\Documents\GAMS\Studio\workspace\output.gdx
```

In addition to compile time, the value of a command line parameter can also be accessed at execution time.

4.40.4 GAMS Compile Time and Execution Time Phase

The GAMS log indicates different phases of a job run in the log file:

```
...
--- Starting compilation
...
--- Starting execution: elapsed 0:00:00.056
...
```

During compilation GAMS converts the GAMS user program into lower-level instructions that are executed during execution time. Before the user program is converted into lower-level instructions the compiler processes the input: the compile time variables and macros are substituted and comments are removed. Moreover, during compilation any dollar control option present in the user code is *executed*. Many of these dollar control options impact the behavior of the compilation phase (e.g. $include instructs the compiler to process a file before continuing processing the remaining part of the current file). The code the compiler actually converts into lower-level instructions is echoed (by default) to the listing file. The compiler also assembles the list of user symbols (sets, parameters, variables, ...) and the list of labels. These lists become immutable after the compiler finishes. So during execution time, for example, no new labels can be added. The only exception from this is during a continued compilation/execution using the save and restart facility. The separation between compile time and execution time is confusing especially for novice users and mistakes as the following are frequent:

```
file fInput / data.txt /;
scalar iCnt; for (iCnt=1 to 100, put fInput iCnt:0:0 /); putClose fInput;
set i /
$include data.txt
      /;
```

The intention of the code, that does *not* work, is clear: The put facility is used to create the input file `data.txt` that is included via the $include instruction. The problem with this code is that the `$include` instruction is *executed* at compile time while the code using the put statement is executed at execution time, i.e. after the compilation phase is over. Hence, the compiler tries to include this file *before* the put instructions are executed. If the file `data.txt` is not present, the compiler will terminate with a compilation error, but if a file with name `data.txt` is present this one will be processed by the `$include` and a mistake like this might remain undiscovered for a long time.

Most of the time GAMS performs compilation and execution in one GAMS job which makes it even harder to grasp the concept of compile and execution time. The command line parameter action can be used to separate the compilation and execution phase into multiple jobs.

4.40.5 List of Command Line Parameters

In the following two subsections we will present an overview of the command line parameters with brief descriptions. Detailed descriptions of all command line parameters follow in section Detailed Descriptions of All Options below.

Note

The value of a command line parameter that was passed to the model can be accessed at compile time and at execution time.

4.40.5.1 General Options

Option	Description
action	GAMS processing request
appendExpand	Expand file append option
appendLog	Log file append option
appendOut	Output file append option
asyncSolLst	Print solution listing when asynchronous solve (Grid or Threads) is used
captureModelInstance	Switch to capture all model instances within a run
case	Output case option for LST file
cErr	Compile time error limit
charSet	Character set flag
checkErrorLevel	Check errorLevel automatically after executing external program
connectIn	Specify YAML Connect script file processed at start of GAMS
connectOut	Specify YAML Connect script file processed at end of GAMS
curDir	Current directory
decryptKey	Key to decrypt a text file that was encrypted via $encrypt
dFormat	Date format
digit	Switch default for "$on/offDigit"
docFile	Filename stem for documentation files
dumpOpt	Writes preprocessed input to the file input.dmp
dumpOptGDX	Defines a GDX file name stem created when using DumpOpt
dumpParms	GAMS parameter logging
dumpParmsLogPrefix	Prefix of lines triggered by DumpParms>1
ECImplicitLoad	Allow implicit loading of symbols from embedded code or not
empty	Switch default for "$on/offEmpty"
encryptKey	Key to encrypt a text file using $encrypt
eolCom	Switch default for "$on/offEolCom" and "$eolCom"
eolOnly	Single key-value pairs (immediate switch)
epsToZero	Treat eps as zero when unloading to GDX
errMsg	Placing of compilation error messages
errNam	Name of error message file
error	Force a compilation error with message
errorLog	Max error message lines written to the log for each error
etLim	Elapsed time limit in seconds
execMode	Limits on external programs that are allowed to be executed
expand	Expanded (include) input file name

Option	Description
fdDelta	Step size for finite differences
fdOpt	Options for finite differences
fErr	Alternative error message file
fileCase	Casing of file names and paths (put, gdx, ref, $include, etc.)
fileStem	Sets the file stem for output files which use the input file name as stem by default
fileStemApFromEnv	Append a string read from an environment variable to the "FileStem"
filtered	Switch between filtered and domain-checked read from GDX
forceWork	Force GAMS to process a save file created with a newer GAMS version or with execution errors
forLim	GAMS looping limit
G205	Use GAMS version 2.05 syntax
GDX	GAMS data exchange file name
gdxCompress	Compression of generated GDX file
gdxConvert	Version of GDX files generated (for backward compatibility)
gdxSymbols	Select symbols that get exported when command line parameter GDX is set
gdxUels	Unload labels or UELs to GDX either squeezed or full
gridDir	Grid file directory
gridScript	Grid submission script
heapLimit	Maximum Heap size allowed in MB
IDCGDXInput	GDX file name with data for implicit input
IDCGDXOutput	GDX file name for data for implicit output
IDCGenerateGDX	Specify GDX file name of input and output side of data contract
IDCGenerateGDXInput	Specify GDX file name of input side of data contract
IDCGenerateGDXOutput	Specify GDX file name of output side of data contract
IDCGenerateJSON	Specify JSON file name of data contract
IDCJSON	Specify JSON file name to verify data contract
IDCProtect	Flag to control assignment protection of external input symbols
IDE	Integrated Development Environment flag
implicitAssign	Switch default for "$on/offImplicitAssign"
inlineCom	Switch default for "$on/offInline" and "$inlineCom"
input	Input file
inputDir, inputDir1..40	Input file directories
interactiveSolver	Allow solver to interact via command line input
jobTrace	Job trace string to be written to the trace file at the end of a GAMS job
keep	Controls keeping or deletion of process directory and scratch files
libIncDir	LibInclude directory
license	Use alternative license file
listing	Switch default for "$on/offListing"
logFile	Log file name
logLine	Amount of line tracing to the log file
logOption	Log option
lstTitleLeftAligned	Write title of LST file all left aligned
maxExecError	Execution time error limit
maxGenericFiles	Maximum number of generic file names tried at execution time file creation
maxProcDir	Maximum number of 225∗ process directories

Option	Description
MCPRHoldFx	Print list of rows that are perpendicular to variables removed due to the holdfixed setting
MIIMode	Model Instance Mode
multi	Switch default for "$on/offMulti[R]"
multiPass	Multipass facility
noNewVarEqu	Triggers a compilation error when new equations or variable symbols are introduced
on115	Generate errors for unknown unique element in an equation
output	Listing file name
pageContr	Output file page control option
pageSize	Output file page size (=0 no paging)
pageWidth	Output file page width
parmFile	Command Line Parameter include file
pLicense	Privacy license file name
prefixLoadPath	Prepend GAMS system directory to library load path
previousWork	Indicator for writing workfile with previous workfile version
procDir	Process Directory
procDirPath	Directory to create process directory in
procTreeMemMonitor	Monitor the memory used by the GAMS process tree
procTreeMemTicks	Set wait interval between memory monitor checks: ticks = milliseconds
profile	Execution profiling
profileFile	Write profile information to this file
profileTol	Minimum time a statement must use to appear in profile generated output
putDir	Put file directory
putND	Number of decimals for put files
putNR	Numeric round format for put files
putPS	Page size for put files
putPW	Page width for put files
reference	Symbol reference file
referenceLineNo	Controls the line numbers written to a reference file
replace	Switch between merge and replace when reading from GDX into non-empty symbol
scrDir	Scratch directory
scrExt	Scratch file extension to be used with temporary files
scrNam	Work file names stem
seed	Random number seed
showOSMemory	Show the memory usage reported by the Operating System instead of the internal counting
stepSum	Summary of computing resources used by job steps
strictSingleton	Error if assignment to singleton set has multiple elements
stringChk	String substitution options
suffixAlgebraVars	Switch default for "$on/offSuffixAlgebraVars"
suffixDLVars	Switch default for "$on/offSuffixDLVars"
suppress	Compiler listing option
symbol	Symbol table file
sys10	Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer

Option	Description
sys11	Dynamic resorting if indices in assignment/data statements are not in natural order
sys12	Pass model with generation errors to solver
sys15	Automatic switching of data structures used in search records
sys16	Disable search record memory (aka execute this as pre-GAMS 24.5)
sys17	Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5)
sys18	Use backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers
sys19	Disable permutation on Column Generation (aka execute this as pre-GAMS 36)
sysDir	GAMS system directory where GAMS executables reside
sysIncDir	SysInclude directory
tabIn	Tab spacing
tFormat	Time format
threadsAsync	Limit on number of threads to be used for asynchronous solves (solveLink=6)
timer	Instruction timer threshold in milli seconds
trace	Trace file name
traceLevel	Modelstat/Solvestat threshold used in conjunction with action=GT
traceOpt	Trace file format option
user1..5	User strings
warnings	Number of warnings permitted before a run terminates
workDir	Working directory
writeOutput	Switch to write output file
zeroRes	The results of certain operations will be set to zero if abs(result) LE ZeroRes
zeroResRep	Report underflow as a warning when abs(results) LE ZeroRes and result set to zero

4.40.5.2 Solver-Related Options

Option	Description
bRatio	Basis detection threshold
CNS	Constrained Nonlinear Systems - default solver
DNLP	Non-Linear Programming with Discontinuous Derivatives - default solver
domLim	Domain violation limit solver default
EMP	Extended Mathematical Programs - default solver
forceOptFile	Overwrites other option file section mechanism
holdFixed	Treat fixed variables as constants
holdFixedAsync	Allow HoldFixed for models solved asynchronously as well
integer1..5	Integer communication cells
intVarUp	Set mode for default upper bounds on integer variables
iterLim	Iteration limit of solver
limCol	Maximum number of columns listed in one variable block
limRow	Maximum number of rows listed in one equation block
LP	Linear Programming - default solver
MCP	Mixed Complementarity Problems - default solver

Option	Description
MINLP	Mixed-Integer Non-Linear Programming - default solver
MIP	Mixed-Integer Programming - default solver
MIQCP	Mixed Integer Quadratically Constrained Programs - default solver
MPEC	Mathematical Programs with Equilibrium Constraints - default solver
NLP	Non-Linear Programming - default solver
nodLim	Node limit in branch and bound tree
optCA	Absolute Optimality criterion solver default
optCR	Relative Optimality criterion solver default
optDir	Option file directory
optFile	Default option file
QCP	Quadratically Constrained Programs - default solver
resLim	Wall-clock time limit for solver
RMINLP	Relaxed Mixed-Integer Non-Linear Programming - default solver
RMIP	Relaxed Mixed-Integer Programming - default solver
RMIQCP	Relaxed Mixed Integer Quadratically Constrained Programs - default solver
RMPEC	Relaxed Mathematical Programs with Equilibrium Constraints - default solver
savePoint	Save solver point in GDX file
scriptExit	Program or script to be executed at the end of a GAMS run
scriptFrst	First line to be written to GAMSNEXT file.
solPrint	Solution report print option
solveLink	Solver link option
solver	Default solver for all model types that the solver is capable to process
solverCntr	Solver control file name
solverDict	Solver dictionary file name
solverInst	Solver instruction file name
solverMatr	Solver matrix file name
solverSolu	Solver solution file name
solverStat	Solver status file name
subSys	Name of subsystem configuration file
sysOut	Solver Status file reporting option
threads	Number of processors to be used by a solver
workFactor	Memory Estimate multiplier for some solvers
workSpace	Work space for some solvers in MB

4.40.5.3 Save and Restart Options

Option	Description
fSave	Creates a forced work file, i.e., the file is saved even if execution errors or other errors occured
restart	Name of a restart file, see The Save and Restart Feature
restartNamed	Name of another matching restart file, see Obfuscated Work Files
save	Creates a work file, see The Save and Restart Feature
saveObfuscate	Creates an obfuscated work file, see Obfuscated Work Files
symPrefix	Prefix all symbols encountered during compilation by the specified string in work file
xSave	Creates a compressed work file
xSaveObfuscate	Creates a compressed obfuscated work file

4.40.6 Detailed Descriptions of All Options

In this section we will give detailed descriptions of all options that may be used as command line parameters, in option statements or as model attributes.

Note

- We indicate for each entry the context in which the option is available.
- The options are listed in alphabetical order for easy reference.
- Synonyms apply only to options set via the command line.

action *(string)*: GAMS processing request

Synonym: A

Available: Command line

This option controls the way GAMS processes the input file. In particular GAMS currently processes the input file in multiple phases and this allows one to restrict the phases used. The two phases in order are:

- *Compilation* During this pass, the file is compiled, and syntax errors are checked for. Data initialization statements like scalar, parameter, and table statements are also processed during this stage.
- *Execution* During this stage, all execution time statements including assignments, loops, and solves are executed.

The special action `GT` is related to the creation of trace reports. See also option traceLevel for details.

Default: `CE`

Value	Meaning
`R`	Restart After Solve
`C`	CompileOnly
`E`	ExecuteOnly
`CE`	Compile and Execute
`GT`	Trace Report

appendExpand *(boolean)*: Expand file append option

Synonym: AE

Available: Command line

This option controls the manner of file opening of the option expand.

Default: `1`

Value	Meaning
`0`	Reset expand file
`1`	Append to expand file

appendLog *(boolean)*: Log file append option

Synonym: AL

Available: Command line

This option is used in conjunction with the setting of logOption to 2 and 4, where the log from the GAMS run is redirected to a file. Setting this option to 1 will ensure that the log file is appended to and not overwritten (replaced).

Default: `0`

Value	Meaning
`0`	Reset log file
`1`	Append to logfile

appendOut *(boolean)*: Output file append option

Synonym: AO

Available: Command line

Setting this option to 1 will ensure that the listing file is appended to and not overwritten (replaced).

Default: `0`

Value	Meaning
`0`	Reset listing file
`1`	Append to listing file

asyncSolLst *(boolean)*: Print solution listing when asynchronous solve (Grid or Threads) is used

Available: Command line, Option statement

This option determines whether the solution listing is printed in the listing file when an asynchronous (grid or threads) solve is used and the function handleCollect or command execute_loadHandle successfully collect the results.

Default: `0`

Value	Meaning
`0`	Do not print solution listing into lst file for asynchronous solves
`1`	Print solution listing into lst file for asynchronous solves

bRatio *(real)*: Basis detection threshold

Available: Command line, Option statement, Attribute statement (use before solve)

The `bRatio` value is used to detect whether the initial point (levels and marginals) passed to the solver represents a basis suitable for use by the solver.

Certain (pivotal) solution procedures can restart from an advanced basis that is constructed automatically using existing basis information, i.e. they do a *warm start*. This option is used to influence whether a warm start is done or not. GAMS provides a hint to the solver to suggest whether a basis can or might be extracted from the initial point. The hint is based on the number of rows with nonzero marginals and is computed internally as:

```
hint := (0 == bRatio) or (rowsWithNonzeroMarg > nRows * bratio)
```

Note that setting `bRatio` to 1 causes the hint to be false, while setting `bRatio` to 0 causes the hint to be true. Note also that this is only a hint to the solver: some solvers (e.g. barrier methods) do not use this hint value.

Default: `0.25`

captureModelInstance *(boolean)*: Switch to capture all model instances within a run

Available: Command line

This option is a debugging option that helps to capture, if set to 1, the model instances that are generated and solved during the execution of the `solve` commands in a GAMS run. This is particularly important if one needs to make a model instance available to GAMS technical support to investigate a solver failure or analyze poor solver performance. The model instances are stored in separate files with names `gams`*NNN*`.gms` where *NNN* is an increasing number. So sorting the files by time or name will correspond to the order in which these instances have been generated and solved. The format of the model instance is the GAMS scalar format and the instance is captured *before* the instance is solved by the selected solver, hence it contains the starting point but not the solution point.

Another way of capturing model instances becomes useful (and is triggered independent of this option) in case the user model does modularization by calling other GAMS programs that execute a solve statement via the GAMS executable. Since options are not automatically passed on to the GAMS subprocesses one might miss capturing some model instances. Moreover, due to constant folding and rounding in the scalar model format, the run of a scalar model instance might not reproduce a particular problematic issue of the original solve. Hence GAMS can capture the binary scratch files produced by the solve statement when executed under solveLink=0 in a ZIP file. This is activated by setting a system environment variable named `ZIPSCRDIR_PGAMS` (e.g. `set ZIPSCRDIR_PGAMS=mymodel`). The content of this environment variable is used as a prefix for the ZIP file names, e.g. `mymodel`*NNN*`.zip` where *NNN* is an increasing number. Such ZIP files can help GAMS support staff to execute the model and have an increased chance to reproduce a problematic issue.

Note

`CaptureModelInstance` cannot be used with solveLink=6/7. If this was set, GAMS will reset `solveLink` to `3/4` automatically.

Default: `0`

Value	Meaning
0	Do not capture model instances
1	Capture model instances

case *(boolean)*: Output case option for LST file

Available: Command line

This option controls the case of the text in the listing file.

Default: `0`

Value	Meaning
0	Write listing file in mixed case
1	Write listing file in upper case only

cErr *(integer)*: Compile time error limit

Available: Command line

The compilation will be aborted after `n` errors have occurred. By default, there is no error limit and GAMS compiles the entire input file and collects all the compilation errors that occur. If the file is too long and the compilation process is time consuming, `cerr` could be used to set to a low value while debugging the input file.

Default: `0`

Value	Meaning
0	No error limit (default)
n	Stop after n errors

charSet *(boolean)*: Character set flag

Available: Command line

This option specifies whether foreign language characters are permitted in comments and text items. For a list of standard GAMS characters, see table Legal Characters.

Default: `1`

Value	Meaning
0	Use limited GAMS characters set
1	Accept any character in comments and text items (foreign language characters)

cheat *(real)*: Cheat value, i.e. minimum solution improvement threshold

Available: Attribute statement (use before solve)

For a branch-and-bound based solver, each new feasible solution must be at least the value of `cheat` better than the current best feasible solution. Note that this may speed up the search, but may cause some solutions, including optimal ones, to be missed. If a model has been solved with a nonzero cheat value, then the optimal solution will be within the cheat value or less of the found solution. Observe that the option `cheat` is specified in absolute terms (like the option optCA), therefore non-negative values are appropriate for both minimization and maximization models. Note that using this option will invalidate any reporting of the dual bound or optimality gaps. Further, certain solver options can override the cheat setting, e.g., the Cplex option objDif, and some solvers may ignore the `cheat` option.

Default: `0`

checkErrorLevel *(boolean)*: Check errorLevel automatically after executing external program

Available: Command line, Option statement

If this option is set to 1, the errorLevel is checked implicitly after executing an external program or via execute, put_utility exec and put_utility shell. The same holds for the execution of a tool from the GAMS tools library via executeTool. An execution error is triggered and the execution is aborted, if it is not 0. So, with `checkErrorLevel = 1` the before mentioned statement behave like execute.checkErrorLevel, put_utility exec.checkErrorLevel, put_utility shell.checkErrorLevel, and executeTool.checkErrorLevel, respectively.

Also, if this option is set as a command line parameter it initializes the state of the dollar control option $on/offCheckErrorLevel. So, with `checkErrorLevel = 1`, the errorLevel is checked implicitly after $call and $hiddenCall. The same holds for calls to a tool from the GAMS tools library via $callTool and $hiddenCallTool. A compilation error is triggered and the compilation is aborted if that is not 0. So, the dollar control options mentioned before behave like $call.checkErrorLevel, $hiddenCall.checkErrorLevel, $callTool.checkErrorLevel, and $hiddenCallTool.checkErrorLevel respectively.

Default: `0`

Value	Meaning
`0`	Do not check errorLevel automatically after execution of external program
`1`	Check errorLevel automatically after execution of external program

CNS *(string)*: Constrained Nonlinear Systems - default solver

Available: Command line, Option statement

The default solver for models of the type Constrained Nonlinear Systems is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

connectIn *(string)*: Specify YAML Connect script file processed at start of GAMS

Available: Command line

The YAML file provided by this command line parameter is passed to and processed by Connect. This allows Connect to prepare data for import at the beginning of a GAMS job. The GAMS program itself does not need to know where the various data files (CSV, Excel, etc) are located. The supply side of the *data contract* is encapsulated in the YAML script.

Here is an example. The content of the file `p.yaml` that is supplied via the option `connectIn` could look like this:

```
- ExcelReader:
    file: myworkbook.xlsx
    symbols:
        - name: p
          range: Sheet1!A1
          rowDimension: 1
          columnDimension: 1
- GDXWriter:
    file: input.gdx
    symbols:
        - name: p
```

The GDX file `input.gdx` can be loaded at the beginning of a GAMS job using gdxLoad:

```
Set i, j;
Parameter p(i<,j<);
$gdxLoad input.gdx p
display p;
```

Alternatively, this can be nicely combined with the IDCGDXInput to load external input symbols. The GDX file `input.gdx` can be specified as value for `IDCGDXInput` (`gams mymodel connectIn=p.yaml IDCGDXInput=input.gdx`):

```
Set i, j;
$onExternalInput
Parameter p(i<,j<) / i1.j1 1 /;
$offExternalInput
display p;
```

The YAML code in the `connectIn` file can also utilize the `GAMSReader` agent. Normally, there are no symbols to read at the job's start, but in case the job *restarts* from an existing work file (see command line option restart) the symbols available from the work file can be read into the Connect database with the `GAMSReader` agent. The YAML code in the `connectIn` file can utilize the `GAMSWriter` agent. It will update the symbols and data read from the restart file at the beginning of the GAMS compilation phase.

connectOut *(string)*: Specify YAML Connect script file processed at end of GAMS

Available: Command line

The YAML file provided by this command line parameter is passed to and processed by Connect. This allows Connect to take the GAMS data at the end of the GAMS job and export it to various formats (CSV, Excel, etc) according to the instructions in the YAML file.

Here is an example. The content of the file `p.yaml` that is supplied via the option `connectOut` could look like this:

```
- GDXReader:
    file: output.gdx
    symbols:
        - name: p
- ExcelWriter:
    file: myworkbook.xlsx
    symbols:
        - name: p
          range: Sheet!A1
          columnDimension: 1
```

The GDX file `output.gdx` can be written at the end of a GAMS job using execute_unload:

```
Set i /i1*i10/, j /j1*j10/;
Parameter p(i,j); p(i,j) = uniform(0,1);
execute_unload 'output.gdx', p;
```

Alternatively, this can be nicely combined with the IDCGDXOutput to process external output symbols. The GDX file `output.gdx` can be specified as value for `IDCGDXOutput` (`gams mymodel connectOut=p.yaml IDCGDXOutput=output.gdx`):

```
Set i /i1*i10/, j /j1*j10/;
$onExternalOutput
Parameter p(i,j); p(i,j) = uniform(0,1);
$offExternalOutput
```

The YAML code in the `connectOut` file can also utilize the `GAMSReader` agent. This reads directly from GAMS memory (without the detour via GDX) and processes any symbol available at the end of the GAMS run. The YAML code in the `connectOut` file cannot utilize the `GAMSWriter` agent.

curDir *(string)*: Current directory

Synonym: CDir

Available: Command line

This option sets the current working directory. It is useful when GAMS is called from an external system like Visual Basic. If it is not specified, it will be set to the directory the GAMS module is called from.

cutOff *(real)*: Cutoff value for branch and bound

Available: Attribute statement (use before solve, reset by solve statement)

Within a branch-and-bound based solver, the parts of the tree with an objective value worse than the cutoff value are ignored. Note that this may speed up the initial phase of the branch and bound algorithm (before the first integer solution is found). However, the true optimum may be beyond the cutoff value. In this case the true optimum will be missed and moreover, no solution will be found.

Observe that this option is specified in absolute terms (like the option optCA).

Default: `0`

decimals *(integer)*: Decimal places for display statements

Available: Option statement

This option specifies the number of decimals that will be printed for numeric values that do not have a specific print format attached. The range is `[0,8]`.

Default: `3`

decryptKey *(string)*: Key to decrypt a text file that was encrypted via $encrypt

Available: Command line

This option provides a key to decrypt a GAMS input file that has been encrypted with a key provided by encryptKey and $encrypt. For more information, see Encrypting Files.

defPoint *(integer)*: Indicator for passing on default point

Available: Attribute statement (use before solve, reset by solve statement)

This option determines the point that is passed to the solver as a basis. By default, the levels and marginals from the current basis are passed to the solver. In some circumstances (mostly during debugging), it can be useful to pass a standard default input point, i.e. with all levels set to 0 or lower bound.

Value	Meaning
0	Pass user defined levels and marginals to solver
1	Pass default levels and marginals to solver
2	Pass default marginals to solver

dFormat *(integer)*: Date format

Synonym: DF

Available: Command line

This option controls the date format in the listing file. The three date formats correspond to various conventions used around the world. For example, the date December 2, 1996 will be written as `12/02/96` with the default `df` value of 0, as `02.12.96` with `df=1`, and as `96-12-02` with `df=2`.

Default: `0`

Value	Meaning
0	Date as mm/dd/yy
1	Date as dd.mm.yy
2	Date as yy-mm-dy

dictFile *(real)*: Force writing of a dictionary file if dictfile > 0

Available: Attribute statement (use before solve, reset by solve statement)

If this option is set to a value that is larger than zero, it will instruct GAMS to make the GAMS names of variables and equations that have been generated by the solve statement available to the solver. In many solver links these names are registered with the solver and hence messages from the solver that involve variables and equations (e.g. an infeasible row or duplicate columns) can be easily interpreted by the user. Consider the following example:

```
Row 'demand(new-york)' infeasible, all entries at implied bounds.

Duplicate columns x(san-diego.new-york) and x(san-diego.chicago) make problem unbounded.
```

If we have `modelname.dictfile=0` the same messages will read as follows:

```
Row 'c4' infeasible, all entries at implied bounds.

Duplicate columns x4 and x5 make problem unbounded.
```

Sometimes a dictionary is required for a successful run. Some solver option use the original GAMS names and need to be matched with the variables `1..n` and equations `1..m` in the solver. The dictionary file with its API allows to calculate such a mapping. Note that this is done automatically inside the solver links, so users do not need to be concerned with it.

For example, in the indicator constraints implementation a binary indicator variable is matched to a constraint. In the model [**INDIC01**] from the GAMS test library, this matching is done in the following GAMS/Cplex option file `cplex.opt`:

```
indic eq3(dice,f,fp)$comp(dice,f,fp) 1
```

Observe that if no dictionary is available, we will get an error:

```
**** Unable to read dictionary file required for indicator constraints
```

However, the dictionary comes at a price. Generating the names and calculating and storing the map takes time and space. In addition, GAMS names take up space in the solver. Thus, if the user needs very fast generation and does not need names, setting `dictFile` to zero is a good option.

Further, note that some solvers allow to suppress the loading of names (using the solver option `names=no`). Suppressing the loading of names facilitates to use the name mapping features required for models like `INDIC01` above, but does not load the names into the solver name space for better reporting (and hence saves some space).

digit *(string)*: Switch default for "$on/offDigit"

Available: Command line

For more info see $on/offDigit.

Default: `off`

Value	Meaning
`off`	Activate $offDigit
`on`	Activate $onDigit

dispWidth *(integer)*: Number of characters to be printed in the column labels of all subsequent display statements

Available: Option statement

This option controls the number of characters that are shown for a label in a column in the context of the display statement. Consider the following example:

```
Set i / thislabelhas24characters /;
Parameter p(i,i) / thislabelhas24characters.thislabelhas24characters 2/;
display p;

option dispWidth=24;
display p;
```

The two display statements in this code will generate the following output:

```
----      3 PARAMETER p

                           thislabel~

thislabelhas24characters       2.000

----      6 PARAMETER p

                           thislabelhas24characters

thislabelhas24characters                    2.000
```

Observe that in the first display, the label in the column is cut off after 10 characters, while in the second display it is shown in full.

Note that the default value is 10 and the range is `[10,31]`.

Default: `10`

dmpOpt *(no value)*: Debugging option: causes GAMS to echo the runtime option settings

Available: Option statement

This debugging option has the effect that all available option statements and their current values are listed in the listing file.

dmpSym *(no value)*: Debugging option: causes GAMS to echo the symbol table to the listing file

Available: Option statement

This debugging option is especially useful for diagnosing memory problems. It has the effect that GAMS will report the number of elements that are stored for each identifier at the point in the program where this option is inserted together with a memory estimate. The report that is generated in this way is called a *memory dump*. This is similar to the option dmpUserSym, but prints GAMS internal symbols and information as well. For details, see section Finding the Causes for Excessive Memory Use.

dmpUserSym *(no value)*: Debugging option: causes GAMS to echo the symbol table to the listing file for user defined symbols only

Available: Option statement

This debugging option is especially useful for diagnosing memory problems. It has the effect that GAMS will report the number of elements that are stored for each identifier at the point in the program where this option is inserted together with a memory estimate. The report that is generated in this way is called a *memory dump*. This is similar to the option dmpSym, but prints user defined symbols only and also leaves out some very technical information which are mostly for internal use. For details, see section Finding the Causes for Excessive Memory Use.

DNLP *(string)*: Non-Linear Programming with Discontinuous Derivatives - default solver

Available: Command line, Option statement

The default solver for models of the type Nonlinear Programs with Discontinuous Derivatives is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

docFile *(string)*: Filename stem for documentation files

Available: Command line

domLim *(integer)*: Domain violation limit solver default

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the maximum number of domain errors (undefined operations like division by zero) a nonlinear solver will perform, while calculating function and derivative values, before it terminates the run and returns solver status `5 EVALUATION ERROR LIMIT`. Nonlinear solvers have difficulties recovering after attempting an undefined operation. Note that some solvers operate in a mode where trial function evaluations are performed. These solvers will not move to points at which evaluation errors occur, thus the evaluation errors at trial points are not counted against the limit.

Default: ∞

domUsd *(integer)*: Number of domain violations

Available: Attribute statement (use after solve)

This model attribute returns the number of domain violations after a solve.

dualCheck *(integer)*: Output on the reduced cost condition

Available: Option statement

If this option is set to 1, the reduced cost condition for each variable in the column listing will be evaluated using the equation marginals. The default value is zero, which means that the calculation will be omitted.

Default: `0`

dumpOpt *(integer)*: Writes preprocessed input to the file input.dmp

Available: Command line

This option with value larger than 9 creates a GAMS input file of that will reproduce results encapsulating all `include` files into one GAMS file. If activated, a file will be written containing GAMS source code for the entire problem. The file name is the input file name with the extension `dmp`. For values smaller than 10, this option tries to encapsulate all the items from a restart file that are needed to execute a solve.

For the values smaller than 10, consider the following example. We will split the transportation model [**TRNSPORT**] into two files and run them with the save and restart feature. Then we will illustrate the option `dumpOpt`. The first file called `trans1.gms` contains the first part of the model up to and including the model statement:

```
Sets
       i   canning plants   / seattle, san-diego /
       j   markets          / new-york, chicago, topeka / ;

Parameters
       a(i)  capacity of plant i in cases
         /    seattle     350
              san-diego   600  /

       b(j)  demand at market j in cases
         /    new-york    325
              chicago     300
              topeka      275  / ;

Table d(i,j)  distance in thousands of miles
                    new-york       chicago      topeka
      seattle          2.5           1.7          1.8
      san-diego        2.5           1.8          1.4  ;

Scalar f  freight in dollars per case per thousand miles  /90/ ;

Parameter c(i,j) transport cost in thousands of dollars per case ;
c(i,j) = f * d(i,j) / 1000 ;

Variables
    x(i,j)  shipment quantities in cases
    z       total transportation costs in thousands of dollars ;
Positive Variable x ;
```

```
Equations
    cost        define objective function
    supply(i)   observe supply limit at plant i
    demand(j)   satisfy demand at market j ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

model transport /all/ ;
```

Note that we removed all comments for brevity. The second file called `trans2.gms` contains the solve statement and the display statement:

```
solve transport using lp minimizing z ;
display x.l, x.m ;
```

We run the first file with the command line parameter save and thus generate a work file. For details on work files, see chapter The Save and Restart Feature. Then we run `trans2.gms` restarting from the saved work file. The result will be equivalent to running the original model [**TRNSPORT**].

Note

The option `dumpOpt` can only be used effectively, if the first line in the second file, `trans2.gms`, is the solve statement.

Now, we will illustrate the use of the option `dumpopt`, by running the second file with the following command:

```
> gams trans2 r=trans dumpopt=1
```

Here `trans` is the name of the saved files generated from the file `trans1.gms`. As a result of this call, a new file will be created. It is called `trans2.dmp` and has the following content:

```
* This file was written with DUMPOPT=1 at 11/25/21 15:15:17
*
*      INPUT = C:\tmp\trans2.gms
*       DUMP = C:\tmp\trans2.dmp
*    RESTART = C:\tmp\0.g0?
*
*           with time stamp of 11/16/21 17:18:53
*
* You may have to edit this file and the input file.

* There are 5 labels

Set WorkFileLabelOrder dummy set to establish the proper order /
   seattle,san-diego,new-york,chicago,topeka /;

Model transport;

Variable z 'total transportation costs in thousands of dollars';

Set i(*) 'canning plants' /
   seattle,san-diego /;

Set j(*) 'markets' /
```

```
   new-york,chicago,topeka /;

Parameter c(i,j) 'transport cost in thousands of dollars per case' /
   seattle.new-york 0.225,seattle.chicago 0.153,seattle.topeka 0.162,san-diego.new-york 0.225,sa

Positive Variable x(i,j) 'shipment quantities in cases';

Parameter a(i) 'capacity of plant i in cases' /
   seattle 350,san-diego 600 /;

Parameter b(j) 'demand at market j in cases' /
   new-york 325,chicago 300,topeka 275 /;

Equation demand(j) 'satisfy demand at market j';

Equation supply(i) 'observe supply limit at plant i';

Equation cost 'define objective function';

*      *** EDITS FOR INPUT FILE ***

*** END OF DUMP ***
```

Note that all the data that enters the model in the solve statement has been regenerated. Observe that the parameter `d` has not been regenerated since it does not appear in the model. Changing the value of the parameter `dumpopt` will have the effect that other names are used for the identifiers in the regenerated file, see table below.

Note

If $onVerbatim is active, `DumpOpt = 11` behaves like `DumpOpt = 21` (comments are kept)

See also dumpOptGDX.

Default: `0`

Value	Meaning
0	No dumpfile
1	Extract referenced data from the restart file using original set element names
2	Extract referenced data from the restart file using new set element names
3	Extract referenced data from the restart file using new set element names and drop symbol text
4	Extract referenced symbol declarations from the restart file
11	Write processed input file without comments
21	Write processed input file with all comments
22	Write processed input with all comments into a separate dump file for each block

dumpOptGDX *(string)*: Defines a GDX file name stem created when using DumpOpt

Available: Command line

This parameter works together with the dumpOpt parameter. If that is set to `1`, `11` or `21` while `dumpOptGDX` is set, GAMS will create a GDX file with the data used in the dump file instead of data statements, and the dump file will load the data from that GDX file. So, with the parameter `dumpOptGDX=trns2` the example above would generate the following dump file:

```
* This file was written with DUMPOPT=1 at 11/25/21 15:14:11
*
*      INPUT = C:\tmp\trans2.gms
*       DUMP = C:\tmp\trans2.dmp
*    RESTART = C:\tmp\0.g0?
* DUMPOPTGDX = trans2
*
*           with time stamp of 11/16/21 17:18:53

$gdxIn trans2.gdx

*
* You may have to edit this file and the input file.

* There are 5 labels

Set WorkFileLabelOrder dummy set to establish the proper order;
$loadDCR WorkFileLabelOrder=*

Model transport;

Variable z 'total transportation costs in thousands of dollars';

Set i(*) 'canning plants';
$loadDCR i

Set j(*) 'markets';
$loadDCR j

Parameter c(i,j) 'transport cost in thousands of dollars per case';
$loadDCR c

Positive Variable x(i,j) 'shipment quantities in cases';

Parameter a(i) 'capacity of plant i in cases';
$loadDCR a

Parameter b(j) 'demand at market j in cases';
$loadDCR b

Equation demand(j) 'satisfy demand at market j';

Equation supply(i) 'observe supply limit at plant i';

Equation cost 'define objective function';

$gdxIn

*      *** EDITS FOR INPUT FILE ***

*** END OF DUMP ***
```

Note that more than one GDX file could be created with a common file stem defined by this parameter. This will be necessary if the source model has multiple data definitions for the same symbol like in this simple example:

```
Set       i    / i1*i3/;
Parameter a(i) / i1 1, i2  2/;
$onMulti
```

```
Parameter a(i) / i1 0, i3  3/;
Scalar    b    /  7 /;
Scalar    b    / 77 /;
```

Running this as

```
   > gams dummy dumpOpt=11 dumpOptGDX=dummyIn
```

will generate this dump file:

```
* This file was written with DUMPOPT=11 at 11/25/21 15:26:02
*
*       DUMP = C:\tmp\dummy.dmp
*      INPUT = C:\tmp\dummy.gms
*    RESTART =
* DUMPOPTGDX = dummyIn
*

$gdxIn dummyIn.gdx

Set       i
$loadDCR i
;
Parameter a(i)
$loadDCR a
;
$onMulti
Parameter a(i)
$gdxIn dummyIn_m1.gdx
$loadDCR a
;
Scalar    b
$loadDCR b
;
Scalar    b
$gdxIn dummyIn_m2.gdx
$loadDCR b
;
* *** EXIT       C:\Data\gspTest\dummy.gms

$gdxIn
*** END OF DUMP ***
```

And this is the data found in the GDX files:

dummyIn.gdx:

```
Set i(*) /
'i1',
'i2',
'i3' /;

Parameter a(*) /
'i1' 1,
'i2' 2 /;
```

dummyIn_m1.gdx:

```
Parameter a(*) /
'i2' 2,
'i3' 3 /;

Scalar b / 7 /;
```

dummyIn_m2.gdx:

```
Scalar b / 77 /;
```

dumpParms *(integer)*: GAMS parameter logging

Synonym: DP

Available: Command line

This option lists the settings of all command line parameters that were changed or set by the user, GAMS or an IDE during the current run. Note that with `dp=2` all file operations are listed, including the full path of each file on which any operation is performed.

Default: `0`

Value	Meaning
0	No logging
1	Lists accepted/set parameters
2	Log of file operations plus list of accepted/set parameters

dumpParmsLogPrefix *(string)*: Prefix of lines triggered by DumpParms>1

Synonym: DPLP

Available: Command line

This option prefixes in the log file the list of all command line parameters that were changed or set by the user, GAMS or an IDE during the current run. Note that the option dumpParms must be greater than 1 for `dumpParmsLogPrefix` to have an effect.

Default: ***

ECImplicitLoad *(string)*: Allow implicit loading of symbols from embedded code or not

Available: Command line, Option statement

The command line parameter ECImplicitLoad initializes both, the option ECImplicitLoad and the dollar control option $on/offECImplicitLoad for the compile-time equivalent behavior.

Default: `on`

Value	Meaning
off	Do not allow implicit loading from embedded code
on	Allow implicit loading from embedded code

eject *(no value)*: Inject a page break into the LST file

Available: Option statement

This option has the effect that a page break is inserted in the listing file.

EMP *(string)*: Extended Mathematical Programs - default solver

Available: Command line, Option statement

The default solver for models of the type Extended Mathematical Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

empty *(string)*: Switch default for "$on/offEmpty"

Available: Command line

For more info see $on/offEmpty.

Default: `on`

Value	Meaning
`off`	Activate $offEmpty
`on`	Activate $onEmpty

encryptKey *(string)*: Key to encrypt a text file using $encrypt

Available: Command line

This option provides a key to encrypt a text file via $encrypt. For more information, see Encrypting Files.

eolCom *(string)*: Switch default for "$on/offEolCom" and "$eolCom"

Available: Command line

If this is set to on `on` or `off` it sets the state of $on/offEOLCom. Other strings (with not more than two characters) will set $EOLCom.

Default: `off`

Value	Meaning
`off`	Activate $offEolCom
`on`	Activate $onEolCom using default EOL comment character
`other`	Activate $onEolCom setting specific EOL comment character(s)

eolOnly *(integer)*: Single key-value pairs (immediate switch)

Synonym: EY

Available: Command line

This option controls formatting of parameters on the command line and is useful in conjunction with the option parmFile.

This option acts as an immediate switch that forces only one option-value pair to be read on a line. Note that by default, any number of option-value pairs may be present on the same line and termination characters and quoting is necessary to determine the end of key/value pair. With this option active the remainder after the key is used as the value independent of quoting or termination characters.

Default: `0`

Value	Meaning
`0`	Any number of keys or values
`1`	Only one key-value pair on a line

epsToZero *(string)*: Treat eps as zero when unloading to GDX

Available: Command line, Option statement

When this is set to `on`, `EPS` values are written as zero when unloading parameters or variable and equation levels to GDX at execution time. The command line parameter `EpsToZero` initializes both, the option `EpsToZero` and the dollar control option $on/offEpsToZero for the compile-time equivalent behavior.

Default: `off`

Value	Meaning
`off`	Treat Eps as Eps
`on`	Treat Eps as Zero

errMsg *(integer)*: Placing of compilation error messages

Available: Command line

This option controls the position of the compilation error messages in the listing file. To illustrate the option, consider the following slice of GAMS code:

```
Set       i     / 1*10 / ;
Set       j(i)  / 10*11 /;
Parameter a(jj) / 12 25.0 / ;
```

After running this code, the listing file will contain the following lines:

```
   1  Set       i     / 1*10 / ;
   2  Set       j(i)  / 10*11 /;
****                         $170
   3  Parameter a(jj) / 12 25.0 / ;
****               $120
   4

120  Unknown identifier entered as set
170  Domain violation for element

**** 2 ERROR(S)   0 WARNING(S)
```

Note that numbers $170 and $120 flag the two errors as they occur, but the errors are explained only at the end of the compilation output. However, if the code is run using the option errmsg=1, the resulting listing file will contains the following:

```
   1  Set       i     / 1*10 / ;
   2  Set       j(i)  / 10*11 /;
****                        $170
**** 170  Domain violation for element
   3  Parameter a(jj) / 12 25.0 / ;
****               $120
**** 120  Unknown identifier entered as set
   4
**** 2 ERROR(S)   0 WARNING(S)
```

Observe that the explanation for each error is provided immediately following the error marker.

Default: 1

Value	Meaning
0	Place error messages at the end of compiler listing
1	Place error messages immediately following the line with the error
2	Suppress error messages

errNam *(string)*: Name of error message file

Available: Command line

This option specifies the name of a file defining the internally used compiler error messages. It is used to change the name from the default name gamserrs.txt.

error *(string)*: Force a compilation error with message

Available: Command line

This option forces a parameter error with a specified message. It is useful in the context of incorporating a GAMS file within another batch file where the user needs to have control over the conditions when GAMS is called. See also section Conditional Compilation.

To illustrate, the default GAMS log file from running a model with the option error=Hallo will look as follows:

```
gams: **** Error: Parameter error(s)
    :             Reading parameter(s) from "command line"
    :             *** Error Hallo
    :             Finished reading from "command line"
```

errorLog *(integer)*: Max error message lines written to the log for each error

Synonym: ER

Available: Command line

This option controls the number of error message lines that are written to the log file.

Under GAMS Studio, the default is reset to 99.

Default: 2147483647

Value	Meaning
0	No error messages to LOG file
n	Number of lines for each error that will be written to LOG file

etAlg *(real)*: Solver dependent timing information

Available: Attribute statement (use after solve)

Unlike etSolve and etSolver, this attribute is set by the individual solver links. If not set, it defaults to NA. This attribute was intended to allow solvers to return the elapsed time used by the solve algorithm without including any model generation, communication, or setup time. However, solvers are free to adapt this convention and return time-related information (but not necessarily elapsed time) for executing the solve algorithm. Please inspect your solver manual for the actual meaning of the value returned in this attribute.

etLim *(real)*: Elapsed time limit in seconds

Synonym: ETL

Available: Command line

This option controls the time limit for a GAMS job. The system will terminate with a compilation or execution error if the limit is reached. A GAMS job will terminate if the elapsed time in seconds exceeds the value of etLim. If a solve statement gets executed, and the resLim for the model to be solved is greater than etLim - timeElapsed, resLim will be reduced automatically.

Default: ∞

etSolve *(real)*: Elapsed time it took to execute a solve statement in total

Available: Attribute statement (use after solve)

This model attribute returns the elapsed time it took to execute a solve statement in total. This time includes the model generation time, the time to read and write files, the time to create the solution report and the time taken by the actual solve. The time is expressed in seconds of wall-clock time.

etSolver *(real)*: Elapsed time taken by the solver only

Available: Attribute statement (use after solve)

This model attribute returns the elapsed time taken by the solver only. This does not include the GAMS model generation time and the time taken to report and load the solution back into the GAMS database. The time is expressed in seconds of wall-clock time.

execMode *(integer)*: Limits on external programs that are allowed to be executed

Available: Command line

A higher value denotes a more restrictive alternative. If the restriction level n is chosen, then the restriction levels less than n will also be active. For example, if restriction level 3 is chosen, then restrictions 2 and 1 will apply too.

Default: 0

Value	Meaning
0	Everything allowed
1	Interactive shells in $call and execute commands are prohibited
2	Embedded Code and all $call and execute commands are prohibited
3	$echo or put commands can only write to directories in or below the working or scratchdir
4	$echo and put commands are not allowed

expand *(string)*: Expanded (include) input file name

Synonym: EF

Available: Command line

This option generates a file that contains information about all the input files processed during a particular compilation. By default, the names of the input files are composed by completing the name with the current directory.

Consider the following exmaple:

```
$call rm expfile.txt
$onecho > file1.inc
a = a*2; display a;
$include file2.inc
$offecho
$onecho > file2.inc
a = a+1; display a;
$include file3.inc
$offecho
$onecho > file3.inc
a = a**2; display a ;
$offecho
parameter a / 1 /;
$include file3.inc
$include file2.inc
$include file1.inc
```

If the model is run with the command line parameter `ef=expfile.txt`, a file called `expfile.txt` will be generated. This file will contain the following lines:

```
1 INPUT       0     0        0        1       29  C:\GAMS\Examples\expand.gms
2 CALL        0     1        1        1        1  rm expfile.txt
3 INCLUDE     1     1       14       14       15  C:\GAMS\Examples\file3.inc
4 INCLUDE     1     1       15       16       19  C:\GAMS\Examples\file2.inc
5 INCLUDE     2     4        2       18       19  C:\GAMS\Examples\file3.inc
6 INCLUDE     1     1       16       20       25  C:\GAMS\Examples\file1.inc
7 INCLUDE     2     6        2       22       25  C:\GAMS\Examples\file2.inc
8 INCLUDE     3     7        2       24       25  C:\GAMS\Examples\file3.inc
9 EXIT        0     1       20       29       29  C:\GAMS\Examples\expand.gms
```

Note that the first row always refers to the parent file, in this case the file `expand.gms`. The first column gives the sequence number of the input files that were encountered. The second column refers to the type of file that is referenced. The following file types are possible:

```
0   INPUT
1   EXIT
2   INCLUDE
3   BATINCLUDE
4   LIBINCLUDE
5   SYSINCLUDE
6   CALL
7   CALL.ASYNC
7   CALLTOOL
8   GDXIN
9   GDXOUT
10   IF EXIST
11   IF DEXIST
12   FUNCLIBIN
13   TERMINATE
14   STOP
```

Observe that `$call` is also listed. The third column describes the depth for nested include files. The fourth column provides the sequence number of the parent file for the file being referenced. The fifth column gives the local line number in the parent file where the dollar control option $include appeared. The sixth column gives the global (expanded) line number which contained `$include`. The seventh column provides the total number of lines in the file after it is processed. The last column provides the name of the file.

Note that the listing in the expand file is similar to the include file summary in the listing file of the model. And like the include file summary, this file will not be written, if $offInclude is set in the model.

fdDelta *(real)*: Step size for finite differences

Available: Command line, Option statement, Attribute statement (use before solve)

This option allows users to control the step size while the numerical Hessian and numerical derivatives are computed in the context of the function suffixes `.hessn` and `.gradn`. For functions with one argument, GAMS evaluates the function at `f(x-d)` and `f(x+d)` for the numerical gradient. If function values are used for the numerical Hessian, GAMS will evaluate at `f(x-2d)`, `f(x)` and `f(x+2d)`. For functions with multiple arguments, the same calculations are performed for the components of the input argument vector.

Default: `1.0E-05`

fdOpt *(integer)*: Options for finite differences

Available: Command line, Option statement, Attribute statement (use before solve)

This option allows users to control how numerical derivatives are computed. The values provide choice regarding the scaling of steps, Hessian calculation method and the use of numerical first derivatives.

Default: `0`

Value	Meaning
0	All derivatives analytically, for numerical Hessian use gradient values, scale delta
1	All derivatives analytically, for numerical Hessian use function values, scale delta
2	Gradient analytically, force Hessian numerically using gradient values, scale delta
3	Gradient analytically, force Hessian numerically using function values, scale delta
4	Force gradient and Hessian numerically, scale delta
10	Same as 0, but no scale of delta
11	Same as 1, but no scale of delta
12	Same as 2, but no scale of delta
13	Same as 3, but no scale of delta

fErr *(string)*: Alternative error message file

Available: Command line

This option redirects the compilation error messages to a file and names the file. By default, the file name is composed by completing the name with the scratch directory and the scratch extension. Note that under default settings such a file with compilation error messages is not generated. This option can be used when GAMS is being integrated into other environments like Visual Basic. The error messages that are reported in the listing file may be extracted with this option and their display may be controlled from the environment that is calling GAMS.

To illustrate, consider the slice of GAMS code that we used to explain the option errMsg. If we call this code with the command line parameter `ferr=myfile.err`, a file called `myfile.err` will be created in the scratch directory. This file will contain the following lines:

```
0     0      0      0 D:\GAMS\NEW.LST
1     1    170     31 D:\GAMS\NEW.GMS
2     2    120     14 D:\GAMS\NEW.GMS
```

Note that the first column refers to the global row number of the error in the listing file. The second column refers to the row number of the error in the individual file where the problem occured. This will be different from the first column only if the error occured in an include file. In this case, the second column will contain the line number in the include file where the error occurred, while the first number will contain the global line number (as reported in the listing file) where the error occured. The number in the third column refers to the error number of the compilation error. The fourth number refers to the column number of the error in the source file. The last column contains the individual file in which the error occurred.

fileCase *(integer)*: Casing of file names and paths (put, gdx, ref, $include, etc.)

Available: Command line

This option facilitates modifying the case of file names and paths. It applies to files created by a GAMS Job such as for example put files, GDX files and reference files. Under Windows, the casing of a created file will only be affected if the file does not yet exist, i.e. `fileCase=2` won't create `trnsport.ref` if there is already a file `TRNSPORT.ref`. It should be noted that `fileCase` also applies to existing files that are used but not created by a GAMS job such as for example $include files or $batInclude files.

Note that many other file names and paths are affected by this option (e.g. the scratch directory (scrDir) or the GAMS system directory (sysDir)) and that it is recommended to use this option with caution.

Default: `0`

Value	Meaning
0	Causes GAMS to use default casing
1	Causes GAMS to upper case file names including path of the file
2	Causes GAMS to lower case file names including path of the file
3	Causes GAMS to upper case file names only (leave the path alone)
4	Causes GAMS to lower case file names only (leave the path alone)

fileStem *(string)*: Sets the file stem for output files which use the input file name as stem by default

Available: Command line

By default, some output files use the input file name as base. If the names of these output files were not set explicitly, then this option may be used to set another name than the input file name as base for these output files. In particular, the names for the following files may be set with `fileStem`: dump files (see option dumpOpt), GDX files (if the option GDX was set to `default`), log files (see option logFile), lst files (see option output), reference files (if the option reference was set to `default`) and trace summary files (see also option trace).

fileStemApFromEnv *(string)*: Append a string read from an environment variable to the "FileStem"

Available: Command line

This option for users that submit GAMS job via mpirun/mpiexec. Such commands will spawn multiple instances of GAMS (the precise number is an argument to mpirun/mpiexec). Each invokation of GAMS will run the identical job and only the contents of an environment variable (`PMI_RANK`) will differentiate the run. Since GAMS will normally write to `modelname.log/lst` if we run the GAMS file `modelname.gms` we will have many jobs writing to the same file. Therefore we use this option to append the content of a particular environment variable (name given by this option) to the default file names (see fileStem). Hence GAMS will create `modelname0.log/lst`, `modelname1.log/lst`, and so forth when started with mpirun/mpiexec and `fileStemApFromEnv` is set to the environment variable that provides the MPI rank of the invokation. We allow to specify the name of the environment variable because different MPI implementations use different variable names (e.g. `PMI_RANK` or `OMPI_COMM_WORLD_RANK`).

filtered *(string)*: Switch between filtered and domain-checked read from GDX

Available: Command line, Option statement

The command line parameter Filtered initializes both, the option Filtered to control the behvior of `gdxLoad` and the dollar control option $on/offFiltered for the compile-time equivalent behavior.

Default: `on`

Value	Meaning
`off`	Load domain checked
`on`	Load filtered

forceOptFile *(integer)*: Overwrites other option file section mechanism

Available: Command line

Default: `0`

forceWork *(boolean)*: Force GAMS to process a save file created with a newer GAMS version or with execution errors

Synonym: FW

Available: Command line

Work files generated by GAMS using the command line parameter save are saved in binary format. The information inside these files will change from one GAMS version to another GAMS version. GAMS makes every attempt to be backward compatible and ensure that all new GAMS systems are able to read save files generated by older GAMS systems. However, this cannot be guaranteed the other way around. Thus, GAMS does not allow to process a save file, that was generated by a newer version of GAMS in general. This can be changed by setting `forceWork` to 1. Also, GAMS does not continue after processing a save while with an execution error by default. This behavior can be changed as well by setting `forceWork=1`.

Attention

Forcing GAMS to continue after reading a save file from a newer GAMS version or a save file with execution errors may lead to unexpected behavior. If it is about the GAMS version and you can recreate the save file you may write it using GAMS command line parameter previousWork=1.

Default: `0`

Value	Meaning
`0`	No translation
`1`	Try translation

forLim *(integer)*: GAMS looping limit

Available: Command line, Option statement

This option specifies the maximum number of permitted executions of control structures with a for statement, a while statement or a repeat statement before GAMS signals an execution error and terminates the control structure.

Default: ∞

fSave *(boolean)*: Creates a forced work file, i.e., the file is saved even if execution errors or other errors occured

Available: Command line

This option allows to save a file even in the face of execution or other errors. How it works depends on the command line parameter save.

Note that the option value of 1 is mainly used by solvers that can be interrupted from the terminal.

Default: `0`

Value	Meaning
`0`	Workfile only written to file specified by SAVE if no errors occur
`1`	Workfile always written to file specified by SAVE or if SAVE is not present to a name made up by GAMS

G205 *(integer)*: Use GAMS version 2.05 syntax

Available: Command line

This option sets the level of the GAMS syntax and is mainly used to ensure backward compatibility. New keywords have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when they are run with the latest version of GAMS. This option enables users to run such old models.

For example, the word "if" is a keyword in GAMS that was introduced with the first version of Release 2.25. Setting the option `g205=1` allows the word "if" to be used as an identifier since it was not a keyword in Release 2.05. As another example, the word "for" is a keyword in GAMS that was introduced with the later versions of Release 2.25. Setting the option `g205=2` allows "for" to be used as an identifier since it was not a keyword in the first version of Release 2.25.

Note

If the values 1 or 2 are specified for option `g205`, then it will not be permitted to use enhancements of the GAMS language that were introduced in later versions.

Default: `0`

Value	Meaning
`0`	Use only latest syntax
`1`	Allow version 2.05 syntax only
`2`	Allow version 2.25 syntax only

GDX *(string)*: GAMS data exchange file name

Available: Command line

This option specifies the name of the GAMS data exchange file and causes a GDX file to be written hat contains all data in the model at the end of the job. Setting `gdx` to the string 'default' causes GAMS to create a GDX file with the gms file root name and a gdx extension. Thus gams the call

```
> gams trnsport gdx=default
```

will cause GAMS to write the GDX file `trnsport.gdx`.

gdxCompress *(boolean)*: Compression of generated GDX file

Available: Command line

This option specifies whether the GDX files are compressed or not.

Default: `0`

Value	Meaning
`0`	Do not compress GDX files
`1`	Compress GDX files

gdxConvert *(string)*: Version of GDX files generated (for backward compatibility)

Available: Command line

This option specifies in which format the GDX files will be written.

Default: `v7`

Value	Meaning
`v5`	Version 5 GDX file, does not support compression
`v6`	Version 6 GDX file
`v7`	Version 7 GDX file

gdxSymbols *(string)*: Select symbols that get exported when command line parameter GDX is set

Available: Command line

This option specifies which symbols get exported when command line parameter GDX is set. By default, all symbols get written to the specified GDX file, but it can be limited to only the ones that were created or updated during the model run, or even to just the ones changed during execution time.

Default: `all`

Value	Meaning
`all`	Write all symbols
`newOrChanged`	Write only symbols that are new or got modified during compile or execution time
`assigned`	Write only symbols that got modified during execution time

gdxUels *(string)*: Unload labels or UELs to GDX either squeezed or full

Available: Command line, Option statement

This option specifies the UEL export mode. Note that `gdxUels` only works for execute_unload* statements during execution time and not for the command line parameter GDX or dollar control options $gdxOut/$unload/$gdxUnload.
The UEL table may be written to a GDX file in two different modes. In `squeezed` mode, only the UELs that are required by the exported symbols are exported. In `full` mode, all UELs are exported. The following code snippet illustrates the difference:

```
Set       i    / i1*i5 /;
Parameter p(i) / i3 3 /;

option gdxuels = squeezed;
execute_unload 'squeezed' p;
execute 'gdxdump squeezed UelTable=i';

option gdxuels = full;
execute_unload 'full' p;
execute 'gdxdump full UelTable=i';
```

The file `squeezed.gdx` will contain the following lines:

```
Set i /
  'i3' /;

Parameter p(*) /
'i3' 3 /;
```

The file `full.gdx` on the other hand, will contain the following lines:

```
Set i /
  'i1' ,
  'i2' ,
  'i3' ,
  'i4' ,
  'i5' /;

Parameter p(*) /
'i3' 3 /;
```

Default: `squeezed`

Value	Meaning
`Squeezed`	Write only the UELs to Universe, that are used by the exported symbols
`Full`	Write all UELs to Universe

gridDir *(string)*: Grid file directory

Synonym: GDir

Available: Command line

This option sets the grid file directory. Note that each GAMS job has only one grid file directory.

gridScript *(string)*: Grid submission script

Synonym: GScript

Available: Command line

This option provides the name of a script file that is used to submit grid computing jobs. If only the file name is given the file is assumed to be located in the system directory. A fully qualified name can be given as well. The script needs to be similar to the file `gmsgrid.cmd` on Windows machines with arguments that give name and location of the solver executable, the solver control file name and the name of the scratch directory. For an example of such a script, see section The Grid Facility: Architecture and Customization. However, note that advanced knowledge of how GAMS sets up and calls solvers is needed for successful use.

Default: `gmsgrid`

handle *(real)*: Unique handle number of SOLVE statement

Available: Attribute statement (use after solve)

The model attribute `handle` contains a unique identification of each submitted solution request and is typically stored in a parameter defined over a set that covers all model instances. The handle number may be used by the functions handleCollect, handleStatus, handleDelete, handleSubmit and readyCollect. For details see chapter The Grid and Multi-Threading Solve Facility.

heapLimit *(real)*: Maximum Heap size allowed in MB

Synonym: HL

Available: Command line

This option allows to limit the amount of memory a GAMS job may use during compilation and execution. If the needed data storage exceeds this limit, the job will be terminated.

Default: ∞

holdFixed *(boolean)*: Treat fixed variables as constants

Available: Command line, Attribute statement (use before solve)

This option facilitates treating fixed variables as constants. Thus the problems size may be reduced.

Default: `0`

Value	Meaning
0	Fixed variables are not treated as constants
1	Fixed variables are treated as constants

holdFixedAsync *(boolean)*: Allow HoldFixed for models solved asynchronously as well

Available: Command line, Option statement

By default, holdFixed is automatically deactivated if a model is solved asynchronously, since this could lead to inconsistent solutions otherwise. To allow this anyway, this parameter can be set to 1.

Default: 0

Value	Meaning
0	Ignore HoldFixed setting for async solves
1	Allow HoldFixed for async solves

IDCGDXInput *(string)*: GDX file name with data for implicit input

Available: Command line

Specify the name of a GDX file that is used for implicit loading of input data. Details about this are described with $onExternalInput.

IDCGDXOutput *(string)*: GDX file name for data for implicit output

Available: Command line

Specify the name of a GDX file that is used for implicit writing of output data. Details about this are described with $onExternalOutput.

IDCGenerateGDX *(string)*: Specify GDX file name of input and output side of data contract

Available: Command line

Specify the name of a GDX file that is used to store all symbols that are declared as external input or external output at the end of a GAMS run.

IDCGenerateGDXInput *(string)*: Specify GDX file name of input side of data contract

Available: Command line

Specify the name of a GDX file that is used to store all symbols that are declared as external input at the end of a GAMS run.

IDCGenerateGDXOutput *(string)*: Specify GDX file name of output side of data contract

Available: Command line

Specify the name of a GDX file that is used to store all symbols that are declared as external output at the end of a GAMS run.

IDCGenerateJSON *(string)*: Specify JSON file name of data contract

Available: Command line

Specify the name of a JSON file that is used to store the information (but not the data) about all symbols that are declared as external input or external output at the end of a GAMS run.

IDCJSON *(string)*: Specify JSON file name to verify data contract

Available: Command line

Specify the name of a JSON file that is read and used to verify that the implicit data contract given by the GAMS model by declaring symbols as external input or external output matches the expected symbol information given by this specified file. This is the expected schema of that file:

```
{
  "$schema":"http://json-schema.org/draft-07/schema#",
  "title":"Input/Output configuration file schema",
  "description":"Configuration file is automatically generated by GAMS and describes the data contract.",
  "type":"object",
  "definitions": {
    "symbolHeader":{
      "description":"Symbol header",
      "type":"object",
      "properties":{
        "alias":{
          "description":"Explanatory text of the header",
          "type":"string",
          "minLength":1
        },
        "type":{
          "description":"GAMS input type",
          "type":"string",
          "enum":[
            "string",
            "numeric"
          ]
        }
      },
      "required": ["alias", "type"]
    },
    "symbol":{
      "description": "Symbol definition",
      "type": "object",
      "properties": {
        "alias":{
          "description": "Explanatory text of the symbol",
          "type":"string",
          "minLength":1
        },
        "symtype":{
          "description": "Symbol type",
          "type":"string",
          "default":"en",
          "enum":[
             "set",
             "parameter",
             "variable",
             "equation"
          ]
        },
        "headers":{
          "description":"Symbol headers",
          "type":"object",
          "minProperties":1,
          "additionalProperties":{
            "$ref": "#/definitions/symbolHeader"
          }
        }
      },
      "required": ["alias", "symtype", "headers"]
    },
    "scalarSymbol": {
      "description": "Scalar symbols that are grouped together in one table",
      "type": "object",
      "properties": {
        "alias":{
          "description": "Explanatory text of the symbol",
          "type":"string",
          "minLength":1
        },
        "symnames":{
          "description": "Symbol names in scalar tables",
```

```
          "type":"array",
          "minItems":1,
          "uniqueItems":true,
          "items":{
            "type":"string",
            "minLength":1
          }
        },
        "symtypes":{
          "description": "Symbol types in scalar tables",
          "type":"array",
          "minItems":1,
          "items":{
            "type":"string",
            "enum":[
               "set",
               "parameter",
               "variable",
               "equation"
            ],
            "minLength":1
          }
        },
        "symtext":{
          "description": "Symbol aliases (exp. text) in scalar tables",
          "type":"array",
          "minItems":1,
          "items":{
            "type":"string",
            "minLength":1
          }
        },
        "headers":{
          "description":"Symbol headers",
          "type":"object",
          "minProperties":1,
          "additionalProperties":{
            "$ref": "#/definitions/symbolHeader"
          }
        }
      },
      "required": ["alias", "symnames", "symtypes", "symtext", "headers"]
    }
  },
  "additionalProperties":false,
  "properties":{
    "modelTitle":{
      "description":"Title of the model",
      "type":"string",
      "default":"Unnamed model"
    },
    "inputSymbols":{
      "description":"Name of the symbols to receive data from MIRO. Identifiers are case insensitive (will
always be lower case).",
      "type":"object",
      "additionalProperties":{
        "oneOf": [
          { "$ref": "#/definitions/scalarSymbol" },
          { "$ref": "#/definitions/symbol"}
        ]
      }
    },
    "outputSymbols":{
      "description":"Name of the symbols to export to MIRO once computation has finished. Identifiers are
case insensitive (will always be lower case).",
      "type":"object",
      "additionalProperties":{
        "oneOf": [
          { "$ref": "#/definitions/scalarSymbol" },
          { "$ref": "#/definitions/symbol"}
        ]
      }
    }
  },
  "required":["modelTitle", "inputSymbols", "outputSymbols"]
}
```

IDCProtect *(boolean)*: Flag to control assignment protection of external input symbols

Available: Command line, Option statement

By default, it is not allowed to change symbols which are declared as external input at execution time. This parameter allows to ignore this restriction. It sets the initial state for the dollar control options $on/offIDCProtect.

Default: 1

Value	Meaning
0	Allow to change external input symbols at execution time
1	Protect external input symbols from being changed at execution time

IDE *(boolean)*: Integrated Development Environment flag

Available: Command line

This option instructs GAMS to write special instructions to the log file that are in turn read by an IDE.

Default: 0

Value	Meaning
0	Unknown environment
1	Runs under GAMS IDE

implicitAssign *(string)*: Switch default for "$on/offImplicitAssign"

Available: Command line

For more info see $on/offImplicitAssign.

Default: off

Value	Meaning
off	Activate $offImplicitAssign
on	Activate $onImplicitAssign

inlineCom *(string)*: Switch default for "$on/offInline" and "$inlineCom"

Available: Command line

If this is set to on `on` or `off` it sets the state of $on/offInline. Other strings (a pair with not more than two characters each) will set $inlineCom.

Default: off

Value	Meaning
off	Activate $offInline
on	Activate $onInline using default inline comment characters
other	Activate $onInline setting specific inline comment characters

input *(string)*: Input file

Synonym: I

Available: Command line

Completing the input file name with the current directory composes the final name. If such a file does not exist and the extension was not specified, the standard input extension will be attached and a second attempt will be made to open an input file.

inputDir, inputDir1..40 *(string)*: Input file directories (searched at compile time)

Synonym: IDIR

Available: Command line

In general, GAMS searches for input and include files in the current working directory only. This option allows the user to specify additional directories for GAMS to search for include and batinclude files as well as GDX file via $gdxIn. A maximum of 40 separate directories may be included. The directories are separated by Operating System specific symbols. For example, on a PC the separator is a semicolon (;) character and under Unix it is the colon (:) character. Note that libinclude and sysinclude files are handled differently. Their paths are specified with the command line parameters libIncDir and sysIncDir respectively.

Consider the following illustration:

```
> gams myfile idir \mydir;\mydir2
```

Note that the search order for the file `myfile.gms` and all included files in PC systems will be as follows: (1) the current directory, (2) the directories specified by `inputdir` in their respective order (here the directories: `\mydir` and `\mydir2`). Under Unix, the corresponding GAMS call will be:

```
> gams myfile idir \mydir:\mydir2
```

Note that the information in the option `inputDir` may be also transferred to GAMS by entering the individual directories separately. A maximum of 40 directories may be passed on in this manner. The number appended to `InputDir` is important since the earlier `InputDir` directories will be searched first.

The example above may alternatively be formulated in the following way:

```
> gams myfile idir1 mydir1 idir2 mydir2
```

Note that in this case the search order will be as follows:

1. `current directory`
2. `mydir1`
3. `mydir2`

Observe that we could modify the command in the following way:

```
> gams myfile idir3 \mydir1 idir2 \mydir2
```

Note that in this case the search order will be as follows:

1. `current directory`
2. `mydir2`
3. `mydir3`

Thus it is not the order in which the directories are specified that matters, but the number of the option `inputDir` that they have been assigned to.

integer1..5 *(integer)*: Integer communication cell N

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies an integer communication cell that may contain any integer number.

interactiveSolver *(boolean)*: Allow solver to interact via command line input

Available: Command line

Default: `0`

Value	Meaning
0	Interaction with solvelink 0 is not supported
1	Interaction with solvelink 0 is supported

intVarUp *(integer)*: Set mode for default upper bounds on integer variables

Available: Command line, Option statement

Default: 0

Value	Meaning
0	Set default upper bound for integer variables to +INF
1	Pass a value of 100 instead of +INF to the solver as upper bound for integer variables
2	Same as 0 but writes a message to the log if the level of an integer variable is greater than 100
3	Same as 2 but issues an execution error if the level of an integer variable is greater than 100

iterLim *(integer)*: Iteration limit of solver

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies the maximum number of permitted solver iterations, before the solver terminates the run. If this limit is reached, the solver will terminate and will return solver status `2 ITERATION INTERRUPT`. Note that the definition of what constitutes an interation depends on the solver. For LP solvers, `iterlim` often refers to the number of simplex iterations (i.e., pivots). For MIP solvers, `iterlim` often refers to the cumulative number of simplex iterations over all solves of LP relaxations. Observe that `iterlim` does not apply to all iterations. For example, it might not apply to barrier iterations and major iterations in nonlinear solvers. For these iterations solver-specific options need to be set. Moreover, some solver links might *reinterpret* the value of this option. For example, if left at default, the solver link might use the default iteration limit of the solver.

Default: 2147483647

iterUsd *(integer)*: Number of iterations used

Available: Attribute statement (use after solve)

This model attribute returns the number of iterations used after a solve.

jobTrace *(string)*: Job trace string to be written to the trace file at the end of a GAMS job

Synonym: JT

Available: Command line

This option specifies a string that is written to the trace file at the end of a GAMS job.

keep *(boolean)*: Controls keeping or deletion of process directory and scratch files

Available: Command line

This option controls whether to keep the process directory. In the process directory the temporary/scratch files are located, unless the options scrDir or procDir were used.

Default: 0

Value	Meaning
0	Delete process directory
1	Keep process directory

libIncDir *(string)*: LibInclude directory

Synonym: LDIR

Available: Command line

This option specifies the name of the directory to be used by GAMS for $libInclude files that do not have a full path specification. An absolute or relative path may be specified. If the option `libIncDir` is not set, it will be set to the subdirectory `inclib` in the GAMS standard locations. A relative path is relative to the GAMS system directory.

Note that if the option `libIncDir` parameter is set, the default library include directories will only be searched, if the libInclude file cannot be found in the specified folder.

Attention

Only *one* directory may be set with the option `libIncDir`. Thus the string specified will be treated as one directory. If additional directories are added, errors will be reported.

Consider the following example:

```
> gams myfile libIncDir mydir
```

Note that GAMS will search for any referenced libInclude file in the directory <`GAMS System Directory`>`/mydir` first.

license *(string)*: Use alternative license file

Available: Command line

This option specifies the name the file that contains the GAMS license. It may be used to point to an explicit license file rather letting GAMS search for the default `gamslice.txt` in various locations including the GAMS system directory. The locations ("Data directories") can be printed by running `gamsinst -listdirs`, see installation notes for details. The `license` option should only be used by advanced users attempting to override internal license information.

limCol *(integer)*: Maximum number of columns listed in one variable block

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of columns that are listed for each variable in the column listing in the listing file. Note that the value of zero will suppress the column listing.

Default: `3`

limRow *(integer)*: Maximum number of rows listed in one equation block

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of rows that are listed for each equation in the equation listing in the listing file. Note that the value of zero will suppress the equation listing.

Default: `3`

line *(integer)*: Line number of last solve of the corresponding model

Available: Attribute statement (use after solve)

This model attribute returns the line number of the last solve of the respective model.

linkUsed *(integer)*: Integer number that indicates the value of SolveLink used for the last solve

Available: Attribute statement (use after solve)

This model attribute returns an integer number that indicates the value of the option solveLink that was used for the last solve.

listing *(string)*: Switch default for "$on/offListing"

Available: Command line

For more info see $on/offListing.

Default: `on`

Value	Meaning
`off`	Activate $offListing
`on`	Activate $onListing

logFile *(string)*: Log file name

Synonym: LF

Available: Command line

This option is used in conjunction with the option logOption or short `LO`. If `lo` is set to 2 or 4, this option will specify the name of the log file name. The name provided by the option is completed using the current directory. If no logfile is given, but the value of `lo` is 2 or 4, then the file name will be the input file name with the extension `.log`.

Consider the following GAMS call:

```
> gams trnsport lo=2 lf=myfile.log
```

Note that the resulting log file will redirected to the file `myfile.log`. It will contain the following lines:

```
--- Starting compilation
--- trnsport.gms(75) 3 Mb
--- Starting execution: elapsed 0:00:00.017
--- trnsport.gms(44) 4 Mb
--- Generating LP model transport
--- trnsport.gms(63) 4 Mb
---   6 rows  7 columns  19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.072

IBM ILOG CPLEX   24.8.2 r59988 Released Jan  3, 2017 DEG x86 64bit/Mac OS X
Cplex 12.7.0.0

Reading data...
```

```
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.01 sec. (0.00 ticks)

Iteration      Dual Objective            In Variable           Out Variable
     1              73.125000    x(Seattle.New-York) demand(New-York) slack
     2             119.025000     x(Seattle.Chicago)  demand(Chicago) slack
     3             153.675000    x(San-Diego.Topeka)   demand(Topeka) slack
     4             153.675000  x(San-Diego.New-York)  supply(Seattle) slack
LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective :         153.675000

--- Restarting execution
--- trnsport.gms(63) 2 Mb
--- Reading solution for model transport
--- trnsport.gms(72) 3 Mb
 instructions that will go to the log file
 more instructions that will go to the log file
*** Status: Normal completion
--- Job trnsport.gms Stop 02/09/17 14:39:20 elapsed 0:00:00.280
```

logLine *(integer)*: Amount of line tracing to the log file

Synonym: LL

Available: Command line

This option may be used to limit the number of line tracing sent out to the log file during the compilation phase of a GAMS run. Note that setting this option to zero will cause the line tracing to be suppressed for all phases of the GAMS processing.

The log file that results from running the model [**TRNSPORT**] with the option `ll=0` is shown below.

```
 --- Starting compilation
--- Starting execution: elapsed 0:00:00.018
--- Generating LP model transport
---   6 rows  7 columns  19 non-zeroes
--- Executing CPLEX: elapsed 0:00:00.060

IBM ILOG CPLEX   24.8.2 r59988 Released Jan  3, 2017 DEG x86 64bit/Mac OS X
Cplex 12.7.0.0

Reading data...
Starting Cplex...
Space for names approximately 0.00 Mb
Use option 'names no' to turn use of names off
Tried aggregator 1 time.
LP Presolve eliminated 1 rows and 1 columns.
Reduced LP has 5 rows, 6 columns, and 12 nonzeros.
Presolve time = 0.01 sec. (0.00 ticks)
```

```
Iteration      Dual Objective            In Variable           Out Variable
     1              73.125000    x(Seattle.New-York) demand(New-York) slack
     2             119.025000     x(Seattle.Chicago)  demand(Chicago) slack
     3             153.675000    x(San-Diego.Topeka)   demand(Topeka) slack
     4             153.675000  x(San-Diego.New-York)  supply(Seattle) slack
LP status(1): optimal
Cplex Time: 0.03sec (det. 0.01 ticks)

Optimal solution found.
Objective :         153.675000

--- Restarting execution
--- Reading solution for model transport
 instructions that will go to the log file
 more instructions that will go to the log file
*** Status: Normal completion
--- Job trnsport.gms Stop 02/09/17 15:43:43 elapsed 0:00:00.275
```

If we compare this output to the output shown in the example of option logFile, we will observe that the line numbers are missing from this log file.

Default: 2

Value	Meaning
0	No line tracing
1	Minimum line tracing
2	Automatic and visually pleasing

logOption *(integer)*: Log option

Synonym: LO

Available: Command line

This option controls the location of the output log of a GAMS run. By default, GAMS directs the log of the run to the standard output. If `logOption` is set to 2, the log will be redirected to a file. Note that if no file name is provided for the log through the option logFile, the file name will be the input file name with the extension `.log`. Observe that the settings zero and 2 may be used to permit jobs to run in the background.

To illustrate, consider the following call:

```
> gams trnsport lo=2
```

Note that the resulting log file, `trnsport.log`, will be identical to the file `myfile.log` that is shown as part of the description of the option logFile.

Default: 3

Value	Meaning
0	No log output
2	Log output to logfile
3	Log output to standard output
4	Log output to logfile and standard output

LP *(string)*: Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Linear Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

lstTitleLeftAligned *(boolean)*: Write title of LST file all left aligned

Available: Command line

Default: `1`

Value	Meaning
`0`	Split LST title into left and right aligned part
`1`	Write LST title completely left aligned

marginals *(integer)*: Indicator for marginals present

Available: Attribute statement (use after solve)

maxExecError *(integer)*: Execution time error limit

Synonym: ExecErr

Available: Command line

This option puts an upper limit on the number of errors that may be found during execution or preprocessing associated with a solve statement. If more than `maxExecError` errors have been found GAMS will abort when hitting the next `solve` statement. If `maxExecError` is greater than the number of pending execution errors, the `solve` will be skipped, but the execution continues afterwards. This value can be overwritten with the function maxExecError.

Default: `0`

Value	Meaning
`0`	No errors allowed limit
`n`	Max number allowed

maxGenericFiles *(integer)*: Maximum number of generic file names tried at execution time file creation

Available: Command line, Option statement

In case of a failed writing of put files and GDX files via execute_unload (e.g. because the given path is invalid or the file is open by another program), GAMS tries to write to a generic file name instead. The number of file names tried can be specified with this option. To avoid this completely and throw an error right away, if the given name is invalid, set this to `0`.

Default: `20`

maxInfes *(real)*: Maximum of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the maximum number of infeasibilities after a solve.

maxProcDir *(integer)*: Maximum number of 225∗ process directories

Available: Command line

This option controls the maximum number of work file directories that may be generated by GAMS. By default they are called 225a, 225b, ..., 225aa, 225ab ... Note that the label `225` may be changed with the option procDir.

Default: `700`

MCP *(string)*: Mixed Complementarity Problems - default solver

Available: Command line, Option statement

The default solver for models of the type Mixed Complementarity Problems is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

MCPRHoldFx *(boolean)*: Print list of rows that are perpendicular to variables removed due to the holdfixed setting

Available: Command line, Option statement, Attribute statement (use before solve)

If option holdfixed is true, fixed columns in row.column matches are removed from MCP models but the matching rows remain. These rows are ignored by the solver. This option causes a list of such rows to be included in the listing file prior to the solve summary.

Default: `0`

meanInfes *(real)*: Mean of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the mean of the infeasibilities after a solve.

measure *(no value)*: Output of time and memory use since the last measure statement or the program beginning

Available: Option statement

This option has the effect that three measurements will be displayed: the memory and time usage since the last option statement `measure` and the total time used. See also the related option profile.

memoryStat *(no value)*: Show memory statistics in the LST file

Available: Option statement

This option has the effect that memory statistics will be shown in the listing file.

MIIMode *(string)*: Model Instance Mode

Available: Command line

This parameter is used in conjunction with the GAMS Model Instance Inspector (MII). The MII needs to have access to the scratch directory and requires a dictionary for the model instance to analyze. So, this parameter forces other parameters to be set:

- Keep is forced to `1`.
- DictFile is forced to `1`.
- SolveLink is forced to `2` with `MIIMode=singleMI` and to `4` with `MIIMode=multiMI`.

Default: `off`

Value	Meaning
`off`	Default behavior
`singleMI`	Setup to inspect a single model instance
`multiMI`	Setup to inspect multiple model instances from one model

MINLP *(string)*: Mixed-Integer Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Mixed Integer Nonlinear Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

MIP *(string)*: Mixed-Integer Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Mixed Integer Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

MIQCP *(string)*: Mixed Integer Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type Mixed Integer Quadratically Constrained Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

modelStat *(integer)*: Integer number that indicates the model status

Available: Attribute statement (use after solve)

This model attribute returns the model status after a solve. Observe that there are compile-time constants that are related to `modelStat`. Note that additional information to the values given in the table below is provided in section Model Status.

Value	Meaning
1	Optimal
2	Locally Optimal
3	Unbounded
4	Infeasible
5	Locally Infeasible
6	Intermediate Infeasible
7	Feasible Solution

Value	Meaning
8	Integer Solution
9	Intermediate Non-Integer
10	Integer Infeasible
11	Licensing Problem
12	Error Unknown
13	Error No Solution
14	No Solution Returned
15	Solved Unique
16	Solved
17	Solved Singular
18	Unbounded - No Solution
19	Infeasible - No Solution

MPEC *(string)*: Mathematical Programs with Equilibrium Constraints - default solver

Available: Command line, Option statement

The default solver for models of the type Mathematical Program with Equilibrium Constraints is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

multi *(string)*: Switch default for "$on/offMulti[R]"

Available: Command line

If this is set to on `on` or `off` it sets the state of $on/offMulti. The value `onR` activates $onMultiR.

Default: `off`

Value	Meaning
`off`	Activate $offMulti
`on`	Activate $onMulti
`onR`	Activate $onMultiR

multiPass *(boolean)*: Multipass facility

Synonym: MP

Available: Command line

This option may be used to instruct GAMS to use a quick syntax checking compilation facility which does not require all items to be declared. This alternative is useful when a large model is assembled from smaller pieces. It allows slices of GAMS code to be independently checked for syntax errors.

Consider the following example:

```
a(i) = b(i)*5 ;
b(i) = c(j) ;
```

By default, running a file containing just these two statements will generate the following listing file:

```
   1  a(i) = b(i)*5 ;
****  $140,120,140
   2  b(i) = c(j) ;
****         $140,120,149

120  Unknown identifier entered as set
140  Unknown symbol
149  Uncontrolled set entered as constant

**** 6 ERROR(S)   0 WARNING(S)
```

Note that both sets `i` and `j` have not been defined or initialized. In addition, the identifiers `a`, `b` and `c` have not been defined either. Further, an assignment cannot be made without the right-hand side of the assignment being known. However, in both assignments there is no data available for the right-hand side. If we run the same two lines with the option `mp=1`, we will get the following listing file:

```
   1  a(i) = b(i)*5 ;
   2  b(i) = c(j) ;
****            $149

149  Uncontrolled set entered as constant

**** 1 ERROR(S)   0 WARNING(S)
```

Observe that the statements have now been processed independently of their context. They are now checked only for consistency. GAMS now assumes that the sets `i` and `j`, as well as the identifiers `a`, `b`, and `c` have been defined and, if necessary, initialized elsewhere. The only error that is reported is the inconsistency of indices in the second statement.

Default: `0`

Value	Meaning
0	Standard compilation
1	Check-out compilation
2	As 1, and skip $call and ignore missing file errors with $include and $gdxin as well as unknown dimension errors with empty data statements

NLP *(string)*: Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Nonlinear Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

nodLim *(integer)*: Node limit in branch and bound tree

Available: Command line, Attribute statement (use before solve)

This option specifies the maximum number of nodes that are to be processed in the branch and bound tree search for a MIP problem. Note that setting `nodLim` can stop solutions that are exhibiting "excessive" iterations: if the limit is reached the algorithm will terminate, without obtaining optimality. In this case the solver status will be `4 TERMINATED BY SOLVER`. Observe that a value of zero is interpreted as 'not set'.

Default: `0`

nodUsd *(integer)*: Number of nodes used by the MIP solver

Available: Attribute statement (use after solve)

This model attribute returns the number of nodes used by the MIP solver after a solve.

noNewVarEqu *(integer)*: Triggers a compilation error when new equations or variable symbols are introduced

Available: Command line

Default: `0`

Value	Meaning
`0`	AllowNewVarEqu
`1`	DoNotAllowNewVarEqu

number *(real)*: Model instance serial number

Available: Attribute statement (use after solve)

This model attribute returns the model instance serial number. Note that the first model solved is assigned number 1, the second number 2 etc. The user may also set a value `n` and the next model solved will be assigned the number `n+1`.

numDepnd *(integer)*: Number of dependencies in a CNS model

Available: Attribute statement (use after solve)

This model attribute returns the number of dependencies identified in a CNS model after a solve.

numDVar *(integer)*: Number of discrete variables

Available: Attribute statement (use after solve)

This model attribute returns the number of discrete variables after a solve.

numEqu *(integer)*: Number of equations

Available: Attribute statement (use after solve)

This model attribute returns the number of equations after a solve.

numInfes *(integer)*: Number of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the number of infeasibilities after a solve.

numNLIns *(integer)*: Number of nonlinear instructions

Available: Attribute statement (use after solve)

This model attribute returns the number of nonlinear instructions after a solve.

numNLNZ *(integer)*: Number of nonlinear nonzeros

Available: Attribute statement (use after solve)

This model attribute returns the number of nonlinear nonzeros after a solve.

numNOpt *(integer)*: Number of nonoptimalities

Available: Attribute statement (use after solve)

This model attribute returns the number of nonoptimalities after a solve.

numNZ *(integer)*: Number of nonzero entries in the model coefficient matrix

Available: Attribute statement (use after solve)

This model attribute returns the number of nonzero entries in the model coefficient matrix after a solve.

numRedef *(integer)*: Number of MCP redefinitions

Available: Attribute statement (use after solve)

This model attribute returns the number of MCP equation-type redefinitions after a solve.

numVar *(integer)*: Number of variables

Available: Attribute statement (use after solve)

This model attribute returns the number of variables after a solve.

numVarProj *(integer)*: Number of bound projections during model generation

Available: Attribute statement (use after solve)

This model attribute returns the number of bound projections during model generation.

objEst *(real)*: Estimate of the best possible solution for a mixed-integer model

Available: Attribute statement (use after solve)

This model attribute returns the estimate of the best possible solution for a MIP or other models with discrete variables. The model attribute is mainly used after solve.

Some GAMS solvers implement algorithms (e.g. branch-and-bound) that generate a bound on the objective function value for the best possible solution. Users may access this bound by using the model attribute `objEst`. Note that this is mainly used for models with discrete variables (e.g. MIP and MINLP), but some global solvers implement spatial branch-and-bound algorithms that also provide such a bound for purely continuous problems. In case the solver does not set the attribute, its value is `na`.

objVal *(real)*: Objective function value

Available: Attribute statement (use after solve)

This model attribute returns the objective function value after a solve.

on115 *(boolean)*: Generate errors for unknown unique element in an equation

Available: Command line

This option generates errors for unknown unique elements in an equation.

Default: `0`

Value	Meaning
0	No error messages
1	Issue error messages

optCA *(real)*: Absolute Optimality criterion solver default

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies an *absolute termination tolerance* for a global solver. General problems are often extremely difficult to solve and proving that a solution that was found for a nonconvex problem is indeed the best possible solution can use enormous amounts of resources. The absolute gap is defined to be |PB-DB|. Here the primal bound PB is the objective function value of the best feasible solution found thus far and the dual bound DB is the current bound on the optimal value of the problem (i.e., lower bound in case of minimization and upper bound in case of maximization).

If the absolute gap is not greater than optCA, the solver will terminate and return solver status 1 NORMAL COMPLETION and model status 8 INTEGER SOLUTION (for a problem with discrete variables) or 2 LOCAL OPTIMAL or 7 FEASIBLE SOLUTION (for a problem without discrete variables). Note that this is a termination test only; setting this option should not change the global search.

Note

As this is an *absolute* criterion, setting the value to 100 means that the objective value will be within the 100 units of the true objective value.

Observe that a nonzero value for optCA will reduce solution time. However, it may cause the true integer optimum to be missed. This will be the case if at the time the solution algorithm stops, the value of the true integer optimum is within the tolerance specified by optCA of the best current solution. Therefore the reported solution could be the best, but it is guaranteed only to be within the tolerance of the true optimal solution.

Default: 0

optCR *(real)*: Relative Optimality criterion solver default

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies a *relative termination tolerance* for a global solver. General problems are often extremely difficult to solve and proving that a solution that was found for a nonconvex problem is indeed the best possible solution can use enormous amounts of resources. The precise definition of optCR depends on the solver. GAMS and some solvers use the following formula to compute the *optimality gap*:

```
|PB-DB| / max(|PB|,|DB|)
```

Here the primal bound PB is the objective function value of the best feasible solution found thus far and the dual bound DB is the current bound on the optimal value of the problem (i.e., lower bound in case of minimization and upper bound in case of maximization). However, two other formulas are also widely used, namely

```
|PB-DB| / |PB|
```

and

```
|PB-DB| / |DB|
```

Different adjustments when the denominator approaches zero or bounds are of different signs will be applied. The solver will stop as soon as it has found a feasible solution proven to be within optCR of optimal, that is, the optimality gap falls below optCR.

Note

As `optCR` is specified in *proportional* terms relative to the objective value, a value of 0.1 means that the objective value will be within 10% of the true objective value.

Observe that the solver will stop after finding a solution proven to be optimal within the tolerance specified with `optCR` and thus the solution time may be reduced. However, setting this option may cause the true integer optimum to be missed. This will be the case if at the time the solution algorithm stops, the value of the true integer optimum is within the tolerance specified by `optCR` of the best current solution. Therefore the reported solution could be the best, but it is guaranteed only to be within the tolerance of the true optimal solution.

Default: `1.0E-04`

optDir *(string)*: Option file directory

Available: Command line

This option may be used to specify the name of the directory for solver option files. By default, the directory will be set to the current working directory.

optFile *(integer)*: Default option file

Available: Command line, Attribute statement (use before solve)

This option instructs the solver to read an option file. The value of `optFile` determines which option file is used (see table below). Solver options allow users to manipulate the way solvers work. This may affect various solver functions including the choice of the branch and bound tree handling strategies. Please consult the solver manuals for options for each solver.

Note that this option is availabe as model attribute and command line parameter. Consider the following GAMS call:

```
> gams myfile optfile=1
```

Observe that the value of 1 for `optFile` means that the option file with the name `solverName.opt` will be used. Here `solverName` is the name of the respective solver. For example, if the solver CONOPT is used, the name of the respective option file is `conopt.opt`.

Note

If `optFile` is set with the model attribute in the GAMS input file, the value of the model attribute will override any `optFile` specifications on the command line.

Different values for `optFile` allow access to different option files for the same solver. Note that the following rule is used: if we specify `optfile = n`, then `solvername.opt` will be used for `n=1`, otherwise `solvername.opX`, `solvername.oXX` or `solvername.XXX` will be used, where `X`'s are the characters representing the value of `n`, for `n` $>$ `1`. Observe that no option file will be used if the value of `optFile` is zero.

Default: `0`

Value	Meaning
0	No option file will be used
1	The option file solvername.opt will be used
2	The option file solvername.op2 will be used
3	The option file solvername.op3 will be used
15	The option file solvername.o15 will be used
222	The option file solvername.222 will be used
1234	The option file solvername.1234 will be used

output *(string)*: Listing file name

Synonym: OO

Available: Command line

By default, the name of the output file (or listing file) is automatically created by combining the name of the input file with the current directory and applying the standard output file extension `.lst`. This option may be used to specify an alternative name for the output file. If the value is a file name without an absolute path, the current directory will compose the final name. If the absolute path is included in the file name, then the name is used as specified.

Consider the following examples:

```
gams trnsport
gams trnsport o=trnsportOut
gams trnsport o=c:\test\trnsport.out
```

Note that the first call will create an output file called `trnsport.lst` (for PC and Unix platforms) in the current directory. The second call will create a file called `trnsportOut` (without extension) in the current directory. The last call will create the file as listed. If the directory `c:\test` does not exist, GAMS will exit with a parameter error.

Creation of the output file can be suppressed by setting the command line parameter writeOutput to `0`.

pageContr *(integer)*: Output file page control option

Synonym: PC

Available: Command line

This option affects the page control in the listing file.

Default: `2`

Value	Meaning
0	No page control, with padding
1	FORTRAN style line printer format
2	No page control, no padding
3	Formfeed character for new page

pageSize *(integer)*: Output file page size (=0 no paging)

Synonym: PS

Available: Command line

This option specifies the number of lines that are used on a page for printing the listing file. The lower bound is zero, which is interpreted as +inf. That means that everything is printed to one page.

Default: `0`

pageWidth *(integer)*: Output file page width

Synonym: PW

Available: Command line

This option sets the print width on a page in the listing file with a possible range from 72 to 32767. If the value is outside the allowed range, the default value will be used.

Note that `.pw` is also a put file attribute. In the context of the put writing facility, it may be used to set the page width of a put file. See page width for further details.

Default: `32767`

parmFile *(string)*: Command Line Parameter include file

Synonym: PF

Available: Command line

This option specifies the name of a secondary customization parameter file to use. It is used to augment the command line adding more command line parameters from a file. It is read from the current directory unless a path is specified. For an example, see section Specifying Options Through a Secondary Parameter File.

pLicense *(string)*: Privacy license file name

Available: Command line

This option gives the name of a privacy license file that contains file encryption codes. A full path should be used. For more information, see Encrypting Files.

prefixLoadPath *(boolean)*: Prepend GAMS system directory to library load path

Available: Command line

The OS environment variable to locate shared libraries used to be prefixed with the GAMS system directory. The option controls if this done or not. For the Windows platform, setting this option has no impact.

Default: `0`

Value	Meaning
`0`	Do not set GAMS system directory at beginning of library load path
`1`	Set GAMS system directory at beginning of library load path

previousWork *(boolean)*: Indicator for writing workfile with previous workfile version

Available: Command line

When GAMS creates a workfile (e.g. using command line parameter restart) by default it uses the most recent version of the workfile format. GAMS is backward compatible with respect to workfile formats, i.e. newer GAMS versions can process workfiles generated by older GAMS version. As long as the workfile format has not changed older GAMS versions can even process workfiles generated by newer GAMS versions (using the forceWork=1 command line parameter). This particular command line parameter allows to create a workfile using the *previous* workfile format with the current GAMS version, so the previous GAMS version

(and probably a few more versions back) will properly restart from this workfile even without issueing a warning.

This option is mostly used in combination with submissions to the NEOS server. Setting this option to 1 allows to submit NEOS jobs with a restart file even if the NEOS version is somewhat older than the local GAMS version.

Default: `0`

Value	Meaning
`0`	Write workfile using the current version
`1`	Write workfile using the previous workfile version

priorOpt *(real)*: Priority option for variable attribute .prior

Available: Attribute statement (use before solve)

Instructs the solver to use the priority branching information passed by GAMS through variable suffix values variable.prior. If and how priorities are used is solver-dependent.

Default: `0`

procDir *(string)*: Process Directory

Available: Command line

This option specifies the name of the process directory. If specified, the directory must already exist and it will not be deleted when GAMS cleans up. By default, the process directory name is chosen automatically from the list 225a, 225b, ..., 225aa, 225ab ..., by skipping over existing entries, and the directory will be deleted during cleanup if the option keep is not used. Very little is written to the process directory, but the scratch directory is used more, and the option scrDir takes its default from the process directory.

procDirPath *(string)*: Directory to create process directory in

Available: Command line

This option specifies the directory where the process directory should be created. If specified, the directory must already exist. While the process directory does not get cleaned automatically if the option procDir is set, this is not the case if the option `procDirPath` is used instead. Thus it allows to conveniently change the location of the process directory without changing the GAMS cleanup behavior. Note that if the location of the process directory is changed, the location of the default scratch directory will be changed accordingly (see option scrDir).

procTreeMemMonitor *(boolean)*: Monitor the memory used by the GAMS process tree

Available: Command line

Setting this option to 1 will cause GAMS to record the high-memory mark for the GAMS process tree (i.e. the `gams` or `gams.exe` process and all its children) and note this information in the log just prior to finishing. This is done via a separate thread that runs periodically (use option procTreeMemTicks to control how often) to construct the GAMS process tree and compute the memory usage for each process in it. There is some overhead with this so it is suggested to use this option only when needed and with a procTreeMemTicks value not smaller than the default.

In addition to the memory high-water marks (measured using both the size of the resident set and the virtual set used by each process in the tree) the report indicates the number of times this measurement was taken and any failures during this computation. For details on how to interpret the resident set and virtual set sizes, see showOSMemory.

Default: `0`

Value	Meaning
0	Do not monitor memory usage for the GAMS process tree
1	Start a thread to monitor memory usage for the GAMS process tree

procTreeMemTicks *(integer)*: Set wait interval between memory monitor checks: ticks = milliseconds

Available: Command line

Sets the wait interval in milliseconds between checks of the GAMS process tree memory usage: see procTreeMemMonitor for details.

Default: 2000

procUsed *(integer)*: Integer number that indicates the used model type

Available: Attribute statement (use after solve)

Value	Meaning
1	LP
2	MIP
3	RMIP
4	NLP
5	MCP
6	MPEC
7	RMPEC
8	CNS
9	DNLP
10	RMINLP
11	MINLP
12	QCP
13	MIQCP
14	RMIQCP
15	EMP

profile *(integer)*: Execution profiling

Available: Command line, Option statement

The *execution profile* of a GAMS run contains the individual and cumulative time required to execute the sections of the GAMS model, as well as information on memory use. The option `profile` controls whether an execution profile will be generated in the listing file. Observe that `profile` is available as command line parameter and option statement.

Note

The value for `profile` that is specified with an option statement in the GAMS input file overrides the value of `profile` that is passed through the command line.

Observe that an execution profile will be generated if the option `profile` is assigned a value larger than zero (zero is the default). Setting `profile` to 1 has the effect that execution times for each statement and the number of set elements over which the particular statement is executed will be reported. However, statements in programming flow control structures like loops will be omitted. Information on the execution of these statements will be included in

the profile if the value is n, with n > 1. Note that an overview of the values for profile is given in the table at the end of this description.

Consider the following GAMS call:

```
> gams trnsport profile=1
```

This call causes the following additional lines to appear in the listing file:

```
----      1 InitE                    0.000     0.000 SECS      3 MB
----      1 ExecInit                 0.000     0.000 SECS      3 MB
----     44 Assignment c             0.011     0.011 SECS      4 MB       6
----     63 Assignment transport     0.000     0.011 SECS      4 MB       3
----     65 Solve Init transport     0.000     0.012 SECS      4 MB
----     57 Equation   cost          0.001     0.013 SECS      4 MB       1
----     58 Equation   supply        0.000     0.013 SECS      4 MB       2
----     59 Equation   demand        0.000     0.013 SECS      4 MB       3
----     65 Solve Fini transport     0.009     0.022 SECS      4 MB      19
----     65 GAMS Fini               0.001     0.001 SECS      4 MB
----      1 InitE                   0.000     0.000 SECS      2 MB
----      1 ExecInit                0.000     0.000 SECS      2 MB
----     65 Solver     transport     0.000     0.000 SECS      2 MB
----     65 Solve Read transport     0.002     0.002 SECS      2 MB
----     67 Display                 0.000     0.002 SECS      3 MB
----     69 Display                 0.000     0.002 SECS      3 MB
----     69 GAMS Fini               0.001     0.001 SECS      3 MB
```

Observe that the first column provides the line number in the input file of the statement that is executed.

The second column reports the type of the respective statement. For an overview of all GAMS statements, see section Classification of GAMS Statements. In addition, ExecInit denotes the beginning of the execution phase of the GAMS input file and GAMS Fini denotes the end of this phase. Note that as soon as a solve statement is processed, GAMS will pass control to the solver system. Once the solver has completed its task, GAMS will restart. Thus we have two ExecInit/ GAMS Fini pairs in our example. Note that only equations are listed, and not variables. This reflects the fact that GAMS uses an equation based scheme to generate a model.

The third and fourth columns show the individual time needed to execute the statement and the cumulative time taken by the GAMS system so far.

The last column gives the number of assignments that were generated in the specified line.

In addition to the lines above, a *profile summary* is created at the end of the listing file. This summary contains (up to) ten of the slowest execution steps. The profile summary from trnsport.lst follows:

```
---- Profile Summary (17 records processed)
     0.011   0.004GB        44 Assignment c (6)
     0.009   0.004GB        65 Solve Fini transport (19)
     0.002   0.002GB        65 Solve Read transport
     0.001   0.004GB        65 GAMS Fini
     0.001   0.004GB        57 Equation   cost (1)
     0.001   0.003GB        69 GAMS Fini
```

Note that execution profiles and profile summaries are particularly useful for detecting the sources of performance problems. For further details, see section Finding the Causes for Slow Program Execution.

Default: 0

Value	Meaning
0	No profiling
1	Minimum profiling
n	Profiling depth for nested control structures

profileFile *(string)*: Write profile information to this file

Synonym: PFILE

Available: Command line

This option causes profiling information to be written to a file. Note that profiling information is only created with the setting `profile=1` or `profile=2`. For example such a file may have the following conten:

```
  1        -1        0.000    0.003 ExecInit
 45         6        0.000    0.004 Assignment c
 66        -1        0.000    0.004 Solve Init transport
 58         1        0.000    0.004 Equation   cost
 60         2        0.000    0.004 Equation   supply
 62         3        0.000    0.004 Equation   demand
 66        19        0.015    0.004 Solve Fini transport
 66        -1        0.000    0.004 GAMS Fini
  1        -1        0.000    0.002 ExecInit
 66        -1        0.000    0.002 Solve Read transport
 68        -1        0.000    0.003 Display
 68        -1        0.000    0.003 GAMS Fini
```

profileTol *(real)*: Minimum time a statement must use to appear in profile generated output

Synonym: PTOL (only available on the command line)

Available: Command line, Option statement

This option sets the profile tolerance in seconds. All statements that take less time to execute than this tolerance are not reported in the listing file. Note that this option is only effective if the value of the option profile is larger than zero.

Default: `0`

putDir *(string)*: Put file directory

Synonym: PDir

Available: Command line

By default, put files are generated and saved in the current working directory. This option may be used to specify an alternative directory. Note that this option does not work if an absolute file name is provided through the file statement.

putND *(integer)*: Number of decimals for put files

Available: Command line

This sets the default for the put file attribute .nd.

Default: `2`

putNR *(integer)*: Numeric round format for put files

Available: Command line

This sets the default for the put file attribute .nr.

Default: `1`

Value	Meaning
0	Item is displayed in F or E format
1	Item is rounded to fit given width and decimals
2	Item is displayed in scientific notation
3	Item is rounded to fit given width
4	Item is displayed in F or E format ignoring given decimals

putPS *(integer)*: Page size for put files

Available: Command line

This sets the default for the put file attribute .ps.

Default: `58`

putPW *(integer)*: Page width for put files

Available: Command line

This sets the default for the put file attribute .pw.

Default: `32767`

QCP *(string)*: Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type Quadratically Constrained Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

real1..5 *(real)*: Real communication cell N

Available: Option statement, Attribute statement (use before solve)

This option specifies a real communication cell that may contain any real number.

reference *(string)*: Symbol reference file

Synonym: RF

Available: Command line

If this option is specified, all symbol references will be written to the specified file. Setting `rf` or `Reference` to the string 'default' will cause GAMS to create a reference file with the file root name of the GAMS input file and the extension `ref`. Thus the call

```
> gams trnsport rf=default
```

will generate the reference file `trnsport.ref`.

The reference file consists of three sections: The information about the code compiled, the information about the symbols in the GAMS program (also known as `symbol table`) and a list of all files included.

The information about the code compiled consists of records with 11 fields. The meaning of the fields is:

- record count
- symbol index
- symbol name
- symbol type
- reference type (available: declared, defined, impl-asn, assigned, ref, control)
- global listing line
- local line
- local column position (due to macros this many not be always 100% correct)
- include nesting level
- index in the file summary
- file name

The symbol table starts with the line that is indicated by a `0` in the record count/first field, followed by the number of records in this symbol table and the text "size of symboltable". The records for the symbol table consists of:

- internal symbol index
- symbol name
- symbol type (numerical)
- symbol type (string)
- dimension (0 for functions and models)
- cardinality (0 for functions, card(sym) for regular symbols, and number of equation symbols for models)
- list of length dimension with the internal symbol index of the domain set (0 if universe)
- symbol text

The list of files included starts with the line that is indicated by a `0` in the record count/first field, followed by the number of records in this file list and the text "input files". The records for the following list has the same information as the Include File Summary.

With the reference file viewer, GAMS Studio offers a convenient tool for inspecting the reference file.

referenceLineNo *(string)*: Controls the line numbers written to a reference file

Synonym: RFLN

Available: Command line

Default: `actual`

Value	Meaning
actual	Actual line number of symbol reference
start	Line number where the statement with the reference starts

reform *(integer)*: Reformulation level

Available: Option statement, Attribute statement (use before solve)

This option triggers an objective function reformulation. The interpretation depends on the solver. The solvers MINOS and SNOPT support this option. Note that the default value is zero and the range is [-2147483647,2147483647].

replace *(string)*: Switch between merge and replace when reading from GDX into non-empty symbol

Available: Command line, Option statement

The command line parameter Replace initializes the option Replace to control the behvior of gdxLoad.

Default: on

Value	Meaning
off	Merge into existing data when loading
on	Replace existing data when loading

resCalc *(real)*: Time spent in function and derivative calculations (deprecated)

Available: Attribute statement (use after solve)

resDeriv *(real)*: Time spent in derivative calculations (deprecated)

Available: Attribute statement (use after solve)

resGen *(real)*: Time GAMS took to generate the model in wall-clock seconds

Available: Attribute statement (use after solve)

This model attribute returns the time GAMS took to generate the model in wall-clock seconds.

resIn *(real)*: Time to import model (deprecated)

Available: Attribute statement (use after solve)

resLim *(real)*: Wall-clock time limit for solver

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies the time in seconds that the solver can run before it can terminate and return the solver status 3 RESOURCE INTERRUPT. The solver should start the clock soon after it starts, so the time required to read in the problem and do any reformulations, preprocessing or presolving is included in the time limit. Where possible, the time limit applies to the wall-clock time: this behavior translates well to multi-threaded solves. Moreover, some solver links might *reinterpret* the value of this option. For example, if left at default (1e+10), the solver link might use the default time limit of the solver.

Note

This value could be automatically reduced, if required because of etLim.

Default: `10000000000`

resOut *(real)*: Time to export solution (deprecated)

Available: Attribute statement (use after solve)

restart *(string)*: Name of a restart file, see The Save and Restart Feature

Synonym: R

Available: Command line

This option specifies the name of a work file that was written with the option save that will be used to restart the GAMS program. The work file is also called *restart file*. For more information including examples, see chapter The Save and Restart Feature.

restartNamed *(string)*: Name of another matching restart file, see Obfuscated Work Files

Synonym: RN

Available: Command line

resUsd *(real)*: Time the solver used to solve the model in seconds

Available: Attribute statement (use after solve)

This model attribute returns the time in seconds used by the solver. Wherever possible, the units used (wall-clock time vs. CPU time) will be the same as used by the reslim option.

RMINLP *(string)*: Relaxed Mixed-Integer Non-Linear Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Relaxed Mixed Integer Nonlinear Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

RMIP *(string)*: Relaxed Mixed-Integer Programming - default solver

Available: Command line, Option statement

The default solver for models of the type Relaxed Mixed Integer Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

RMIQCP *(string)*: Relaxed Mixed Integer Quadratically Constrained Programs - default solver

Available: Command line, Option statement

The default solver for models of the type Relaxed Mixed Integer Quadratically Constrained Programs is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

RMPEC *(string)*: Relaxed Mathematical Programs with Equilibrium Constraints - default solver

Available: Command line, Option statement

The default solver for models of the type Relaxed Mathematical Program with Equilibrium Constraints is set during installation. The user may change this default by setting this option to the desired solver.

Observe that if the solver was changed using an option statement, the default solver may be reset later in the program with another option statement, where the value of the option is set to `default`.

rngBndMax *(real)*: Maximum absolute non-zero value of bounds passed to the solver (excluding infinity)

Available: Attribute statement (use after solve)

rngBndMin *(real)*: Minimum absolute non-zero value of bounds passed to the solver

Available: Attribute statement (use after solve)

rngMatMax *(real)*: Maximum absolute non-zero value of coefficients in the model matrix passed to the solver (excluding infinity)

Available: Attribute statement (use after solve)

Note

For non-linear models, the value of the coefficients looked at depends on the activity levels of one or more of the variables. More details are described here.

rngMatMin *(real)*: Minimum absolute non-zero value of coefficients in the model matrix passed to the solver

Available: Attribute statement (use after solve)

Note

For non-linear models, the value of the coefficients looked at depends on the activity levels of one or more of the variables. More details are described here.

rngRhsMax *(real)*: Maximum absolute non-zero value of right hand sides passed to the solver (excluding infinity)

Available: Attribute statement (use after solve)

rngRhsMin *(real)*: Minimum absolute non-zero value of right hand sides passed to the solver

Available: Attribute statement (use after solve)

rObj *(real)*: Objective function value from the relaxed solve of a mixed-integer model when the integer solver did not finish

Available: Attribute statement (use after solve)

This model attribute returns the objective function value from the relaxed solve of a MIP when the integer solver did not finish.

save *(string)*: Creates a work file, see The Save and Restart Feature

Synonym: S

Available: Command line

This option specifies the name of a work file to be written. The work file is intended to be used later to restart the GAMS program and it is also frequently referred to as *save file* or *restart file*. If no explicit file extension is provided, the default file extension `.g00` is used. If no specific file path is provided the work file is created in the current directory.

Note

- The character "?" is not allowed in save file names - that use was intended for old (pre-GAMS 21.7) save files only.
- Save files are platform-independent.

For further information including examples, see chapter The Save and Restart Feature.

saveObfuscate *(string)*: Creates an obfuscated work file, see Obfuscated Work Files

Synonym: SO

Available: Command line

savePoint *(integer)*: Save solver point in GDX file

Synonym: SP (only available on the command line)

Available: Command line, Option statement, Attribute statement (use before solve)

This option instructs GAMS to save a point format GDX file that contains the information on the current solution point.

Default: `0`

Value	Meaning
`0`	No point GDX file is to be saved
`1`	A point GDX file from the last solve is to be saved
`2`	A point GDX file from every solve is to be saved
`3`	A point GDX file from the last solve is to be saved in the scratch directory
`4`	A point GDX file from every solve is to be saved in the scratch directory

scaleOpt *(boolean)*: Employ user specified variable and equation scaling factors

Available: Attribute statement (use before solve)

This option determines whether GAMS will employ user-specified variable and equation scaling factors. It must be set to a nonzero value if scaling factors are to be used. For more details on scaling, see section Model Scaling - The Scale Option.

Default: `0`

scrDir *(string)*: Scratch directory

Synonym: SD

Available: Command line

This option specifies the name of the scratch directory. The scratch directory is used by GAMS for intermediate files that are generated during execution. The scratch directory and all its contents are usually deleted at the end of the GAMS run. By default, the scratch directory takes its value from the process directory that is specified with the option procDir. If neither the scratch directory nor the process directory are specified, the scratch directory will be set to a subdirectory of the current working directory with an internally generated name. If the scratch directory is specified, the respective directory must already exist and neither the content nor the directory itself will be deleted by GAMS at the end of the run.

Note that the option solveLink may be used to reduce or eliminate the need for intermediate files.

scrExt *(string)*: Scratch file extension to be used with temporary files

Synonym: SE

Available: Command line

This option specifies the name of the extension for the temporary files that GAMS is generating during execution.

Default: `dat`

scriptExit *(string)*: Program or script to be executed at the end of a GAMS run

Available: Command line

By default, GAMS does not call an exit script anymore. If this is required, the option `scriptExit` has to be set explicitly to the script that should be called after GAMS terminates. Note that an empty template of an exit script is in the GAMS system directory: it is called `gmsxitnt.cmd` for Windows and `gmsxitus.run` for Unix.

scriptFrst *(string)*: First line to be written to GAMSNEXT file.

Synonym: SF

Available: Command line

This option specifies the first line written to `gamsnext`. The default is an empty string and the *first* line is not written.

scrNam *(string)*: Work file names stem

Synonym: SN

Available: Command line

This option specifies the name stem that is used to complete the names of intermediate work files. Note that the name stem must have at least one '?'. The name will be completed with the scratch directory and the standard scratch name extension.

seed *(integer)*: Random number seed

Available: Command line, Option statement

This option specifies the seed that is used for the pseudo random number generator.

Default: `3141`

showOSMemory *(integer)*: Show the memory usage reported by the Operating System instead of the internal counting

Available: Command line

GAMS keeps track of how much dynamic memory it has allocated and shows this in the log file at various points. With this option, memory usage statistics from the operating system can be shown instead. This is useful if the dynamic memory allocation is known or suspected to significantly underestimate the total memory usage of the GAMS process.

The resident set size (RSS) of a process is that part of the process address space actually residing in main memory (RAM). It excludes things like memory allocated but never used, memory that has been swapped out to disk, and parts of the executable that have never been loaded.

The virtual set size (VSS) is a measure of the entire address space used by the process, whether that space resides in physical memory or not. On Linux, it is essentially the RSS plus the items excluded from the RSS, i.e. the parts of the process address space that are not currently in physical memory. On macOS, the VSS seems to be a wild overestimate - we don't recommend using VSS on this platform. On Windows, we take VSS from the `PagefileUsage` reported by the OS. The `PagefileUsage` for each process does not include some major contributions to the process address space, such as mapped files. For this reason, the process RSS may be larger than its VSS on Windows, but VSS is still a good measure of dynamic memory allocation done by a process.

Default: `0`

Value	Meaning
0	Show memory reported by internal accounting
1	Show resident set size reported by operating system
2	Show virtual set size reported by operating system

solPrint *(integer or string)*: Solution report print option

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the printing of the solution listing to the listing file.

N.B: the command line accepts the string and numeric value, the option statement only the string value, and the attribute statement accepts the numeric value and compile-time constant.

The table below offers an overview of the different types of values and a description of the associated meaning.

String Value	Numeric Value	Compile-Time Constant	Meaning
off	0	solprint.Off	Remove solution listings following solves
on	1	solprint.On	Include solution listings following solves
silent	2	solprint.Silent	Suppress all solution information

Default: On

solSlack *(boolean)*: Causes the equation output in the listing file to contain slack variable values instead of level values

Available: Option statement

If the value of this option is set to 1, the equation output in the listing file will contain slack variable values instead of level values.

Default: 0

Value	Meaning
0	includes equation levels in the solution part of the LST file following solves
1	includes equation slacks in the solution part of the LST file following solves

solveLink *(integer)*: Solver link option

Synonym: SL (only available on the command line)

Available: Command line, Option statement, Attribute statement (use before solve)

This option specifies what solver linking conventions are used when GAMS executes a solve statement. Note that values 3 and 6 are relevant for grid computing and multi-threaded solves, respectively. Observe that there are compile-time constants that are associated with this option. When the model instance is saved to the scratch directory (values 0, 1, 2, and 3), the solution produced by the solve is also written to the scratch directory and read in by GAMS when the execution resumes.

Default: 2

Value	Meaning
0	Model instance and entire GAMS state saved to scratch directory, GAMS exits (and vacates memory), and the solver script is called. After the solver terminates, GAMS restarts from the saved state and continues to executing
1	Model instance saved to scratch directory, the solver is called from a shell while GAMS remains open
2	Model instance saved to scratch directory, the solver is called with a spawn (if possible) or a shell (if spawn is not possible) while GAMS remains open - If this is not supported by the selected solver, it gets reset to 1 automatically
3	Model instance saved to scratch directory, the solver starts the solution and GAMS continues
4	Model instance saved to scratch directory, the solver starts the solution and GAMS waits for the solver to come back but uses same submission process as 3 (test mode)

Value	Meaning
5	The model instance is passed to the solver in-memory - If this is not supported by the selected solver, it gets reset to 2 automatically
6	The model instance is passed to the solver in-memory, the solver starts the solution and GAMS continues
7	The model instance is passed to the solver in-memory, the solver starts the solution and GAMS waits for the solver to come back but uses same submission process as 6 (test mode)

solveOpt *(integer or string)*: Multiple solve management

Available: Command line, Option statement, Attribute statement (use before solve)

This option will instruct GAMS how to manage the model solution if only part of the variables and equations in the particular problem are solved.

Observe that this option is available as model attribute, option statement and command line parameter. The values for model attributes are numeric, while the values for option statements are text strings. The command line parameter accepts both, numeric and text values. In addition, there are compile-time constants that are associated with this option. The first of the two tables below offers an overview of how the different types of values are related and the second table gives the values in numerical terms and a description of the associated meaning.

Numeric Value	String Value	Compile-Time Constant
0	replace	solveOpt.Replace
1	merge	solveOpt.Merge
2	clear	solveOpt.Clear

Default: Merge

Value	Meaning
0	The solution information for all equations appearing in the model is completely replaced by the new model results; variables are only replaced if they appear in the final model
1	The solution information for all equations and variables is merged into the existing solution information
2	The solution information for all equations appearing in the model is completely replaced; in addition, variables appearing in the symbolic equations but removed by conditionals will be removed

solver *(string)*: Default solver for all model types that the solver is capable to process

Available: Command line, Option statement

The command line parameter solver=abc initializes the default solver for the model types the solver "abc" is capable of to abc. This initialization is done before the default solvers of individual model types are set via command line parameters. Consider the following example:

```
> gams trnsport lp=conopt solver=soplex
```

Note that this GAMS call will first set the solver SOPLEX as the solver for the model types LP and RMIP, since these are the model types that SOPLEX can handle. Then Conopt will

be reset as the default solver for LPs. Observe that the order of these parameters on the command line is irrelevant. If multiple occurrences of the option `solver` appear, the last entry will set the value of the option.

In addition, the solver for multiple model types may be set in the GAMS model source code via the following statement:

```
option solver = abc;
```

This statement sets the solver for the model types the solver abc can handle to `abc`. Note that in such an option statement the order of other solver setting options is significant. Consider the following example:

```
option lp=conopt, solver=soplex;
```

This statement will first set the solver for LPs to Conopt and in the next step to SOPLEX because SOPLEX is capable of handling the model type LP. In some cases it makes sense to set a solver twice. Consider the following example:

```
option solver=conopt, solver=cbc;
```

This option statement has the effect that models of the types CNS, DNLP, NLP, QCP, RMIQCP or RMINLP will be solved with Conopt and models of the types LP, RMIP or MIP will be solved with CBC.

Note that as usual, a specification of a solver through an option statement in the GAMS source takes precedence over a specification on the command line.

solverCntr *(string)*: Solver control file name

Synonym: SCNTR

Available: Command line

This option specifies the solver control file name. Note that the name is completed with the scratch directory and the scratch extension.

solverDict *(string)*: Solver dictionary file name

Synonym: SDICT

Available: Command line

This option specifies the solver dictionary file name. Note that the name completed with the scratch directory and the scratch extension.

solverInst *(string)*: Solver instruction file name

Synonym: SINST

Available: Command line

This option specifies the solver instruction file name. Note that the name is completed with the scratch directory and the scratch extension.

solverMatr *(string)*: Solver matrix file name

Synonym: SMATR

Available: Command line

This option specifies the solver matrix file name. Note that the name is completed with the scratch directory and the scratch extension.

solverSolu *(string)*: Solver solution file name

Synonym: SSOLU

Available: Command line

This option specifies the solver solution file name. Note that the name is completed with the scratch directory and the scratch extension.

solverStat *(string)*: Solver status file name

Synonym: SSTAT

Available: Command line

This option specifies the solver status file name. Note that the name is completed with the scratch directory and the scratch extension.

solveStat *(integer)*: Indicates the solver termination condition

Available: Attribute statement (use after solve)

This model attribute indicates the solver termination condition. Observe that there are compile-time constants that are related to `solveStat`. Note that additional information to the values given in the table below is provided in section Solver Status.

Value	Meaning
1	Normal Completion
2	Iteration Interrupt
3	Resource Interrupt
4	Terminated By Solver
5	Evaluation Interrupt
6	Capability Problems
7	Licensing Problems
8	User Interrupt
9	Setup Failure
10	Solver Failure
11	Internal Solver Failure
12	Solve Processing Skipped
13	System Failure

stepSum *(boolean)*: Summary of computing resources used by job steps

Available: Command line

This option controls the generation of a step summary of the processing times taken by GAMS during a given run.

For example, the call

```
> gams trnsport stepsum=1
```

will generate the following *step summaries* in the listing file:

```
STEP SUMMARY:     0.016       0.016 STARTUP
                  0.005       0.005 COMPILATION
                  0.065       0.065 EXECUTION
                  0.001       0.001 CLOSEDOWN
                  0.087       0.087 TOTAL SECONDS
                  0.089       0.089 ELAPSED SECONDS
                  3.942       3.942 MAX HEAP SIZE (MB)
```

Note that this step summary will be printed before the model is sent to the solver, thus it may be found before the solve summary. The second step summary will be printed after solution, it will appear at the very end of the listing file:

```
STEP SUMMARY:     0.004       0.020 STARTUP
                  0.000       0.005 COMPILATION
                  0.003       0.068 EXECUTION
                  0.000       0.001 CLOSEDOWN
                  0.007       0.094 TOTAL SECONDS
                  0.239       0.328 ELAPSED SECONDS
                  2.899       3.942 MAX HEAP SIZE (MB)
```

Observe that the first column reports the time for the individual section of the run, while the second column reports accumulated times including previous sections.

Default: `0`

Value	Meaning
0	No step summary
1	Step summary printed

strictSingleton *(boolean)*: Error if assignment to singleton set has multiple elements

Available: Command line, Option statement

This option affects the behavior of a membership assignment to a singleton set. If the value is set to zero, GAMS will not complain about a singleton set with more than one element, but will take only the first element. However, if the value is set to 1, a singleton set definition with more than one element will cause an error. Also, if this option is set as a command line parameter it initializes the state of the dollar control option $on/offStrictSingleton. So, it influences the singleton set checking at both compile- and execution time.

Default: `1`

Value	Meaning
0	Take first record if assignment to singleton set has multiple elements
1	Error if assignment to singleton set has multiple elements

stringChk *(integer)*: String substitution options

Available: Command line

This option affects the result of the check for %xxx% symbols. Note that %xxx% symbols may be environment variables (also at execution time) or compile-time variables.

Default: 0

Value	Meaning
0	No substitution if symbol undefined and no error
1	Error if symbol undefined
2	Remove entire symbol reference if undefined and no error

subSys *(string)*: Name of subsystem configuration file

Available: Command line

This option specifies the name of the configuration file that contains solver defaults and other information. This option should be used only by advanced users who attempt to override internal subsystem information. In case you want to add a solver to your GAMS system, please inspect the section Adding Solvers via gamsconfig.yaml.

subSystems *(no value)*: Lists all solvers available as well as the current default and active solvers in the LST file

Available: Option statement

This option has the effect that all available subsystems will be displayed in the listing file. Note that a solver is considered a subsystem.

suffixAlgebraVars *(string)*: Switch default for "$on/offSuffixAlgebraVars"

Available: Command line

For more info see $on/offSuffixAlgebraVars.

Default: on

Value	Meaning
off	Activate $offSuffixAlgebraVars
on	Activate $onSuffixAlgebraVars

suffixDLVars *(string)*: Switch default for "$on/offSuffixDLVars"

Available: Command line

For more info see $on/offSuffixDLVars.

Default: off

Value	Meaning
off	Activate $offSuffixDLVars
on	Activate $onSuffixDLVars

sumInfes *(real)*: Sum of infeasibilities

Available: Attribute statement (use after solve)

This model attribute returns the sum of infeasibilities after a solve.

suppress *(boolean)*: Compiler listing option

Available: Command line

If set to 1, this option will suppress the echoing of the contents of the input file(s) to the listing file. Note that this option is similar in functionality to the dollar control option $offlisting.

Note

The dollar control options $on/offlisting will affect the echo print in the listing file only if `suppress` is set to zero. If `suppress` is set to 1, the input file(s) will not be echoed to the listing file and the dollar control options will not have any effect on the listing file.

Default: `0`

Value	Meaning
0	Standard compiler listing
1	Suppress compiler listing

symbol *(string)*: Symbol table file

Available: Command line

This option specifies the name of a partial symbol table that may be written in conjunction with reference files.

symPrefix *(string)*: Prefix all symbols encountered during compilation by the specified string in work file

Available: Command line

sys10 *(boolean)*: Changes rpower to ipower when the exponent is constant and within 1e-12 of an integer

Available: Command line, Option statement

Default: `0`

Value	Meaning
0	Disable conversion
1	Enable conversion

sys11 *(integer)*: Dynamic resorting if indices in assignment/data statements are not in natural order

Available: Command line, Option statement

Speed-up for expressions containing constant indices or indices that are not in the natural order at the cost of increased memory use.

Default: `0`

Value	Meaning
0	Automatic optimization/restructuring of data
1	No optimization
2	Always optimize/restructure

sys12 *(integer)*: Pass model with generation errors to solver

Synonym: noSolveSkip (only available on the command line)

Available: Command line, Option statement

Default: 0

sys15 *(integer)*: Automatic switching of data structures used in search records

Available: Command line, Option statement

Default: 0

Value	Meaning
0	Automatic switching to dense data structures
1	No switching
2	Always switch
1x	Print additional information in lst file

sys16 *(integer)*: Disable search record memory (aka execute this as pre-GAMS 24.5)

Available: Command line, Option statement

Default: 0

sys17 *(integer)*: Disable sparsity trees growing with permutation (aka execute this as pre-GAMS 24.5)

Available: Command line, Option statement

Default: 0

sys18 *(integer)*: Use backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers

Available: Command line, Option statement

By default GAMS accepts floating-point numbers with arbitrarily many digits and converts them to correctly-rounded double-precision values. This option selects a backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers (see also the offDigit dollar control option).

Default: 0

Value	Meaning
0	Use modern scheme for reading floating-point numbers
1	Use backward compatible (i.e. pre-GAMS 31) scheme for reading floating-point numbers

sys19 *(integer)*: Disable permutation on Column Generation (aka execute this as pre-GAMS 36)

Available: Command line, Option statement

Default: `0`

sysDir *(string)*: GAMS system directory where GAMS executables reside

Available: Command line

This option sets the GAMS system directory. It is useful if there are multiple systems installed on the machine or when GAMS is called from an external system like Visual Basic.

sysIdent *(real)*: Solver identification number

Available: Attribute statement (use after solve)

sysIncDir *(string)*: SysInclude directory

Synonym: SDIR

Available: Command line

This option specifies the name of the directory to be used by GAMS for sysinclude files that do not have a full path specification. An absolute or relative path may be specified. If this option is not set, it will be set to the GAMS system directory.

Note that if this option is set, the default system include directory will not be searched.

Attention

Only *one* directory may be set with the option `sDir`. Thus the string specified will be treated as one directory. If additional directories are added, errors will be reported.

Consider the following example:

```
> gams myfile sdir mydir
```

Note that GAMS will search for any referenced sysinclude file in the directory `mydir`.

sysOut *(boolean or string)*: Solver Status file reporting option

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls whether additional solver generated output (the solver status file) is included in the listing file. Note that the contents of the solver status file are useful for debugging or to get additional information about a solver run. Normally, only those messages flagged by the solver as destined for the listing file will be listed. If the solver crashes or encounters any unexpected difficulties, the contents of the solver status file will be automatically sent to the listing file.

Note that the boolean values (`off`: 0 and `off`: 1) are deprecated, they are only relevant for backward compatibility. The table below gives the boolean values and the associated meaning.

String Value	Meaning
`off`	Suppress additional solver generated output
`on`	Include additional solver generated output

Default: `Off`

sysVer *(real)*: Solver version

Available: Attribute statement (use after solve)

tabIn *(integer)*: Tab spacing

Available: Command line

This option sets the tab spacing. The default value is 8, which means that the tabs are at columns 1, 9, 17, ... and the intermediate columns are replaced by blanks.

Default: `8`

Value	Meaning
`0`	Tabs are not allowed
`1`	Tabs are replaced by blanks
`n`	Tabs are 1, n+1, 2n+1,.. (default: n=8)

tFormat *(boolean)*: Time format

Synonym: TF

Available: Command line

This option controls the time format in the listing file. The three date formats correspond to the various conventions used around the world. For example, the time 7:45∼PM will be written as 19:45:00 with the default `tf` value of zero and as 19.45.00 with `tf=1`.

Default: `0`

Value	Meaning
`0`	Time as hh:mm:ss
`1`	Time as hh.mm.ss

threads *(integer)*: Number of processors to be used by a solver

Available: Command line, Option statement, Attribute statement (use before solve)

This option controls the number of processors to be used by a solver, often by using several threads for parallel computations. If the number is greater than the number of available processors, it will be reduced to the number of processors available. A value of zero means that the solver or solver-link will decide on the number of processors to use.

Default: `0`

Value	Meaning
`0`	Solver decides on number of processors to use
`n`	Use n processors
`minus_n`	Number of processors to leave free for other tasks

threadsAsync *(integer)*: Limit on number of threads to be used for asynchronous solves (solveLink=6)

Available: Command line, Option statement

Default: `-1`

Value	Meaning
`0`	Use number of available processors
`n`	Use n threads
`minus_n`	Number of processors to leave free for other tasks

timer *(integer)*: Instruction timer threshold in milli seconds

Available: Command line

This option specifies an instruction timer threshold in milli seconds. That means that only details about internal GAMS intructions that took more than `n` milli seconds are echoed to the log.

Default: `0`

Value	Meaning
`0`	Interpreted as +inf, no details echoed
`n`	Echo all details about internal GAMS instructions that took more than n milli seconds to the log

tolInfeas *(real)*: Infeasibility tolerance for an empty row of the form a.. 0∗x =e= 0.0001;

Available: Attribute statement (use before solve)

This option specifies the infeasibility tolerance for an empty row of the following form:

```
a.. 0*x =e= 0.0001;
```

If the option is not set, a tolerance of 10 times the machine precision will be used. Empty rows that fail this infeasibility check will be flagged with the listing file message `Equation infeasible due to rhs value`.

tolInfRep *(real)*: This attribute sets the tolerance for marking infeasible in the equation listing

Available: Attribute statement (use before solve)

This option sets the tolerance for marking an equation `infeasible` in the equation listing. Note that the default value is 1.0e-13.

Default: `1.0E-13`

tolProj *(real)*: Tolerance for setting solution values to a nearby bound when reading a solution

Available: Attribute statement (use before solve)

When a solution is returned from a solver, equation and variable level values that are very close to their lower or upper bounds are projected or moved to that bound, and a count of the projections/moves made is included in the solution report's `REPORT SUMMARY`. This option controls the relative tolerance used in testing for closeness: project if `abs(level-bound)` $<$ `tolproj`$*$`(1+abs(level))`. For MCP models, this projection is also applied to the variable marginal values. In this case the projection count is not incremented and the test is an absolute and not a relative test: project if `abs(marginal)` $<$ `tolproj`.

Default: `0`

trace *(string)*: Trace file name

Available: Command line

This option specifies the trace file name and causes a trace file to be written. Note that if a previous trace file of the same name already exists, then all new data output will be appended. Therefore, users should be careful to delete all old versions of the trace file if the a file name is reused and they do wish the new data to be appended.

traceLevel *(integer)*: Modelstat/Solvestat threshold used in conjunction with action=GT

Synonym: TL

Available: Command line

Default: `0`

traceOpt *(integer)*: Trace file format option

Available: Command line

This option specifies the format of the trace file. Note that several different types of trace files may be created, depending on what output information is desired.

Default: `0`

Value	Meaning
0	Solver and GAMS step trace
1	Solver and GAMS exit trace
2	Solver trace only
3	Solver trace only in format used for GAMS performance world
5	Gams exit trace with all available trace fields

tryInt *(real)*: Whether solver should make use of a partial integer-feasible solution

Available: Attribute statement (use before solve, reset by solve statement)

Signals the solver to make use of a partial or near-integer-feasible solution stored in current variable values to get a quick integer-feasible point. The exact form of implementation depends on the solver and may be partly controlled by solver settings or options. See the solver manuals for details.

tryLinear *(real)*: Examine empirical NLP model to see if there are any NLP terms active. If there are none the default LP solver will be used

Available: Attribute statement (use before solve)

If this option is set to 1, empirical NLP models will be examined to determine if there are any active NLP terms. If there are none, the default LP solver will be used. The procedure also checks to see if QCP and DNLP models can be reduced to an LP; MIQCP and MINLP can be solved as a MIP; RMIQCP and RMINLP can be solved as an RMIP. Note that the default value is zero.

user1..5 *(string)*: User string N

Synonym: U1

Available: Command line

This option permits users to enter a text for up to 5 user-defined options. The double dash parameters supersede these parameters.

warnings *(integer)*: Number of warnings permitted before a run terminates

Available: Command line

This option specifies the maximum number of allowable warnings, before the run terminates.

Default: ∞

workDir *(string)*: Working directory

Synonym: WDir

Available: Command line

This option sets the working directory. This option is useful when GAMS is called from an external system like Visual Basic. If it is not specified, the working directory will be set to the directory curDir.

workFactor *(real)*: Memory Estimate multiplier for some solvers

Available: Command line, Attribute statement (use before solve)

This option instructs the solver how much workspace to allocate for problem solution relative to the solver-computed estimate. For example, setting the value to 2 will double the memory estimate. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect. Note that in cases where both options `workfactor` and workSpace are specified, the value for `workSpace` will take precedence.

Default: `1`

workSpace *(real)*: Work space for some solvers in MB

Available: Command line, Attribute statement (use before solve)

This option instructs the solver how much workspace in Megabytes to allocate. If it is not speicified by the user, the solver will estimate the size. In cases where a solver allocates memory dynamically as it is needed, this option will have no effect, or it may be used as a memory limit.

writeOutput *(boolean)*: Switch to write output file

Available: Command line

This option controls whether to write the output file.

Default: `1`

Value	Meaning
`0`	Suppress output file creation
`1`	Write output file

xSave *(string)*: Creates a compressed work file

Synonym: XS

Available: Command line

This option specifies the name of a save file written in ASCII format in older GAMS systems (versions older than 21.7), in order for the save file to be platform independent and may be moved to machines with different operating systems.

In GAMS systems from release 22.3 and newer this option has the effect that compressed save files are written.

xSaveObfuscate *(string)*: Creates a compressed obfuscated work file

Synonym: XSO

Available: Command line

zeroRes *(real)*: The results of certain operations will be set to zero if abs(result) LE ZeroRes

Available: Command line

This option specifies the threshold value for internal rounding to zero in certain operations.

Default: `0`

zeroResRep *(boolean)*: Report underflow as a warning when abs(results) LE ZeroRes and result set to zero

Available: Command line

This option causes GAMS to issue warnings whenever a rounding occurs because of the setting of the option zeroRes.

Default: `0`

Value	Meaning
`0`	No warning when a rounding occurs because of ZeroRes
`1`	Issue warnings whenever a rounding occurs because of ZeroRes

zeroToEps *(string)*: Treat zero as eps

Available: Option statement

When this is set to `on`, zero values are interpreted as `EPS` when loading non-scalar parameters from GDX or embedded code at execution time. Also, when deriving a set from a parameter in a GDX file at execution time, set records will be created for parameter records with value 0, if `zeroToEps` is set to `on`, but not, if it is `off`.

See dollar control option $onEps for the compile-time equivalent.

Default: `off`

Value	Meaning
`off`	Treat Zero as Zero
`on`	Treat Zero as Eps

4.40.7 Executing an External Program

External programs may be run during a GAMS job either using the `$call`, `Execute` or `Put_utility` syntax. The $call procedure is executed at the moment that is encountered during compilation. The Execute and Put_utility commands causes the external program to be run during GAMS program execution. The contrast between these statements is important in two ways.

- Influence on results that can be included in a GAMS program – Anything run with $Call can generate files that can be included in the subsequent compilation. On the other hand files generated with Execute and Put_utility cannot be included because $Include operates only at compile time (unless you use Save and Restart).
- Influence on results that can be fed into the external program – Obviously when one is running an external program there is the desire to pass it data depicting results of the GAMS execution. $Call cannot do this as the data passed have to exist at compile time and cannot use the result of any GAMS calculations and solves in the current program. Execute commands on the other hand can use any data generated during a run which arise before the Execute and Put_utility command's position in the file through passage via put files or other mechanisms.

The big difference between the `$call` and `execute` is

- $call
  - can generate results to be immediately incorporated back into GAMS
  - cannot use GAMS results generated within this run because the $Call is executed at compile time.
- `execute` and `put_utility`
  - can cause a program to be started using results generated by the GAMS program (note such results do have to have been saved in an external file using a command like put)
  - cannot generate results which can be immediately reincluded into the GAMS program because new material can only be added compile time.

4.40.7.1 Execute

This command uses the syntax

```
execute[.async[NC]|.checkErrorLevel] "[=]command_to_execute"
```

to execute a program specified by command_to_execute. The execution occurs during the GAMS execution phase.

- The `=` will call the program directly, while without the `=` GAMS calls a shell that executes the program. When a program is executed in a shell, mechanisms like redirection (>) and pipe (|) will work.
- The `.async` suffix makes GAMS go ahead without waiting.
- The `.asyncNC` option tells the operating system to start the run a new console rather than sharing the console of the parent process allowing use of multiple processors. This is available under Windows only.
- The `.checkErrorLevel` suffix checks the errorLevel implicitly, raises an execution error and aborts the execution, if that is not 0.
- Since this occurs during execution one cannot use the compile time $Include to incorporate the results of that external run into the GAMS code except through a **GAMS from GAMS** approach as discussed below or through save and restart use (see the Save Restart chapter).

4.40.7.2 Asynchronous Execution

The `.async` variant of `$call` and `execute` start a job without waiting for the result. One can continue in the GAMS program and collect the return code of the job later. There are three ways to start a job asynchronously:

- `$call.async` ... (compile phase)
- `execute.async` '...'; (execution phase)
- 'put_utility fx 'exec.async' / '...'; / put_utility fx 'shell.async' / '...';` (execution phase)

After each of those the function JobHandle can be used to get the Process ID (pid) of the last job started. With `jobStatus(pid)` one could check for the status of a job. Possible return values are:

- 0: error (input is not a valid PID or access is denied)
- 1: process is still running
- 2: process is finished with return code which could be accessed by errorLevel
- 3: process not running anymore or was never running, no return code available

With `jobTerminate(pid)` a interrupt signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

With `jobKill(pid)` a kill signal can be sent to a running job. If this was successful the return value is one, otherwise it is zero.

The model [**ASYNCEXEC**] from the GAMS Test Library demonstrates the use of this feature.

4.40.8 Executing a GAMS Tool

To call tools from the GAMS Tool Library, there are counterparts to the $call and Execute, namely $callTool and ExecuteTool. The behavior is identical to the methods explained in the introduction of Executing an External Program.

4.40.8.1 ExecuteTool

This command uses the syntax

```
executeTool[.checkErrorLevel] '[toolCategory.]toolName tool_arguments';
```

to execute a tool specified by `[toolCategory.]toolName` together with the arguments `tool_arguments`. The execution occurs during the GAMS execution phase.

- The `.checkErrorLevel` suffix checks the errorLevel implicitly, raises an execution error and aborts the execution, if that is not 0.
- Since this occurs during execution one cannot use the compile time $Include to incorporate the results of that external run into the GAMS code except through save and restart use (see the Save Restart chapter).

For examples please refer to the GAMS Tools Library chapter.

4.41 Dollar Control Options

4.41.1 Introduction

Dollar control options are used to indicate compiler directives and options. Dollar control options are not part of the GAMS language and must be entered on separate lines marked with the symbol $ in the first column. A dollar control option line may be placed anywhere within a GAMS program and it is processed during the compilation of the program. The symbol $ is followed by one or more options separated by spaces. Since the dollar control options are not part of the GAMS language, they do not appear on the compilation output in the listing file unless an error has been detected or the user has requested them to be shown (with the option $onDollar).Note that dollar control option lines are not case sensitive and a continued compilation uses the previous settings.

This chapter is organized as follows. First an overview of the dollar control options will be given in section List of Dollar Control Options, where the options will be presented in groups reflecting their major functional categories. Section Detailed Description of Dollar Control Options will contain a reference list of all dollar control options in alphabetical order with detailed description for each.

We will conclude this chapter with separate sections for two important topics: Conditional Compilation, Macros in GAMS, Compressing and Decompressing Files, and Encrypting Files.

4.41.1.1 Syntax

In general, the syntax in GAMS for dollar control statements is as follows:

```
$option_name argument_list {option_name argument_list}
```

The symbol $ in the first column indicates that this is a dollar control statement. It is followed by the name of the dollar control option `option_name` and the list of arguments `argument_list` of the option. Depending on the particular option, the number of arguments required can vary from 0 to many. More than one dollar control option may be activated in one statement. Note that in this case the symbol $ is not repeated. Observe that some dollar control options require that they be the first option on a line.

Note

- No blank space is permitted between the character $ and the first option that follows.
- The effect of the dollar control option is felt immediately after the option is processed.
- Dollar control options are not part of the GAMS language they instruct the compiler to perform some task. Therefore, dollar control options are not terminated with a semicolon as real GAMS language statements.

A simple example of a list of dollar control options is shown below:

```
$title Example to illustrate dollar control options
$onsymxref onsymlist
```

Note that there is no blank space between the character $ and the option that follows. The first dollar control option $title sets the title of the pages in the listing file to the text that follows the option name. In the second line of the example above, two options are set: $onSymXRef and $onSymList. These options turn on the echoing of the symbol cross reference table and symbol listing in the compilation output in the listing file.

Observe that it is also permitted to place a dollar control statement in a column other than column 1. However, in this case the statement must begin with the symbols $$, like in this example

```
    $$title Example showing that dollar control option can start in any column with an extra $ added
```

4.41.2 List of Dollar Control Options

The dollar control options are grouped into nine major functional categories affecting

- the input comment format
- the input data format
- the output format
- reference maps
- program control
- GDX operations
- compile-time variables and environment variables
- macro definitions
- compressing and encrypting source files

The following subsections briefly describe the options in each of the categories.

4.41.2.1 Dollar Control Options Affecting the Input Comment Format

Option	Description
comment	Set the comment character
eolCom	Set the end-of-line comment character(s)
inlineCom	Set the in-line comment characters
maxCol	Set the right-hand margin of the input file
minCol	Set left-hand margin of the input file
offEolCom	Turn end-of-line comments off
offInline	Turn in-line comments off
offMargin	Turn margin marking off
offNestCom	Turn nested comments off
offText	Turn text mode off
onEolCom	Turn end-of-line comments on
onInline	Turn in-line comments on
onMargin	Turn margin marking on
onNestCom	Turn nested comments on
onText	Turn text on: the following lines are comment

Note that comments in GAMS are introduced in section Comments.

4.41.2.2 Dollar Control Options Affecting the Input Data Format

Option	Description
dollar	Set the 'dollar' character
offDelim	Turn delimited data statement syntax off
offDigit	Turn the number precision check off

Option	Description
offEmbedded	No embedded text or data allowed
offEmpty	Disallow empty data initialization statements
offEnd	Disallow alternate program control syntax
offEps	Disallow interpretation of EPS as zero
offExternalInput	End of external input section
offGlobal	Disallow inheritance of parent file settings
offIDCProtect	Allow to change external input symbols at execution time
offImplicitAssign	Trigger compilation error 141 if symbol has not been assigned
offSuffixDLVars	Do not allow domain limited variables with suffixes in model
offSuffixAlgebraVars	Do not allow variables with suffixes in model algebra
offTroll	Do not recognize Troll periodicity in set definitions using sequences
offUNDF	Do not allow UNDF as input
offUni	Do not allow domain violations in assignments
offWarning	Do not convert domain errors into warnings
onDelim	Turn delimited data statement syntax on
onDigit	Turn number precision check on
onEmbedded	Allow embedded text or data in set and parameter statements
onEmpty	Allow empty data initialization statements
onEnd	Allow alternate program control syntax
onEps	Interpret zero as EPS
onExternalInput	Start of external input section
onGlobal	Force inheritance of parent file settings
onIDCProtect	Protect external input symbols from being changed at execution time
onImplicitAssign	Suppress compilation errors 141 and implicitly assign symbol if symbol has not been assigned
onSuffixDLVars	Allow domain limited variables with suffixes in model
onSuffixAlgebraVars	Allow variables with suffixes in model algebra
onTroll	Recognize Troll periodicity in set definitions using sequences
onUNDF	Allow UNDF as input
onUni	Allow domain violations in assignments
onWarning	Convert certain domain errors into warnings
use205	Language syntax of release 2.05
use225	Language syntax of release 2.25 Version 1
use999	Latest language syntax
version	Test GAMS compiler version number

4.41.2.3 Dollar Control Options Affecting the Output Format

Option	Description
double	Double-spaced listing follows
echo	Echo text to a file
echoN	Echo text to a file without ending the line
eject	Advance to next page
hidden	Ignore text and do not list
lines	Next number of lines have to fit on the page
log	Send message to the log
offDollar	Turn the listing of dollar control option lines off

Option	Description
offEcho	End of block echo
offExternalOutput	End of external output section
offEpsToZero	Do not interpret `EPS` as zero when writing to GDX
offInclude	Turn the listing of include file names off
offListing	Turn echoing input lines to listing file off
offLog	Turn line logging off
offPut	End of block put
offUpper	Following print to listing file is mixed cased
offVerbatim	Stop verbatim copy
onDollar	Turn the listing of dollar control option lines on
onEcho	Start of block echo with substitution
onEchoS	Start of block echo with substitution
onEchoV	Start of block echo without substitution
onEpsToZero	Interpret `EPS` as zero when writing to GDX
onExternalOutput	Start of external output section
onInclude	Include file name echoed to listing file
onListing	Input lines echoed to listing file
onLog	Reset line logging
onPut	Start of block `put` without substitution
onPutS	Start of block `put` with substitution
onPutV	Start of block `put` without substitution
onUpper	Following print to listing file is all upper cased
onVerbatim	Start verbatim copy if dumpopt $\geq$ 10
remark	Comment line with suppressed line number
[x]save[.keepCode]	Create a (compressed) save file during compilation with and without execution code compiled so far
single	Single-spaced listing follows
stars	Set "****" characters in listing file
sTitle	Set subtitle and reset page
title	Set title and reset page

4.41.2.4 Dollar Control Options Affecting the Listing of Reference Maps

Option	Description
offSymList	Turn symbol list off
offSymXRef	Turn symbol cross reference listing off
offUElList	Turn unique element listing off
offUElXRef	Turn unique element cross reference off
onSymList	Turn symbol list on
onSymXRef	Turn symbol cross reference listing on
onUElList	Turn unique element listing on
onUElXRef	Turn unique element cross reference on

4.41.2.5 Dollar Control Options Affecting Program Control

Option	Description
abort[.noError]	Issue an (error) message and abort compilation
batInclude	Include file with substitution arguments
call	Execute another program during compilation
call.Async[NC]	Execute another program asynchronously during compilation
call.checkErrorLevel	Execute another program during compilation and raise error in case of problem
callTool	Execute a GAMS tool during compilation
callTool.checkErrorLevel	Execute a GAMS tool during compilation and raise error in case of problem
clear	Reset all data for an identifier to its default value
clearError[s]	Clear compilation errors
else	Else clause
elseIf	ElseIf structure with case sensitive comparison
elseIfE	ElseIf structure with expression evaluation
elseIfI	ElseIf structure with case insensitive comparison
endif	Close ifThen/ifThenE/ifThenI control structure
error	Issue an error message
exit	Exit from compilation
funcLibIn	Load extrinsic function library
goto	Go to line with given label name
hiddenCall	Execute another program (hidden) during compilation
hiddenCall.Async[NC]	Execute another program (hidden) asynchronously during compilation
hiddenCall.checkErrorLevel	Execute another program (hidden) during compilation and raise error in case of problem
hiddenCallTool	Execute a GAMS tool (hidden) during compilation
hiddenCallTool.checkErrorLevel	Execute a GAMS tool (hidden) during compilation and raise error in case of problem
if	Conditional processing, case sensitive
ifE	If statement with expression evaluation
ifI	Conditional processing, case insensitive
ifThen	IfThen-elseIf structure with case sensitive comparison
ifThenE	IfThen-elseIf structure with expression evaluation
ifThenI	IfThen-elseIf structure with case insensitive comparison
include	Include file
kill	Kill data connected with identifier
label	Label name as entry point from $goto .
libInclude	Include file from library directory
offCheckErrorLevel	Do not check errorLevel automatically after $[hidden]call
offECImplicitLoad	Do not allow implicit loading from embedded code
offEmbeddedCode	Ends embedded code section
offFiltered	Turn behavior of $[gdx]load* to domain checked read
offMulti	Turn redefinition of data off
offOrder	Allow `lag` and `lead` operations on dynamic or unordered sets
offRecurse	Disable recursive include files
offStrictSingleton	Take first label if data statement for singleton set has multiple elements
onCheckErrorLevel	Throw compilation error, if errorLevel is not 0 after $[hidden]call

Option	Description
onECImplicitLoad	Allow implicit loading from embedded code
onEmbeddedCode	Starts embedded code section with substitution
onEmbeddedCodeS	Starts embedded code section with substitution
onEmbeddedCodeV	Starts embedded code section without substitution
onFiltered	Turn behavior of $[gdx]load* to filtered read
onMulti	Turn redefinition of data on (merging into existing data)
onMultiR	Turn redefinition of data on (replacing existing data)
onOrder	`lag` and `lead` operations on constant and ordered sets only
onRecurse	Enable recursive include files
onStrictSingleton	Error if data statement for singleton set has multiple elements
maxGoTo	Maximum number of jumps to the same label
phantom	Define a phantom element
shift	bat/lib/sysInclude argument shift operation
stop	Stop compilation
sysInclude	Include file from system directory
terminate	Terminate compilation and execution
warning	Issue compilation warning

Note that conditional compilation in GAMS is discussed in section Conditional Compilation below.

4.41.2.6 Dollar Control Options for GDX Operations

Option	Description
declareAndLoad	Declare and load all symbols from specified GDX file
gdxIn	Open GDX file for input
gdxLoad	Load specified symbols from specified GDX file
gdxLoadAll	Load all symbols which are known to GAMS from specified GDX file
gdxOut	Open GDX file for output
gdxUnload	Unload symbols into specified GDX file.
load	Load symbols from previously opened GDX file
loadDC	Load symbols from previously opened GDX file - domain checked
loadDCM	Load symbols from previously opened GDX file - domain checked - merge
loadDCR	Load symbols from previously opened GDX file - domain checked - replace
loadFiltered	Load symbols from previously opened GDX file - domain filtered
loadFilteredM	Load symbols from previously opened GDX file - domain filtered - merge
loadFilteredR	Load symbols from previously opened GDX file - domain filtered - replace
loadIdx	Load symbols from previously opened GDX file which has been written using an indexed write
loadM	Load symbols from previously opened GDX file - merge
loadR	Load symbols from previously opened GDX file - replace
offExternalInput	End of external input section
offExternalOutput	End of external output section
offFiltered	Turn behavior of $[gdx]load* to domain checked read
onExternalInput	Start of external input section
onExternalOutput	Start of external output section
onFiltered	Turn behavior of $[gdx]load* to filtered read
unLoad	Unload symbols into previously opened GDX file.

Note that GDX facilities and utilities are introduced in chapter GAMS Data eXchange (GDX).

4.41.2.7 Dollar Control Options for Compile-Time Variables and Environment Variables

Option	Description
drop	Drop a scoped compile-time variable
dropGlobal	Drop a global compile-time variable
dropLocal	Drop a local compile-time variable
dropEnv	Drop an OS environment variable
escape	Define the % escape symbol
eval	Evaluate and define a scoped compile-time variable
eval.Set	Evaluate and define a scoped compile-time variable based on a GAMS set
evalGlobal	Evaluate and define a global compile-time variable
evalGlobal.Set	Evaluate and define a global compile-time variable based on a GAMS set
evalLocal	Evaluate and define a local compile-time variable
evalLocal.Set	Evaluate and define a local compile-time variable based on a GAMS set
prefixPath	Prefix the environment variable `PATH`.
scratchFileName	Set a name for a temporary file in the scratch directory using the scratch file extension
setArgs	Define local compile-time variables using argument list
setComps	Unpack dotted names into compile-time variables
setDDList	Check double dash GAMS parameters
setDDList.Cont	Start list of identifiers to check double dash GAMS parameters
setEnv	Define an OS environment variable
set	Define a scoped compile-time variable
setGlobal	Define a global compile-time variable
setLocal	Define a local compile-time variable
setNames	Unpack a filename into local compile-time variables
show	Show current GAMS compile-time variables, macros, and active files
showFiles	Show active input and include files
showMacros	Show current macros
showVariables	Show current GAMS compile-time variables
splitOption	Unpack a key/value pair into scoped environment variables

See also sections Compile-Time Variables and Environment Variables in GAMS.

4.41.2.8 Dollar Control Options for Macro Definitions

Option	Description
macro	Preprocess macro definition
offDotL	Do not assume `.l` for variables in assignments
offDotScale	Assume `.scale` for variables and equations in assignments
offExpand	Do not expand macros when processing macro arguments
offLocal	Limit `.local` nesting to one
offMacro	Do not recognize macros for expansion
onDotL	Assume `.l` for variables in assignments and put statements

Option	Description
onDotScale	Assume `.scale` for variables and equations in assignments and put statements
onExpand	Expand macros when processing macro arguments
onLocal	No limit on `.local` nesting
onMacro	Recognize macros for expansion

Note that macros are introduced in section Macros in GAMS below.

4.41.2.9 Dollar Control Options for Compressing and Encrypting Source Files

Option	Description
compress	Create compressed GAMS system file
decompress	Decompress a GAMS system file
encrypt	Create an encrypted GAMS system file
expose	Remove all access control restrictions
hide	Hide objects from the user
protect	Protect objects from being modifiied by the user
purge	Remove the objects and all associated data

4.41.3 Detailed Description of Dollar Control Options

In this section we will describe each dollar control option in detail. Note that the dollar control options are listed in alphabetical order for easy reference. Note further, that in each entry the default value, if applicable, is given in parentheses.

$abort[.noError]

Syntax:

```
$abort[.noError] [text]
```

If used as `$abort`, this option will issue a compilation error and abort the compilation. It may be followed by a text.

Example:

```
$if not %system.fileSys% == UNIX
$abort We only do UNIX
```

This stops compilation if the operating system is not Unix. Running the example above on Windows will result in the compilation being aborted and the following lines in the listing file:

```
    2  $abort We only do UNIX
****        $343

Error Messages
343  Abort triggered by above statement
```

This option has a variant: `$abort.noError`. If the extension `.noError` is used the compilation will be aborted as well, but there will be no error. If a save file is written, all remaining unexecuted code will be flushed. This allows effective reuse of the save file.

Note that there is also an abort statement in GAMS, it is used to terminate the execution of a program.

See also $exit, $error, $stop, and $terminate.

$batInclude

Syntax:

```
$batInclude external_file {arg}
```

The `$batInclude` facility performs the same task as the $include facility: it inserts the contents of the specified file `external_file` at the location of the call. However, in addition, the option `$batInclude` also passes on arguments `arg` which may be used inside the include file. `External_file` is the name of the batch include file, it may be quoted or unquoted. The arguments `arg` are passed on to the batch include file. These arguments are treated as character strings that are substituted by numbers inside the included file. The arguments may be single unbroken strings (quoted or unquoted) or quoted multi-part strings.

Note that the syntax has been modeled after the DOS batch facility. Inside the batch file, a parameter substitution is indicated by using the character `%` followed immediately by an integer value corresponding to the order of parameters on the list where `%1` refers to the first argument, `%2` to the second argument, and so on. If an integer value is specified that does not correspond to a passed parameter, then the parameter flag is substituted with a null string. The parameter flag `%0` is a special case that will substitute a fully expanded file name specification of the current batch included file. The flag `%$` is the current `$` symbol (see $dollar). Observe that parameters are substituted independent of context and the entire line is processed before it is passed to the compiler. There is one exception: parameter flags that appear in comments are not substituted.

Attention

- GAMS requires that processing the substitutions must result in a line of less than or equal to the maximum input line length.
- The case of the passed parameters is preserved, thus it may be used in string comparisons.

Example:

```
$batInclude "file1.inc" abcd "bbbb" "cccc dddd"
```

Note that `file1.inc` is included with `abcd` as the first parameter, `bbbb` as the second parameter and `cccc dddd` as the third parameter.

```
Parameter a,b,c ;
a = 1 ; b = 0 ; c = 2 ;
$batInclude inc2.inc b a
display b ;
$batInclude inc2.inc b c
display b ;
$batInclude inc2.inc b "a+5"
display b ;
```

The external file `inc2.inc` contains the following line:

```
%1 = sqr(%2) - %2 ;
```

The echo print in the corresponding listing file follows:

```
 1  Parameter a,b,c ;
 2  a = 1 ; b = 0 ; c = 2 ;
BATINCLUDE C:\tmp\inc2.inc
 4  b = sqr(a) - a ;
 5  display b ;
BATINCLUDE C:\tmp\inc2.inc
 7  b = sqr(c) - c ;
 8  display b ;
BATINCLUDE C:\tmp\inc2.inc
 10  b = sqr(a+5) - a+5 ;
 11  display b ;
```

Note that the option `$batInclude` appears three times with different arguments. GAMS is interprets the contents of the batch include file in turn as:

```
b = sqr(a) - a ;
b = sqr(c) - c ;
b = sqr(a+5) - a+5 ;
```

Note that the third call is not interpreted as `sqr(a+5)-(a+5)`, but instead as `sqr(a+5)-a+5`. The results of the display statement are shown at the end of the listing file are given below:

```
----      5 PARAMETER b                  =        0.000
----      8 PARAMETER b                  =        2.000
----     11 PARAMETER b                  =       40.000
```

Observe that the third call leads to `b = sqr(6)-1+5`, thus the final value of `b` is 40. Suppose the statement in the batch include file is modified to read as follows:

```
%1 = sqr(%2) - (%2) ;
```

With this modification the output generated by the display statement will be as follows:

```
----      5 PARAMETER b                  =        0.000
----      8 PARAMETER b                  =        2.000
----     11 PARAMETER b                  =       30.000
```

Note that the third call leads to `b = sqr(6)-6` which results in `b` taking a value of 30.

Note

The option `$batInclude` without any arguments is equivalent to the option $include.

See also $include, $libInclude, $sysInclude.

$call

Syntax:

```
$call [=]command
```

This option passes a `command` to the current operating system command processor and interrupts compilation until the command has been completed. If the command string is empty or omitted, a new interactive command processor will be loaded.

Example:

```
$call dir
```

This command creates a directory listing on a PC.

Note that the command string may be passed to the system and executed directly without using a command processor by prefixing the command with an '=' sign. Compilation errors will be issued if the command or the command processor cannot be loaded and executed properly.

```
$call gams trnsport
$call =gams trnsport
```

The first call will run the model [**TRNSPORT**] in a new command shell. The DOS command shell does not send any return codes from the run back to GAMS. Therefore any errors in the run are not reported back. The second call, however, will send the command directly to the system. The return codes from the system will be intercepted correctly and they will be available to the GAMS system through the errorLevel function.

Attention

- Some commands (like `copy` on a PC and `cd` in Unix) are shell commands and cannot be spawned off to the system. Using these in a system call will create a compilation error.

  ```
  $call 'copy myfile.txt mycopy.txt'
  $call '=copy myfile.txt mycopy.txt'
  ```

  The first call will work on a PC, but the second will not. The copy command may be used only from a command line shell. The system is not aware of this command (Try this command after clicking Run under the Start menu in Windows. You will find that it does not work).
- One has to take special care when the first argument of the `command` contains spaces (for example an executable with absolute path), e.g.:

  ```
  $call C:\path with spaces\gams.exe trnsport.gms
  ```

  This would fail, since the command processor would try to call `C:\path` as the executable. So, one might think that protecting the spaces using quotes will resolve this problem:

  ```
  $call "C:\path with spaces\gams.exe" trnsport.gms
  ```

  However, the quotes delimit the entire `$call command`, so that `trnsport.gms` gets interpreted as new dollar control option instead of an argument for the executable. The solution is to use nested quotes: One pair to delimit the `$call command` and another one to protect the spaces of the executable like this:

  ```
  $call '"C:\path with spaces\gams.exe" trnsport.gms'
  ```

  It gets even more complicated if the argument contains spaces as well, e.g. if we have `my model.gms` instead of `trnsport.gms`. Another pair of quotes solves this:

  ```
  $call '"C:\path with spaces\gams.exe" "my model.gms"'
  ```

See also $call.Async, $call.checkErrorLevel, $hiddenCall.

$call.Async[NC]

Syntax:

```
$call.Async[NC] command
```

`$call.Async` works like $call but allows asynchronous job handling. This means users may start a job `command` without having to wait for the result, they may continue in their model and collect the return code of the job later. The function jobHandle may be used to get the process ID (pid) of the last job started. The status of the job may be checked using the function jobStatus(pid). An interrupt signal to a running job may be sent with the function jobTerminate(pid). With the function jobKill(pid) a kill signal may be sent to a running job.

The difference between `$call.Async` and `$call.AsyncNC` is, that the latter starts processes in a *new console*, rather than sharing the console of the parent process.

Note

On non-Windows platforms `$call.AsyncNC` and `$call.Async` are synonyms.

$call.checkErrorLevel

Syntax:

```
$call.checkErrorLevel [=]command
```

`$call.checkErrorLevel` works like $call but checks the errorLevel implicitly, raises a compilation error and aborts compilation, if that is not 0.

Example:

```
$call.checkErrorLevel gams trnsport
```

This is doing the same as:

```
$call gams trnsport
$ifE errorLevel<>0 $abort 'Problem calling gams trnsport'
```

$callTool

Syntax:

```
$callTool command
```

This option calls a tool of the GAMS Tools Library at compile time.

Example:

```
$callTool [data.]ExcelDump excelFile [gdxOut=fileOut.gdx]
```

This tool writes all worksheets of an Excel workbook to GAMS symbols.

Note

- Similar to $call, `$callTool` returns a shell code that can be checked via `errorLevel`. In case one expects the tools to perform without error, it is recommended to add the suffix .checkErrorLevel. This will stop the entire execution of GAMS if an error occurs while executing the tool.
- `$callTool[.checkErrorLevel] [...]` is ignored while $onExternalInput is active and IDCGDXInput is set.

Attention

- $callTool is only available for calling tools from the GAMS Tools Library. For general commands to the operating system $call must be used.

See also $callTool.checkErrorLevel, $hiddenCallTool.

$callTool.checkErrorLevel

Syntax:

```
$callTool.checkErrorLevel command
```

$callTool.checkErrorLevel works like $callTool but checks the errorLevel implicitly, raises a compilation error and aborts compilation, if that is not 0.

Example:

```
$callTool.checkErrorLevel [data.]ExcelDump excelFile [gdxOut=fileOut.gdx]
```

This is doing the same as:

```
$callTool [data.]ExcelDump excelFile [gdxOut=fileOut.gdx]
$ifE errorLevel<>0 $abort 'Problem calling gams trnsport'
```

$clear

Syntax:

```
$clear  ident {ident}
```

This option resets all data for the identifiers ident to their default values. Note that only the following data types may be reset: sets, parameters, variables and equations. Note further, that the clearing is carried out during compile time and not when the GAMS program executes.

Example:

```
Set    i  / 1*20 /;
Scalar a  / 2 /;
$clear i a
display i, a;
```

The option $clear resets i and a to their default values: an empty set for i and zero for a. The output generated by the display statement follows:

```
----      4 SET i
                                                    ( EMPTY )

----      4 PARAMETER a                  =        0.000
```

Attention

The two-pass processing of a GAMS file may lead to seemingly unexpected results. Both the dollar control options and the data initialization is done in the first pass, and assignments in the second, irrespective of their relative locations. This is an issue particularly with $clear since data can be both initialized and assigned.

```
Scalar a / 12 /;
a = 5;
$clear a
display a;
```

Note that the scalar data initialization statement is processed during compilation and the assignment statement a = 5; during execution. In the order that it is processed, the example above is read by GAMS as:

```
* compilation step
Scalar a /12/ ;
$clear a
* execution step
a = 5;
display a ;
```

Therefore the result is that a takes the value of 5. The output from the display statement is as follows:

```
----      4 PARAMETER a                   =        5.000
```

Compare also $kill and the execution time option clear.

$clearError[s]

Syntax:

```
$clearError[s]
```

This option ($clearError and $clearErrors are synonyms) clears GAMS awareness of compiler errors and turn them into warning messages instead.

Example:

```
Scalar z / 11 /;
$eval x sqrt(-1)
$clearError
$log %x%
Display z;
```

Note that without the use of $clearError the program above would not continue with the execution after line 2.

$comment (∗)

Syntax:

```
$comment char
```

This option changes the symbol indicating a single line comment from the default ∗ to the single character char. Note that after this option is used, the new comment character char cannot be used in column 1 as before, since it got a special meaning. Note further, that the case of the character does not matter if it is used as a comment character. This option should be used with great care and we recommend to reset the symbol quickly to the default.

Attention

The case of the start-of-line comment character does not matter when being used.

Example:

```
$comment c
c now we use a FORTRAN style comment symbol
$comment *
* now we are back to the default
```

See also section Comments.

$compress

Syntax:

```
$compress source target
```

This option causes the file `source` to be compressed into the packed file `target`.

Example: Consider the following example where the well-known model [**TRNSPORT**] is used:

```
$call gamslib trnsport
$compress trnsport.gms t2.gms
$include t2.gms
```

The first command retrieves the file `trnsport.gms` and the second command compresses it. Note that a compressed GAMS file is treated like any other GAMS file, therefore it may be included and executed as usual. Large data files that do not change often can be compressed this way to save disk space.

The following example serves as a little utility to compress and decompress files:

```
$ifthen set decompress
$  if not set input  $set input file_c.gms
$  if not exist %input% $abort No file input file %input% exist
$  if not set output $set output file.gms
$  log Decompressing %input% into %output%
$  decompress %input% %output%
$else
$  if not set input  $set input file.gms
$  if not exist %input% $abort No file input file %input% exist
$  if not set output $set output file_c.gms
$  log Compressing %input% into %output%
$  compress %input% %output%
$endif
```

The program (saved to a file called `compress.gms`) can be used as follows:

```
> gams compress.gms --input myfile.gms --output myfile_c.gms
> gams compress.gms --decompress=1 --input myfile_c.gms --output myfile.gms
```

See also $decompress. Further details are given in chapter Compressing and Decompressing Files.

$declareAndLoad

Syntax:

```
$declareAndLoad GDXFileName
```

This option loads all symbols from a specified GDX file. It is similar to $gdxLoadAll, but does not only load the symbols declared so far but also declares symbols found in the GDX file on the fly if they were unknown. If a symbol already exists in GAMS the GDX declaration better matches it's previous GAMS declaration exactly (same rules as if we would write the second declaration in source, e.g. different symbol text is allowed or incremental declaration first `Variable x;` and in GDX `Positive Variable x;`) otherwise we trigger an error.

Example:

If there is a GDX file `trnsport` created from the [**TRNSPORT**] model, the following code will declare and load all symbols seen in the trnsport model before:

```
$declareAndLoad trnsport.gdx

option DmpUserSym;
```

This can be seen in the `.lst` file afterwards:

```
SYMBOL TABLE DUMP (USER SYMBOLS ONLY), NR ENTRIES = 12
ENTRY                                ID   TYPE DIM  LENGTH MEMORYEST DEFINED ASSIGNED DATAKNOWN
  138                                 i    SET   1       2      0 MB    TRUE    FALSE      TRUE
  139                                 j    SET   1       3      0 MB    TRUE    FALSE      TRUE
  140                                 a  PARAM   1       2      0 MB    TRUE    FALSE      TRUE
  141                                 b  PARAM   1       3      0 MB    TRUE    FALSE      TRUE
  142                                 d  PARAM   2       6      0 MB    TRUE    FALSE      TRUE
  143                                 f  PARAM   0       1      0 MB    TRUE    FALSE      TRUE
  144                                 c  PARAM   2       6      0 MB    TRUE    FALSE      TRUE
  145                                 x    VAR   2       6      0 MB    TRUE    FALSE      TRUE
  146                                 z    VAR   0       1      0 MB    TRUE    FALSE      TRUE
  147                              cost    EQU   0       1      0 MB    TRUE    FALSE      TRUE
  148                            supply    EQU   1       2      0 MB    TRUE    FALSE      TRUE
  149                            demand    EQU   1       3      0 MB    TRUE    FALSE      TRUE
END OF SYMBOL TABLE DUMP
```

See also $(on|off)Filtered, $(on|off)Multi, and $onMultiR.

$decompress

Syntax:

```
$decompress source target
```

This option causes the compressed file `source` to be decompressed into the unpacked file `target`.

Example: Consider the following example where the well-known model [**TRNSPORT**] is used:

```
$call gamslib trnsport
$compress trnsport.gms t2.gms
$decompress t2.gms t3.gms
$call diff t3.gms trnsport.gms
$if errorlevel 1 $abort t3.gms and trnsport.gms are not identical!
```

The first command retrieves the file `trnsport.gms`, the second command compresses it and the third command decompresses the compressed file. Note that the resulting file, `t3.gms`, is identical to the original file `trnsport.gms` which is tested via the `diff` program.

See also $compress. Further details are given in chapter Compressing and Decompressing Files.

$dollar ($)

Syntax:

```
$dollar char
```

This option changes the current 'dollar' symbol to the single character `char`.

Note

The special %$ substitution symbol can be used to get the current 'dollar' symbol.

Example:

```
$dollar #
#log now we can use '%$' as the '$' symbol
```

$double

Syntax:

```
$double
```

The lines following this option will be echoed double spaced to the echo print in the listing file.

Example:

```
Set    i / 1*2 / ;
Scalar a / 1   / ;
$double
Set    j / 10*15 / ;
Scalar b / 2     / ;
```

The resulting echo print in the listing file looks as follows:

```
1  Set i /1*2/ ;
2  Scalar a /1/ ;

4  Set j /10*15/ ;

5  Scalar b /2/ ;
```

Note that lines before the option $double are listed single spaced, while the lines after the option are listed with double space.

See also $single.

$drop

Syntax:

```
$drop VARNAME
```

This option destroys (removes from the program) the scoped compile-time variable VARNAME that was defined with the dollar control option $set.

Example:

```
$set NAME my name
$if set NAME $log Scoped compile-time variable NAME is set to "%NAME%"
$drop NAME
$if not set NAME $log Scoped compile-time variable NAME is not available anymore
```

See also $set, $dropGlobal, and $dropLocal.

$dropEnv

Syntax:

```
$dropEnv VARNAME
```

This dollar control option destroys (removes from the program) the operating system environment variable `VARNAME`. For detailed information, see the dollar control option .

Example:

```
$if setEnv GDXCOMPRESS $dropEnv GDXCOMPRESS
```

See also $setEnv, and $if setEnv.

$dropGlobal

Syntax:

```
$dropGlobal VARNAME
```

This option destroys (removes from the program) the global compile-time variable `VARNAME` that was defined with the dollar control option $setGlobal.

Example:

```
$setGlobal NAME my name
$if setGlobal NAME $log Global compile-time variable NAME is set to "%NAME%"
$dropGlobal NAME
$if not setGlobal NAME $log Global compile-time variable NAME is not available anymore
```

See also $setGlobal, and $drop.

$dropLocal

Syntax:

```
$dropLocal VARNAME
```

This option destroys (removes from the program) the local compile-time variable `VARNAME` that was defined with the dollar control option $setLocal.

```
$setLocal NAME my name
$if setLocal NAME $log Local compile-time variable NAME is set to "%NAME%"
$dropLocal NAME
$if not setLocal NAME $log Local compile-time variable NAME is not available anymore
```

See also $setLocal, and $drop.

$echo

Syntax:

```
$echo text >[>] external_file
```

This option allows to write the text `text` to a file `external_file`. The text and the file name may both be quoted or unquoted. The file name is expanded using the working directory. The option `$echo` tries to minimize file operations by keeping the file open in anticipation of another `$echo` to be appended to the same file. The file will be closed at the end of the compilation or when an option $call or any variant of the option $include is encountered. The redirection symbols > and >> have the usual meaning of starting at the beginning or appending to an existing file respectively.

Example:

```
$echo                                                    > echo.txt
$echo  The message written goes from the first non blank >> echo.txt
$echo 'to the first > or >> symbol unless the text is'   >> echo.txt
$echo "is quoted. The input File is %gams.input%. The"   >> echo.txt
$echo 'file name "echo.txt" will be completed with'      >> echo.txt
$echo  %gams.workdir%.                                   >> echo.txt
$echo                                                    >> echo.txt
```

The content of the resulting file `echo.txt` is the following:

```
The message written goes from the first non blank
to the first > or >> symbol unless the text is
is quoted. The input File is C:\tmp\echoTest.gms. The
file name "echo.txt" will be completed with
C:\tmp\.
```

See also $on/offEcho, and $echoN.

$echoN

Syntax:

```
$echoN text >[>] external_file
```

This option sends a text message `text` to an file `external_file` like $echo but writes no end of line marker so the line is repeatedly appended to by subsequent commands. The redirection symbols > and >> have the usual meaning of starting at the beginning or appending to an existing file respectively. Note that the text and the file name may be quoted or unquoted. By default the file will be saved in the working directory.

Example:

```
$echoN 'Text to be sent'           > 'aaa.txt'
$echoN 'More text'                >> aaa.txt
$echoN And more and more and more >> aaa.txt
$echo This was entered with $echo >> 'aaa.txt'
$echo This too                    >> aaa.txt
```

The created file `aaa.txt` contains the following text:

```
Text to be sentMore textAnd more and more and moreThis was entered with $echo
This too
```

See also $on/offEcho, and $echo.

$eject

Syntax:

```
$eject
```

This option advances the echo print to the next page.

Example:

```
$eject
Set       i,j ;
Parameter Data(i,j) ;
$eject
Scalar a;
a = 7;
```

The statements following the first `$eject` will be listed on one page in the echo print of the listing file and the statements following the second `$eject` will be listed on the next page.

$else

Syntax:

```
$ifThen[E|I] cond
  ...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option always appears together with the option $ifThen[E/I]. It is followed by an instruction which is executed if the conditional expression of the matching option $ifThen[E/I] is not true. For an example, see section Conditional Compilation with $ifThen and $else.

See also $ifThen, $elseIf and section Conditional Compilation.

$elseIf

Syntax:

```
$ifThen[E|I] cond
  ...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option always appears together with the option $ifThen[E/I]. It is followed by another condition and instruction. For an example, see section Conditional Compilation with $ifThen and $else.

See also $ifThen, $else, $elseIfE, $elseIfI and section Conditional Compilation.

$elseIfE

Syntax:

```
$ifThen[E|I] cond
  ...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option does the same as $elseIf but evaluates numerical values of the control variables.

See also $elseIf and section Conditional Compilation.

$elseIfI

Syntax:

```
$ifThen[E|I] cond
  ...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option does the same as $elseIf but it is case insensitive.

See also $elseIf and section Conditional Compilation.

$encrypt

Syntax:

```
$encrypt source target
```

This option causes a file to be converted into an encrypted file. Here `source` is the name of the source file to be encrypted and `target` is the name for the resulting encrypted file. Note that encryption requires the *secure* option to be licensed and is available for commercial licenses only. The command line parameter pLicense specifies the target license to be used for encryption. The encrypted file can only run on a system licensed with the license file used for encryption. No special action is required on the executing system since GAMS recognizes whether a file is encrypted and will process it accordingly. There is no option to *decrypt* an encrypted file, so better keep the original unencrypted file.

Further details and examples are given in chapter Encrypting Files.

$endIf

Syntax:

```
$ifThen[E|I] cond
  ...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This must option must be matched with one of the options $ifThen, $ifThenE or $ifThenI. For an example, see section Conditional Compilation with $ifThen and $else.

See also $ifThen and section Conditional Compilation.

$eolCom (!!)

Syntax:

```
$eolCom char[char]
```

This option redefines and activates the end-of-line comment symbol, which may be one character or a sequence of two characters. By default, this is initialized to `!!`, but is not active. The option $onEolCom is used to activate the end-of-line comments. If `$eolCom` is used, $onEolCom is set automatically.

Example:

```
$eolCom //
Set i /1*2/ ;     // set declaration
Parameter a(i) ;  // parameter declaration
```

Here the character sequence `//` serves as the end-of-line-comment indicator.

Attention

It is not allowed to reset the end-of-line comment symbol to the current end-of-line comment symbol. This would cause an compilation error as in the following example:

```
$eolCom //
$eolCom //
```

Some end of line character settings can cause confusion. The widely used end of line character sequence `//` is also legal GAMS syntax in `put` statement to indicate two line breaks:

```
file fx; put fx;
put 'first line' // 'second line' //;
$eolCom //
put 'third line' // 'fourth line';
```

results in a put file with the following content:

```
first line

second line

third line
```

This can also confuse syntax highlighting in editors (or on thsi web page). Other popular end of line characters like `#` and `@` are also used as GAMS syntax, see Controlling the Cursor On a Page.

The default for dealing with end-of-line comments can be set using the command line parameter eolCom.

See also section Comments for more about comments in GAMS.

$error

Syntax:

```
$error [text]
```

This option will issue a compilation error and will continue with the next line.

Example:

```
$if not exist myfile
$error File myfile not found - will continue anyway
```

Note that the first line checks if the file `myfile` exists. If the file does not exist, it will generate an error with the comment `File myfile not found - will continue anyway` and then the compilation will continue with the next line.

See also $abort, $exit, $terminate, and $stop.

$escape

Syntax:

```
$escape character
```

This option allows users to work with text sequences containing % without substitution.

This causes all subsequent commands of the form %symbol% to not have parameter substitution done for them. As a consequence, no parameter substitutions are performed in GAMS statements (mostly useful in display and put statements) and the outcome of such statements where %symbol% is used is just %symbol%.

Note that the effect of the option $escape may be reversed with the option $escape %.

Example:

```
$set tt DOIT

file it; put it;

display "first %tt%";
display "second %&tt%&";
put "display one ", "%system.date%" /;
put "display two " "%&system.date%&"/;

$escape &
display "third %tt%";
display "fourth %&tt%&";
put "display third ", "%system.date%" /;
put "display fourth " "%&system.date%&"/;

$escape %
display "fifth %tt%";
display "sixth %&tt%&";
put "display fifth ", "%system.date%" /;
put "display sixth " "%&system.date%&"/;
```

The output generated by the display statements follows:

```
----      6 first DOIT

----      7 second %&tt%&

----     12 third DOIT

----     13 fourth %tt%

----     18 fifth DOIT

----     19 sixth %&tt%&
```

The file it.put will contain the following lines:

```
display one 08/10/17
display two %&system.date%&
display third 08/10/17
display fourth %system.date%
display fifth 08/10/17
display sixth %&system.date%&
```

Note that this option was introduced to facilitate writing GAMS code (or command.com/cmd.exe batch scripts) from GAMS including unsubstituted compile-time variables. Text can also be written at compile-time without parameter substitution via option $on/offEchoV and at run-time via $on/offPutV.

Note

In GAMS the escape character *follows* the character (%) that needs to be escaped. In many other languages the escape character precedes the to be escaped character.

$eval

Syntax:

```
$eval VARNAME expression
```

This option evaluates a numerical expression at compile time and places it into a *scoped* compile-time variable. In turn the option $ifE may be used to do numeric testing on the value of this variable.

`VARNAME` is the name of a compile-time variable and `expression` is an expression that consists of constants, functions, operators and other compile-time variables with numerical values. Note that no whitespace is allowed in the expression which can be overcome by additional parentheses.

Example:

```
$eval b1 ifthen(uniform(0,1)<0.5,0,1)
$eval b2 ifthen(uniform(0,1)<0.5,0,1)
$eval b3 (%b1%)xor(%b2%)
$log b1=%b1% b2=%b2% b1 xor b2=%b3%
```

The first two lines use the `uniform` function to generate a random number between 0 and 1 and assign `0` if this number is less than 0.5 otherwise `1` via the `ifthen` function to the scoped compile-time variable `b1` and `b1`. In the third line we apply the logical `xor` operator to `b1` and `b2` and store the result in `b3`. The parentheses are required because the more natural expression `%b1% xor %b2%` contains spaces. In the forth line we print the values and result to the log.

```
b1=1 b2=1 b1 xor b2=0
```

The expression are evaluated using IEEE nonstop arithmetic, so no evaluation errors are triggered as demonstrated in the following example:

```
$eval OneDividedByZero 1/0
$log 1/0=%OneDividedByZero%
```

This produces the following log:

```
1/0=+INF
```

The `$eval` and related dollar control options give access to a reduced set of GAMS functions: abs, card, ceil, cos, errorlevel, exp, fact, floor, frac, gamsrelease, gamsversion, gday, gdow, ghour, gleap, gmillisec, gminute, gmonth, gsecond, gyear, ifthen, jdate, jnow, jobhandle, jobkill, jobstatus, jobterminate, jstart, jtime, log, log10, log2, max, min, mod, numcores, pi, power, round, sameas, sign, sin, sleep, sqr, sqrt, tan, trunc, and uniform. The available operators are: +, -, *, /, ** and even ^ (integer power) which is not available in regular GAMS expression and requires the use of the function `ipower`. The comparison relations are <, >, <=, >=, <>, and =. The logical operators are **not**, **and**, **or**, **xor**, **imp**, and **eqv**.

The expression also allows the use of dollar on the right. In the following example we replace the `ifthen` function by a dollar one the right:

```
$eval b1 1$(uniform(0,1)>=0.5)
$eval b2 1$(uniform(0,1)>=0.5)
$eval b3 (%b1%)xor(%b2%)
$log b1=%b1% b2=%b2% b1 xor b2=%b3%
```

Moreover, the `$eval` has access to data available at compile time. The expression can access the value of scalars and for other symbols we can use the `card` function to access the cardinality (at this point) of the symbol. Here is an example:

```
Scalar ac 'Avogadro constant' / 6.0221409e+23 /;
$eval log_ac round(log10(ac))
$log round(log10(ac))=%log_ac%
Set d / d0*d%log_ac% /;
$eval card_d card(d)
$log card(d)=%card_d%
```

Access to individual records of symbols is not possible. The embedded code facility allows access to symbol records at compile time.

The user has the possibility to hide some of the GAMS functions by symbols with the same name. In this case, the GAMS function can still be accessed, but needs to be prefixed with `system.` as in the following example:

```
Scalar pi /3/;

$eval myPi pi
$eval GAMSPi system.pi
$log myPi   = %myPi%
$log GAMSPi = %GAMSPi%
```

This is the resulting output in the log:

```
myPi   = 3
GAMSPi = 3.14159265358979
```

See also $eval.Set, $evalGLobal, $evalLocal, $ifE, and $set.

$eval.Set

Syntax:

```
$eval.Set VARNAME SETID[.[First|Last]TE|[First|Last]TL|[First|Last]TN|TS]
```

This option evaluates an attribute of a GAMS set at compile time and places it into a *scoped* compile-time variable. `VARNAME` is the name of a compile-time variable and `SETID` is the identifier of a GAMS set. The following attributes are allowed in this context:

Attribute	Description
`[First OR Last]TE`	Explanatory text of the first or last element of the set `SETID`
`[First OR Last]TL`	Label of the first or last element of the set `SETID`
`[First OR Last]TN`	Set identifier with with element labels of the first or last element of the set `SETID`
`TS`	Explanatory text of the set `SETID`. This is also the default, if no attribute is specified

Note

- `First??` and `??` are synonyms in the table above. It always accesses the first element of the referenced set.
- `Last??` always accesses the last element of the referenced set.
- For Singleton Sets `First??`, `Last??` and `??` are synonyms.

Example:

```
Singleton Set h Greeting / Hello          'Welcome' /;
Set           p Person   / Mr.President   'Male'
                           Mrs.Chancellor 'Female'  /;

$eval.Set X h.TE
$log %X%
$eval.Set X p.lastTL
$log %X%
$eval.Set X p.FirstTN
$log %X%
```

This will generate the following in the log:

```
Welcome
Mrs.Chancellor
p('Mr','President')
```

See also $eval, $evalGLobal.Set, $evalLocal.Set, and $set.

$evalGlobal

Syntax:

```
$evalGlobal VARNAME expression
```

This option evaluates a numerical expression at compile time and places it into a *global* compile-time variable. The syntax and behavior otherwise is identical to $eval.

$evalGlobal.Set

Syntax:

```
$evalGlobal.Set VARNAME SETID[.[First|Last]TE|[First|Last]TL|[First|Last]TN|TS]
```

This option evaluates an attribute of a GAMS set at compile time and places it into a *global* compile-time variable. The syntax and behavior otherwise is identical to $eval.Set.

$evalLocal

Syntax:

```
$evalLocal VARNAME expression
```

This option evaluates a numerical expression at compile time and places it into a *local* compile-time variable. The syntax and behavior otherwise is identical to $eval.

$evalLocal.Set

Syntax:

```
$evalLocal.Set VARNAME SETID[.[First|Last]TE|[First|Last]TL|[First|Last]TN|TS]
```

This option evaluates an attribute of a GAMS set at compile time and places it into a *local* compile-time variable. The syntax and behavior otherwise is identical to $eval.Set.

$exit

Syntax:

```
$exit
```

This option will cause the compiler to exit (stop reading) from the current file. This is equivalent to having reached the end of file.

Example:

```
Scalar a ;
a = 5 ;
display a ;
$exit
a = a+5 ;
display a ;
```

Note that the lines following the option `$exit` will not be compiled.

Observe that there is a difference to the dollar control option $stop. If there is only one input file, `$stop` and `$exit` will have the same effect. If the option `$exit` occurs *within* an include file, it acts like an end-of-file on the include file. However, if the option `$stop` occurs *within* an include file, GAMS will stop reading all input.

See also $abort, $error, $terminate, and $stop.

$expose

Syntax:

```
$expose all | ident1 ident2 ...
```

This option removes all privacy restrictions from identifiers.

With explicit identifiers the privacy restrictions are removed only for the listed identifiers and with `all` the restrictions are removed for *all* identifiers. The privacy restrictions may be set with the dollar control options $hide or $protect. Note that a special license file is needed for this feature to work and that the `expose` only takes effect in subsequent restart files. For further information, see chapter Secure Work Files.

$funcLibIn

Syntax:

```
$FuncLibIn InternalLibName ExternalLibName
```

This makes extrinsic function libraries available to a model. `InternalLibName` is the internal name of the library in the GAMS code and `ExternalLibName` is the name of the shared library in the file system. See Using Function Libraries for more information.

$gdxIn

Syntax:

```
$gdxIn [GDXFileName]
```

This option is used in a sequence to load specified items from a GDX file. Here `GDXFileName` denotes the name of the GDX file (with or without the extension `.gdx`) and the command opens the specified GDX file for reading. The use of `$gdxIn` without a file name closes the currently open GDX file. The command is used in conjunction with the option $load or one of its variants.

Example:

```
set i,j; parameters a(i), b(j), d(i,j), f;
$gdxIn mydata.gdx
$load i j a b d f
$gdxIn
```

See also $load, and $gdxOut.

$gdxLoad

Syntax:

```
$gdxLoad GDXFileName [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option loads specified symbols from a specified GDX file. It basically combines $gdxIn and $load in one call. This can be particularly useful for $[bat|lib]include files when you want to be sure, to not interfere with potentially previously opened GDX files. For example, a $load after a "$gdxLoad" will still load the symbols from the GDX file opened before:

```
$gdxIn A.gdx
$load a b c
$gdxLoad B.gdx d
* The following $load gets the symbols e, f, and g from GDX file A.gdx
$load e f g
```

See also $(on|off)Filtered, $(on|off)Multi, and $onMultiR.

$gdxLoadAll

Syntax:

```
$gdxLoadAll GDXFileName
```

This option loads all symbols known from a specified GDX file. It is similar to $gdxLoad, but does not need/allow a list of symbols to be loaded. Instead it compares the list of previous declared symbols with the symbols in the specied GDX file and loads all symbols that match. If we have incompatibilities between symbols in GDX and GAMS (the incompatibilities are the same as if we would do an explicit `$load sym`) CMEX triggers an error. Symbols in GDX that don't exist in GAMS are just ignored.

Example:

If there is a GDX file `trnsport` created from the **[TRNSPORT]** model, the following code will load the symbols `i`, `j` and `c`. All other symbols are ignored:

```
Set       i,j,k;
Parameter c(i,j);
$gdxLoadAll trnsport.gdx
```

See also $(on|off)Filtered, $(on|off)Multi, and $onMultiR.

$gdxOut

Syntax:

```
$gdxOut [GDXFileName]
```

This option is used in a sequence to unload specified items to a GDX file at compile time. Here `GDXFileName` denotes the name of the GDX file (with or without the extension GDX) and the command opens the specified GDX file for writing. The use of `$gdxOut` without a file name closes the currently open output GDX file and initiates the actual writing of the sequence of symbols. The command is used in conjunction with the dollar control option $unLoad.

Example:

```
set i /i1*i3/; parameters a(i) /i1 3, i2 87, i3 1/;
$gdxOut mydata.gdx
$unLoad i a
$gdxOut
```

See also $unLoad, and $gdxIn.

$gdxUnload

Syntax:

```
$gdxUnload GDXFileName [sym1[,] sym2=gdxSym2[,] ...]
```

This option unloads symbols from a specified GDX file. It basically combines $gdxOut and $unload in one call. This can be particularly useful for $[bat|lib]include files when you want to be sure, to not interfere with potentially previously opened GDX files. For example, a $unload after a "$gdxUnload" will still unload the symbols to the GDX file opened before:

```
$gdxOut A.gdx
$unload a b c
$gdxUnload B.gdx d
* The following $unload puts the symbols e, f, and g into GDX file A.gdx
$unload e f g
```

$goto

Syntax:

```
$goto id
$label id
```

This option will cause GAMS to search for a line starting with `$label id` and then continue reading from there. This option can be used to skip over or repeat sections of the input files. In $batinclude files the target labels or label arguments can be passed as parameters because of the manner in which parameter substitution occurs in such files. In order to avoid infinite loops, jumps to the same label are restricted to a maximum of 100 times by default. This maximum may be changed with the option $maxGoto.

Example:

```
Scalar a ;
a = 5;
display a ;
$goto next
a = a+5 ;
display a ;
$label next
a = a+10 ;
display a ;
```

Note that GAMS will continue from line `$label next` after reading line `$goto next`. Observe that all lines in between are ignored. Therefore the final value of `a` in the example above will be 15.

Attention

The lines `$goto` and $label have to be in the same file. If the target label is not found in the current file an error will be issued.

See also $label, $maxGoto.

$hidden

Syntax:

```
$hidden text
```

A line starting with this option will be ignored and will not be echoed to the listing file. This option is used to enter information only relevant to the person manipulating the file.

Example:

```
$hidden You need to edit the following lines if you want to:
$hidden
$hidden     1. Change form a to b
$hidden     2. Expand the set
```

The lines above serve as comments to the person who wrote the file. However, these comments will not be visible in the listing file and are therefore hidden from view.

Note

This option is particularly useful when the input file is encrypted.

$hiddenCall

Syntax:

```
$hiddenCall [=]command
```

This option does the same as $call but the statement is neither shown on the log nor the listing file.

$hiddenCall.Async[NC]

Syntax:

```
$hiddenCall.Async[NC] command
```

This option does the same as $call.Async[NC] but the statement is neither shown on the log nor the listing file.

$hiddenCall.checkErrorLevel

Syntax:

```
$hiddenCall.checkErrorLevel [=]command
```

This option does the same as $call.checkErrorLevel but the statement is neither shown on the log nor the listing file.

$hiddenCallTool

Syntax:

```
$hiddenCallTool command
```

This option does the same as $callTool but the statement is neither shown on the log nor the listing file.

$hiddenCallTool.checkErrorLevel

Syntax:

```
$hiddenCallTool.checkErrorLevel command
```

This option does the same as $callTool.checkErrorLevel but the statement is neither shown on the log nor the listing file.

$hide

Syntax:

```
$hide all | ident1 ident2 ...
```

This option hides identifiers so they cannot be displayed or computed, but they may still be used in model calculations (i.e. commands when the solve statement is executed).

With explicit identifiers the listed identifiers are hidden and with `all` all identifiers are hidden. These restrictions may be removed with the dollar control options expose or purge. Note that a special license file is needed for this feature to work.

For further information, see chapter Secure Work Files.

$if

Syntax:

```
$if [not] conditional_expression new_input_line
```

This dollar control option provides the greatest amount of control over conditional processing of the input file(s).

For more information on the `conditional expressions` allowed, details on the `new_input_line` and examples, see section Conditional Compilation below.

See also $ifE, $ifI, $ifThen.

$ifE

Syntax:

```
$ifE [not] conditional_expression new_input_line
```

This dollar control option does the same as the option $if but allows constant expression evaluation. The `conditional_expression` may take two different forms:

```
expr1 == expr2      TRUE   if (expr1-expr2)/(1+abs(expr2)) < 1e-12
expr                TRUE   if expr1 <> 0
```

Example:

```
Scalar a;
$ifE (log2(16)^2)=16           a=0; display a;
$ifE log2(16)^2 == 16          a=1; display a;
$ifE NOT round(log2(16)^2-16)  a=2; display a;
$ifE round(log2(16)^2-16)      a=3; display a;
$ifE round(log2(16)^2-17)      a=4; display a;
```

This will create the following output:

```
----      3 PARAMETER a                  =        1.000

----      4 PARAMETER a                  =        2.000

----      6 PARAMETER a                  =        4.000
```

See also $if and section Conditional Compilation.

$ifI

Syntax:

```
$ifI [not] conditional_expression new_input_line
```

This option is working like the option $if. The only difference is that $if makes comparisons involving text in a case sensitive fashion while `$ifI` is case insensitive.

See also $if and section Conditional Compilation.

$ifThen

Syntax:

```
$ifThen[E|I] cond
  ...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option is a form of the option $if that controls whether a number of statements are active. The syntax for the condition is generally the same as for the option $if. Like `$if`, it is case sensitive. Often it is followed by one or more of the following dollar control options: $else, $elseIf, $elseIfI, $elseIfE. The option `$ifThen` must be matched with the option $endIf that marks the end of the construct. An example is given in section Conditional Compilation with $ifThen and $else.

Note that users may add a tag to the `$ifThen` and $endIf. For example, `$ifThen.tagOne` has to match with `$endif.tagOne`.

Example:

```
$ifThen.one x == y
display "it1";
$elseIf.one a == a
display "it2";
$ifThen.two c == c
display "it3";
$endIf.two
$elseIf.one b == b
display "it4";
$endIf.one
```

The resulting listing file will contain the following lines:

```
----      2 it2

----      4 it3
```

Note that the first condition (`x == y`) is obviously not true and the fourth condition (`b == b`) is not tested because the second condition (`a == a`) was already true.

Also note that the false clause is not interpreted by the compiler. So the following code would compile without problems:

```
$ifThen x == y
$the
$compiler
doesn't
complain
$endIf
```

See also $if, $ifThenE, $ifThenI, $else, $elseIF and section Conditional Compilation.

$ifThenE

Syntax:

```
$ifThen[E|I] cond
  ...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option does the same as the option $ifThen but evaluates numerical values of the control variables.

See also $ifThen and section Conditional Compilation.

$ifThenI

Syntax:

```
$ifThen[E|I] cond
  ...
{ $elseIf[E|I] cond
  ... }
[ $else
  ... ]
$endIf
```

This option does the same as the option $ifThen but it is case insensitive.

See also $ifThen and section Conditional Compilation.

$include

Syntax:

```
$include external_file
```

This option inserts the contents of a specified text file at the location of the call. Note that text files that have been compressed with Posix utility `gzip` or have been encrypted with tool ENDECRYPT are also allowed. `External_file` is the name of the file that is included. It can be quoted or unquoted. Note that include files may be nested.

The include file names are processed in the same way like the input file. The names are expanded using the working directory. If the file cannot be found and no extension is given, the standard GAMS input extension is tried. However, if an incomplete path is given, the file name is completed using the include directory. By default, the library include directory is set to the working directory. The default directory search path may be extended with the command line parameter InputDir.

Note that the start of the include file is marked and the include file is echoed to the echo print in the listing file. This reference to the include file may be omitted by using the option $offInclude.

Example:

```
$include myfile
$include "myfile"
```

Both statements above are equivalent and the search order for the include file is as follows:

1. `myfile` in current working directory
2. `myfile.gms` in current working directory
3. `myfile` and `myfile.gms` (in that order) in directories specified by the command line parameter InputDir.

Attention

The current settings of the dollar control options are passed on to the lower level include files. However, the dollar control options set in the lower level include file are passed on to the parent file only if the option $onGlobal is set.

Note that details on the compilation output of include files are given in section The Include File Summary.

See also $batInclude, $libInclude, $sysInclude.

$inlineCom (/* */)

Syntax:

```
$inlineCom char[char] char[char]
```

This option redefines and activates the in-line comment symbols. These symbols are placed at the beginning and the end of the in-line comment and are one character or a two character sequence at the beginning and the end. By default, the system is initialized to ' /*' and ' */', but is not active. The option $onInline is used to activate the in-line comments. If `$inlineCom` is used, $onInline is set automatically.

Example:

```
$inlineCom {{ }}
Set {{ this is an inline comment }} i / 1*2 / ;
```

Note that the character pairs {{ }} serve as the indicator for in-line comments.

Attention

It is not allowed to reset the option `$inlineCom` to the current symbol for in-line comments. This would cause an compilation error as in the following example:

```
$inlinecom {{  }}
$inlinecom {{  }}
```

Note

The option $onNestCom enables the use of nested comments.

The default for dealing with inline comments can be set using the command line parameter inlineCom.

See also section Comments.

$kill

Syntax:

```
$kill ident {ident}
```

This option removes all data for the identifiers `ident`, only the type and dimension are retained (this means that these identifiers will be declared but not defined anymore). Note that only the data of the following data types may be removed: sets, parameters, variables and equations. Note further that the data removal is carried out during compile time and not when the GAMS program executes.

Example:

```
Set i / 1*20 /;
Scalar a /2/;
$kill i a
```

Note that the effect of the third line above is that all data from `a` and `i` is removed, so the set `i` and the scalar `a` are declared, but not initialized or assigned to. Note that after `i` and `a` have been killed, a display statement for them will trigger an error. However, new data may be assigned to identifiers that were previously killed. Thus the following statements are valid if appended to the code above:

```
Set i / i1 *i3 /;
a = 7;
```

Observe that this option needs to be distinguished from the dollar control option $clear, that *resets* the data to the default values.

$label

Syntax:

```
$goto id
$label id
```

This option marks a line to be jumped to by a dollar control option $goto. Any number of labels may be used in files and not all of them need to be referenced. Re-declaration of a label identifier will not generate an error and only the first occurrence encountered by the GAMS compiler will be used for future $goto references.

Example:

```
Scalar a ;
a = 5 ;
display a ;
$goto next
a = a+5 ;
display a ;
$label next
a = a+10 ;
display a ;
```

When GAMS reaches the line `$goto next`, it continues from the line `$label next`. All lines in between are ignored. Therefore in the example above, the final value of `a` is 15.

Attention

If several dollar control options appear in one line and `label` is one of them, then `label` must be listed first.

See also $goto, $maxGoto.

$libInclude

Syntax:

```
$libInclude external_file {arg}
```

This option is mostly equivalent to the option $batInclude. However, if an incomplete path is given, the file name is completed using the library include directory. By default, the library include directory is set to the directory `inclib` in the GAMS standard locations. Note that the list of default directories searched may be prepended with the command line parameter libIncDir.

Example:

```
$libInclude abc x y
```

This call will first look for the include file `inclib/abc` in the standard locations. If this file does not exist there GAMS will look for the file `inclib/abc.gms` in the standard locations. The arguments `x` and `y` are passed on to the include file and are interpreted as explained in the detailed description of the option $batInclude.

See also $include, $batInclude, $sysInclude.

$lines

Syntax:

```
$lines n
```

This option starts a new page in the listing file if less than `n` lines are available on the current page.

Example:

```
$hidden Never split the first few lines of the following table
$lines 5
Table io(i,j) Transaction matrix
...
;
```

This will ensure that if there are less than five lines available on the current page in the listing file before the next statement (in this case, the table statement) is echoed to it, the contents of this statement are echoed to a new page.

$load

Syntax:

```
$load [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option is preceded and succeeded by the option $gdxIn that open the GDX file for reading. The option `$load` loads specified items from the GDX file. Note that more than one instance of `$load` may occur. A listing of the GDX file contents will be created if the option `$load` is not followed by arguments.

Examples

Consider the following example, where `transsol` is the GDX file of the transportation model [**TRNSPORT**]

```
$gdxIn transsol
$load
Sets i, j; Parameters a(i), b(j), d(i,j), f;
$load i j a b d f
$gdxIn
```

A comma between the symbols is optional. The follow example works identically:

```
$gdxIn transsol
$load
Sets i, j; Parameters a(i), b(j), d(i,j), f;
$load i, j, a, b, d, f
$gdxIn
```

The $load without any arguments produces a table of contents of the GDX container in the listing file:

```
Content of GDX C:\Users\default\Documents\gamsdir\projdir\transsol.gdx
               5 UELs

    Number Type       Dim       Count  Name
         1 Set          1           2  i           canning plants
         2 Set          1           3  j           markets
         3 Parameter    1           2  a(i)        capacity of plant i in cases
         4 Parameter    1           3  b(j)        demand at market j in cases
         5 Parameter    2           6  d(i,j)      distance in thousands of miles
         6 Parameter    0           1  f           freight in dollars per case per thousand mile
         7 Parameter    2           6  c(i,j)      transport cost in thousands of dollars per ca
         8 Variable     2           6  x(i,j)      shipment quantities in cases
         9 Variable     0           1  z           total transportation costs in thousands of do
        10 Equation     0           1  cost        define objective function
        11 Equation     1           2  supply(i)   observe supply limit at plant i
        12 Equation     1           3  demand(j)   satisfy demand at market j
```

Symbols may be loaded with new names with the following syntax: $load i=gdx_i j=j_gdx. The universal set may be loaded using $load uni=*.

```
$gdxIn transsol
Sets i, jj, uni; Parameters a(i), bb(jj), d(i,jj), f;
$load i jj=j uni=* a bb=b d f
$gdxIn
display uni;
```

This results in a display of all used labels:

```
----      5 SET uni
Seattle  ,    San-Diego,    New-York ,    Chicago  ,    Topeka
```

The syntax sym<[=]GDXSym[.dimI] allows to load a one dimensional set from a symbol in the GDX file that has even a higher dimensionality. GAMS tries to find the set sym as a domain in the symbol GDXSym and uses the labels from this index position (with < the first domain set from the right and with <= from the left). If no domain information is stored in the GDX file or the domain information does not match the suffix .dimI allows to pick a fixed index position.

In the following we work with a GDX file created by the following code:

```
set i / i1*i3 /, ii(i,i) / i1.i2, i2.i3 /;
execute_unloadDI 'ii', i, ii;
```

Now use use this GDX file to load the first and second index from ii:

```
set i, i1;
$gdxIn ii
* Load first index from ii as i
$load i<=ii i1<ii.dim1
display i, i1;
```

the display lists all labels from the first index of ii:

```
----      5 SET i  Domain loaded from ii position 1
i1,    i2

----      5 SET i1  Domain loaded from ii position 1
i1,    i2
```

Now we match from the right and get the second index of ii:

```
set i, i2;
$gdxIn ii
* Load second index from ii as i
$load i<ii i2<ii.dim2
display i, i2;
```

The resulting listing file will contain the following lines:

```
----      5 SET i  Domain loaded from ii position 2
i2,    i3

----      5 SET i2  Domain loaded from ii position 2
i2,    i3
```

This type of projection loading can be useful to extract the domain sets from a single parameter that is stored in a GDX file:

```
set i,j,k; parameter data(i,j,k);
$gdxIn data
$load i<data.dim1 j<data.dim2 k<data.dim3 data
```

Attention

Loading an item that was already initialized will cause a compilation error.

For example, the following code snippet will cause a compilation error:

```
Set j / 1*5 /;
$gdxIn transsol
$load j
$gdxIn
```

Note that GAMS offers variants of $load that do not cause a compilation error in such a case: $loadM and $loadR.

Note

One can load the level values of a variable into a parameter of the same dimension using the syntax parametername=var.l as follows

```
parameter storexlevel(i,j);
$gdxin trnsport
$load storexlevel=x.l
$gdxin
display storexlevel;
```

Loading the level values for a variable into the same variable (x=x.l) works but loads everything including bounds, scales, marginals and levels.

$loadDC

Syntax:

```
$loadDC [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option is an alternative form of $load. It performs domain checking when items are loaded. Any domain violations will be reported and flagged as compilation errors. This is basically the opposite of $loadFiltered. All other features are the same as discussed under $load .

Example: Consider the following example where `transsol` is the GDX file of the transportation model [**TRNSPORT**].

```
Set i, j;
Parameter b(i), a(j);
$gdxIn transsol
$load i b
$loadDC j a
$gdxIn
```

Note that in contrast to the example above, the parameter `a` is indexed over the set `i` and the parameter `b` is indexed over the set `j` in the file `transsol`. While `$load i b` does not generate an error and `b` is just empty, the option `$loadDC j a` triggers a domain violation error because in `transsol` `a` is indexed over `i` and produces a list of errors in the listing file:

```
--- LOAD  a = 3:a
**** Unique domain errors for symbol a
 Dim  Elements
   1  seattle, san-diego

   5  $loadDC j a
****           $649
```

$loadDCM

Syntax:

```
$loadDCM [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option combines the functionality of *merging* as in $loadM and domain checking as in $loadDC.

Example:

Consider the following example where `transsol` is the GDX file of the transportation model [**TRNSPORT**].

```
Set i, uni 'all labels';
Parameter abFail(i), ab(uni) 'capacity and demand';
$gdxIn transsol
$load i abFail=a
$loadDCM abFail=b
$loadDCM uni=i uni=j ab=a ab=b
$gdxIn
display uni, ab;
```

Here we try to merge parameters `a` and `b` together into one parameter. The first attempt (to merge it into parameter `abFail`) would fail because of line 5 and result into a domain violation report as described with dollar control option $loadDC. In the second attempt we first merge the sets `i` and `j` into set `uni` and then merge the parameters `a` and `b` into `ab`. If one comments line 5 the resulting display looks as follows:

```
----      8 SET uni  all labels
seattle  ,    san-diego,    new-york ,    chicago  ,    topeka

----      8 PARAMETER ab  capacity and demand
seattle   350.000,    san-diego 600.000,    new-york  325.000
chicago   300.000,    topeka    275.000
```

$loadDCR

Syntax:

```
$loadDCR [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option combines the functionality of *replacing* data as in $loadR and domain checking as in $loadDC.

Example:

Consider the following example where `transsol` is the GDX file of the transportation model [**TRNSPORT**].

```
Set uni 'all labels';
Parameter ab(uni) 'capacity and demand';
$gdxIn transsol
$loadM uni=i uni=j ab=a
$loadDCR ab=b
$gdxIn
display uni, ab;
```

Here we try to read twice into the parameter `ab`. First GDX symbol `a` and `b` are read into `ab`. GDX symbol `b` is read with *replace* and hence the parameter `ab` contains the elements of `b` only.

$loadFiltered

Syntax:

```
$loadFiltered [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option is an alternative form of $load, in particular, it works like `$load` while $onFiltered is set. That means, when items are loaded, domain violations are not flagged as compilation errors, but items violating the domain definition are just filtered out. This is basically the opposite of $loadDC. All other features are the same as discussed under $load .

Example: Consider the following example where `transsol` is the GDX file of the transportation model [**TRNSPORT**].

```
Set i / seattle /
    j / new-york, chicago /;
Parameter a(i), b(j);
$gdxIn transsol
$loadFiltered a
$loadDC       b
$gdxIn
```

While `$loadFiltered a` does not generate an error, but just ignores the data that does not match the defined domain, `$loadDC b` triggers a domain violation error.

See also $loadFilteredM and $loadFilteredR.

$loadFilteredM

Syntax:

```
$loadFilteredM [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option combines the functionality of *merging* as in $loadM and domain filtering as in $loadFilterd.

Example:

Consider the following example where `transsol` is the GDX file of the transportation model [**TRNSPORT**].

```
Set i, uni 'all labels';
Parameter abFilter(i), ab(uni) 'capacity and demand';
$gdxIn transsol
$load          i abFilter=a
$loadFilteredM abFilter=b
$loadFilteredM uni=i uni=j ab=a ab=b
$gdxIn
Display uni, ab, abFilter;
```

Here we try to merge parameters `a` and `b` together into one parameter. The first attempt (to merge it into parameter `abFilter`) does not add anything, since the elements in b do not match the defined domain `i` (compare $loadFilterd). In the second attempt we first merge the sets `i` and `j` into set `uni` and then merge the parameters `a` and `b` into `ab`. The `Display` statement produces the following output:

```
----     31 SET uni  all labels

seattle  ,    san-diego,    new-york ,    chicago  ,    topeka


----     31 PARAMETER ab  capacity and demand

seattle   350.000,    san-diego 600.000,    new-york  325.000,    chicago   300.000,    topeka


----     31 PARAMETER abFilter  capacity of plant i in cases

seattle   350.000,    san-diego 600.000
```

$loadFilteredR

Syntax:

```
$loadFilteredR [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option combines the functionality of *replacing* data as in $loadR and domain filtering as in $loadFilterd.

Example:

Consider the following example where `transsol` is the GDX file of the transportation model [**TRNSPORT**].

```
Set       uni     'all labels';
Parameter ab(uni) 'capacity and demand';
$gdxIn transsol
$loadFilteredM uni=i uni=j ab=a
$loadFilteredR ab=b
$gdxIn

Display uni, ab;
```

Here we try to read twice into the parameter `ab`. First GDX symbol `a` and `b` are read into `ab`. GDX symbol `b` is read with *replace* and hence the parameter `ab` contains the elements of `b` only:

```
----     31 SET uni  all labels

seattle  ,    san-diego,    new-york ,    chicago  ,    topeka


----     31 PARAMETER ab  capacity and demand

new-york 325.000,    chicago  300.000,    topeka   275.000
```

$loadIdx

Syntax:

```
$loadIdx [sym1[,] sym2=gdxSym2[,] ...]
```

This option is an alternative form of $load. Each symbol in the GDX file to read must be a parameter and should have been written using an indexed write; see also execute_unloadidx.

See Example 3 - Reading a GDX File and [**ldidx01**] on how to use `$loadidx`.

$loadM

Syntax:

```
$loadM [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option is an alternative form of $load. Instead of replacing an item or causing a *symbol redefined* error if the item was already initialized it *merges* the contents. Records that would result in domain violations will be ignored.

Example:

Consider the following example where `transsol` is the GDX file of the transportation model [**TRNSPORT**].

```
Set i, uni 'all labels';
Parameter ab(uni) 'capacity and demand';
$gdxIn transsol
$loadM uni=i uni=j ab=a ab=b
$gdxIn
display uni, ab;
```

Here we merge parameters `a` and `b` together into one parameter `ab`. We first merge the sets `i` and `j` into set `uni` and then merge the parameters `a` and `b` into `ab`. The resulting display looks as follows:

```
----      6 SET uni  all labels
seattle  ,    san-diego,    new-york ,    chicago  ,    topeka

----      6 PARAMETER ab  capacity and demand
seattle   350.000,    san-diego 600.000,    new-york  325.000
chicago   300.000,    topeka    275.000
```

$loadR

Syntax:

```
$loadR [sym1[,] sym2=gdxSym2[,] sym3<[=]gdxSym3[.dimI][,] ...]
```

This option is a variant of the option $load. With `$loadR` we can have multiple loads into the same symbols and the data stored in GAMS will be replaced with the one from the GDX container.

Example:

Consider the following example, where `transsol` is the GDX file of the transportation model [**TRNSPORT**]:

```
Sets i  / 1*3 /
     j  / 1*2 /;
$gdxIn transsol
$loadR i j
$gdxIn
display i, j;
```

The resulting listing file will contain the following lines:

```
----      6 SET i  canning plants
Seattle  ,    San-Diego

----      6 SET j  markets
New-York,    Chicago ,    Topeka
```

$log

Syntax:

```
$log text
```

This option will send a message `text` to the log file. Recall that by default, the log file is the console. The default log file may be reset with the command line parameters logOption and logFile.

Attention

- Leading blanks are ignored when the text is written out to the log file as a result of using the `$log` option.
- All special `%` symbols will be substituted before the text passed through the `$log` option is sent to the log file.

Example:

```
$log
$log       The following message will be written to the log file
$log  with leading blanks ignored. All special % symbols will
$log  be substituted before this text is sent to the log file.
$log  This was line %system.incLine% of file %system.incName%
$log
```

The log file that results by running the lines above will contain the following lines:

```
The following message will be written to the log file
with leading blanks ignored. All special % symbols will
be substituted before this text is sent to the log file.
This was line 5 of file C:\tmp\logTest.gms
```

Note that `%system.incLine%` is replaced by 5 which is the line number where the string replacement was requested. Note further that `%system.incName%` is substituted with the name of the file completed with the absolute path. Observe that the leading blanks on the second line of the example are ignored.

$macro

Syntax:

```
$macro name(arg1,arg2,arg3, ...) macro_body
```

This option defines a macro in GAMS. Here `name` is the name of the macro, `arg1,arg2,arg3,...` are the arguments and `macro_body` defines what the macro should do. The macro names follow the rules for identifiers. The macro `name` cannot be used for other symbols. For further details and examples, see section Macros in GAMS below.

$maxCol (80001)

Syntax:

```
$maxCol n
```

This option restricts the valid range of input columns at the *right* margin. Note that all input after column `n` is treated as comment, therefore it is ignored.

Example:

```
$maxCol 30
Set    i  / vienna, rome /;   set definition
Scalar a  / 2.3          /;   scalar definition
```

Observe that the text strings `set definition` and `scalar definition` are treated as comments and are ignored since they begin on or after column 31.

Any changes in the margins via `$maxCol` or $minCol will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option `$minCol 20 maxCol 110` will trigger the following message:

```
NEW MARGINS: 20-110
```

Note

- GAMS requires that the right margin set by `$maxCol` is greater than 15.
- GAMS requires that the right margin set by `$maxCol` is greater than the left margin set by $minCol.

See also $on/offMargin and section Comments.

$maxGoto 100

Syntax:

```
$maxGoTo n
```

This option sets the maximum number of jumps to the same label and is used in the context of the options $goTo and $label. Once the maximum number is reached a compilation error is triggered. Such a limit has been implemented to avoid infinite loops at compile time.

Example:

```
Scalar a / 1 /;
$maxGoTo 5
$label label1
a = a+10;
display a ;
$goTo label1
```

Note that a compilation error is triggered if `$goTo label1` is called for the fifth time.

$minCol (1)

Syntax:

```
$minCol n
```

This option restricts the valid range of input columns at the *left* margin. Note that all input before column `n` is treated as comment, therefore it is ignored.

Example:

```
$minCol 30
Set definition              Set    i / vienna, rome /;
Scalar definition           Scalar a / 2.3          /;
```

Observe that the text strings `set definition` and `scalar definition` are treated as comments and are ignored since they are placed before column 30.

Any changes in the margins via the option $maxCol or `$minCol` will be reported in the listing file with the message that gives the valid range of input columns. For example, the dollar control option `$minCol 20 maxCol 110` will trigger the message:

```
NEW MARGINS: 20-110
```

Attention

GAMS requires that the left margin set by the option `$minCol` is smaller than the right margin set by the option $maxCol.

See also $on/offMargin and section Comments.

$(on|off)CheckErrorLevel ($offCheckErrorLevel)

Syntax:

```
$onCheckErrorLevel
$offCheckErrorLevel
```

If the option `$onCheckErrorLevel` is active, the errorLevel is checked implicitly after $call, $hiddenCall, $callTool and $hiddenCallTool. A compilation error is triggered and the compilation is aborted if it is not 0. With `$offCheckErrorLevel`, this is not done, but one can do it explicitly. So, with `$onCheckErrorLevel` $call behaves like $call.checkErrorLevel, $hiddenCall behaves like $hiddenCall.checkErrorLevel, $callTool behaves like $callTool.checkErrorLevel and $hiddenCallTool behaves like $hiddenCallTool.checkErrorLevel.

The default for dealing with error levels can be set using the command line parameter checkErrorLevel.

$(on|off)Delim ($offDelim)

Syntax:

```
$onDelim
$offDelim
```

This option controls whether data in table statements may be entered in delimited format. As delimiter one can use `"."`, `","`, or `"TAB"`.

Example:

```
Sets  plant  'plant locations' / NEWYORK, CHICAGO, LOSANGELES /
      market 'demands'         / MIAMI, HOUSTON, PORTLAND     /;

Table dist(plant,market)
$onDelim
,MIAMI,HOUSTON,PORTLAND
NEWYORK,1300,1800,1100
CHICAGO,2200,1300,700
LOSANGELES,3700,2400,2500
$offDelim
;

Display dist;
```

The resulting listing file will contain the following output:

```
----     12 PARAMETER dist

                MIAMI     HOUSTON    PORTLAND

NEWYORK      1300.000    1800.000    1100.000
CHICAGO      2200.000    1300.000     700.000
LOSANGELES   3700.000    2400.000    2500.000
```

$(on|off)Digit ($onDigit)

Syntax:

```
$onDigit
$offDigit
```

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

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

The `$offDigit` control remains useful for changing how GAMS handles numbers with very large or small magnitude. With `$onDigit`, GAMS throws compiler warnings/errors for these values. With `$offDigit`, GAMS quietly converts them to UNDF or EPS, respectively.

Example:

```
scalar aboveTheLimit / 222e299 /;
scalar tiny / 1.0e-275 /;
scalar tinier / 1.0e-285 /;
$offdigit
scalar uberTheLimit / 222e299 /;
scalar klein   / 1.0e-275 /;
scalar kleiner / 1.0e-285 /;
```

The resulting listing file will contain the following lines:

```
   1  scalar aboveTheLimit / 222e299 /;
****                                $106
   2  scalar tiny / 1.0e-275 /;
****                         $595
   3  scalar tinier / 1.0e-285 /;
****                           $594
   5  scalar uberTheLimit / 222e299 /;
   6  scalar klein   / 1.0e-275 /;
   7  scalar kleiner / 1.0e-285 /;
Error Messages
106  abs(number) out of range
        abs(number) must be less than 1E300
        $offdigit can be used to replace large numbers with UNDF
594  Exponent very very small - rejected
       $offdigit can be used to replace with EPS
595  Exponent very small
       $offdigit can be used to quietly accept very small values
```

The default for dealing with digits can be set using the command line parameter digit.

$(on|off)Dollar ($offDollar)

Syntax:

```
$onDollar
$offDollar
```

This option controls the echoing of dollar control option lines in the listing file.

Example:

```
$hidden This line will not be displayed
$onDollar
$hidden This line will be displayed
$offDollar
$hidden This line will not be displayed
```

The compilation output of the resulting listing file will contain the following lines:

```
2  $onDollar
3  $hidden This line will be displayed
```

Note that all lines between the option `$onDollar` and the option `$offDollar` are echoed in the listing file. Note further that the effect of this option is immediate: the line `$onDollar` is echoed in the listing file, while the line `$offDollar` is not.

$(on|off)DotL ($offDotL)

Syntax:

```
$onDotL
$offDotL
```

This option activates or deactivates the automatic addition of the attribute `.L` to variables on the right-hand side of assignments and in put statement. It is most useful in the context of macros. For further information, see section Macros in GAMS below.

Example:

Consider the following code snippet which can be added to the end if the **[TRNSPORT]** model:

```
$onDotL
file fx; put fx;
loop((i,j),
   put / x(i,j)
);
$offDotL
file fx; put fx;
loop((i,j),
   put / x(i,j)
);
```

This is the generated output:

```
        50.00
       300.00
         0.00
       275.00
         0.00
       275.00
x('seattle','new-york')
x('seattle','chicago')
x('seattle','topeka')
x('san-diego','new-york')
x('san-diego','chicago')
x('san-diego','topeka')
```

First, we see the `level` for each `x` record. After setting `$offDotL` we are back at the default which means, that the .tn attribute is used implicitly.

$(on|off)DotScale ($offDotScale)

Syntax:

```
$onDotScale
$offDotScale
```

This option activates or deactivates the automatic addition of the attribute `.Scale` to variables and equations on the right-hand side of assignments and in put statements. As with on|offDotL It is most useful in the context of macros. For further information, see section Macros in GAMS below.

$(on|off)Echo[S|V][.tag]

Syntax:

```
$onEcho[S|V][.tag]  >[>] external_file
text
{text}
$offEcho[.tag]
```

This option is used to send one or more lines of `text` to an file `external_file`. The text and the file name may be quoted or unquoted. The external file is not closed until the end of the compilation or when the option $call or any variant of the option $include is encountered. Note that the redirection symbols > and >> have the usual meaning: > creates a new file and writes to it or - in case there exists already a file with the respective name - overwrites the existing file and >> appends to a file. Note further that parameter substitutions are permitted with `$onEcho`. The option `$onEcho` has two more variants: `$onEchoS` and `$onEchoV`. `$onEchoS` allows parameter substitutions like `$onEcho`, so it is just a synonym which makes it more obvious that parameter substitution is allowed with the appended `S`. The option `$onEchoV` does not allow parameter substitutions but writes the text verbatim.

Example:

```
$set it TEST
$onEchoS  > externalfile1.txt
send %it% to external file
line 2 to send
$offEcho

$onEchoV  > externalfile2.txt
send %it% to external file
line 2 to send
$offEcho
```

The `externalfile1.txt` will contain the following lines:

```
send TEST to external file
line 2 to send
```

The `externalfile2.txt` will contain these lines:

```
send %it% to external file
line 2 to send
```

Observe that in the first case `%it%` is substituted with `TEST`, but in the second case there is no substitution.

Note that by default the external file will be placed in the current working directory if there is no path specified.

One may add a `tag` to the `$onEcho[S|V]` and `$offEcho`. This allows to use nested echo blocks like this:

```
$onEcho.outer > outerFile.gms
$onEcho.inner > innerFile.txt
Line 1 of text
Line 2 of text
$offEcho.inner
$offEcho.outer
```

The resulting file `outerFile.gms` looks like this:

```
$onEcho.inner > innerFile.txt
Line 1 of text
Line 2 of text
$offEcho.inner
```

See also options $echo, and $echoN.

$(on|off)ECImplicitLoad ($onECImplicitLoad)

Syntax:

```
$onECImplicitLoad
$offECImplicitLoad
```

This option enables or disables the implicit loading of symbols set in embedded code sections, which are not loaded explicitly with $offEmbeddedCode.

Example:

```
Set i(*);

$offECImplicitLoad
$onEmbeddedCode Python:
i = ['a','b']
print(i)
gams.set("i", i)
$offEmbeddedCode
```

This will result in a compilation error, since `i` is set in the embedded code block (by `gams.set("i", i)`), but not loaded with `$offEmbeddedCode`. If `$onECImplicitLoad` would be active, `$offEmbeddedCode` would be treated like `$offEmbeddedCode i` here.

See also ECImplicitLoad.

$(on|off)Embedded ($offEmbedded)

Syntax:

```
$onEmbedded
$offEmbedded
```

This option enables or disables the use of embedded values in parameter and set data statements. If enabled, the explanatory text for set elements is concatenated with blank separators. For parameters, the embedded values get multiplied.

Example:

```
Set k / a,b /
    l / a   /;
Set       i(k,l) / a.a 'aaaa cccc dddd'
                   b.a 'bbbb cccc dddd' /;
Parameter m(k,l) / a.a 12
                   b.a 24 /;
$onEmbedded
Set       j(k,l) / (a aaaa, b bbbb).(a cccc) dddd /;
Parameter n(k,l) / (a 1,    b 2)   .(a 3)    4    /;
```

Note that the explanatory text of the set elements in `i` and `j` as well as the values of the parameters `m` and `n` are identical.

$(on|off)EmbeddedCode[S|V][.tag]

Syntax:

```
$onEmbeddedCode[S|V][.tag] Connect|Python: [arguments]
Embedded code
{Embedded code}
$offEmbeddedCode[.tag] {symbol[<[=]embSymbol[.dimX]]}
```

This option is used to execute one or more lines of embedded code (e.g. Python code) while GAMS stays alive. The embedded code has access to GAMS symbols and can read and change them.

Note that parameter substitutions are permitted with `$onEmbeddedCode`. The option `$onEmbeddedCode` has two more variants: `$onEmbeddedCodeS` and `$onEmbeddedCodeV`. `$onEmbeddedCodeS` allows parameter substitutions like `$onEmbeddedCode`, so it is just a synonym which makes it more obvious that parameter substitution is allowed with the appended `S`. The option `$onEmbeddedCodeV` does not allow parameter substitutions but passes the code verbatim to the embedded code engine. The optional `arguments` given to `$onEmbeddedCode[S|V]` can be accessed in the Python code.

`$offEmbeddedCode` can be followed by a GAMS symbol or a list of GAMS symbols. If GAMS symbols are specified they get updated in the GAMS database after the embedded code got executed. The syntax `symbol<[=]embSymbol[.dimX]` allows to load a one dimensional set from a symbol which was set in the embedded code that has even a higher dimensionality (here we call `<[=]` the `projection operator`). GAMS tries to find the set `symbol` as a domain in the symbol `embSymbol` and uses the labels from this index position (with `<` the first domain set from the right and with `<=` from the left). If no domain information is stored in the GDX file or the domain information does not match the suffix `.dimX` allows to pick a fixed index position (`X` needs to be replaced by the desired index position).

One may add a `tag` to the `$onEmbeddedCode[S|V]` and `$offEmbeddedCode`. For example, `$onEmbeddedCode.tagOne` would match with `$offEmbeddedCode.tagOne`.

Example:

```
Set cc      / "France - Paris", "France - Lille", "France - Toulouse"
              "Spain - Madrid", "Spain - Cordoba", "Spain - Seville", "Spain - Bilbao"
              "USA - Washington DC", "USA - Houston", "USA - New York",
              "Germany - Berlin", "Germany - Munich", "Germany - Bonn" /
    country
    city
    mccCountry(cc,country<)  Mapping between country and related elements in set cc
    mccCity(cc,city<)        Mapping between city and related elements in set cc;

$onEmbeddedCode Python:
mccCountry = []
mccCity = []
for cc in gams.get("cc"):
  r = str.split(cc, " - ", 1)
  mccCountry.append((cc,r[0]))
  mccCity.append((cc,r[1]))
gams.set("mccCountry",mccCountry)
gams.set("mccCity",mccCity)
$offEmbeddedCode mccCountry mccCity

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry, mccCity;
```

The data definition of the sets `country` and `city` happen while loading the set `mccCountry` and `mccCity` implicitly because of the implicit set definition in declarations of `mccCountry` and `mccCity`. Also note, that the list of output symbols does not need to be explicitly specified (for most embedded code engines). Performing the actual write in the embedded code (here via `gams.set`) is sufficient to signal GAMS to load the symbols. Providing an explicit list of symbols can be required (for embedded code engines without implicit loading) and be advantageous for code readability even for embedded code engines that support this. With the dollar control option $offECImplicitLoad providing an explicit list of output symbols can be enforced and setting symbols inside the embedded code not in the explicit list will lead to compilation errors.

The program above will result in the following listing file:

```
----     25 SET country
Spain  ,    USA    ,    Germany,    France

----     25 SET city
Berlin       ,    Bilbao       ,    Cordoba      ,    Madrid
New York     ,    Washington DC,    Paris        ,    Houston
Munich       ,    Lille        ,    Seville      ,    Bonn
Toulouse

----     25 SET mccCountry
France - Paris     .France
France - Lille     .France
France - Toulouse  .France
Spain - Madrid     .Spain
Spain - Cordoba    .Spain
Spain - Seville    .Spain
Spain - Bilbao     .Spain
USA - Washington DC.USA
USA - Houston      .USA
USA - New York     .USA
Germany - Berlin   .Germany
Germany - Munich   .Germany
Germany - Bonn     .Germany

----     25 SET mccCity
France - Paris     .Paris
France - Lille     .Lille
France - Toulouse  .Toulouse
Spain - Madrid     .Madrid
Spain - Cordoba    .Cordoba
Spain - Seville    .Seville
Spain - Bilbao     .Bilbao
USA - Washington DC.Washington DC
USA - Houston      .Houston
USA - New York     .New York
Germany - Berlin   .Berlin
Germany - Munich   .Munich
Germany - Bonn     .Bonn
```

Using the `projection operator` on the `$offEmbeddedCode` line the same task could be done like this:

```
Set cc      / "France - Paris", "France - Lille", "France - Toulouse"
              "Spain - Madrid", "Spain - Cordoba", "Spain - Seville", "Spain - Bilbao"
              "USA - Washington DC", "USA - Houston", "USA - New York",
              "Germany - Berlin", "Germany - Munich", "Germany - Bonn" /
    country
    city
    mccCountry(cc,country)  Mapping between country and related elements in set cc
    mccCity(cc,city)        Mapping between city and related elements in set cc;

$onEmbeddedCode Python:
mccCountry = []
mccCity = []
for cc in gams.get("cc"):
  r = str.split(cc, " - ", 1)
  mccCountry.append((cc,r[0]))
  mccCity.append((cc,r[1]))
gams.set("mccCountry",mccCountry)
```

```
gams.set("mccCity",mccCity)
$offEmbeddedCode country<mccCountry city<mccCity

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry, mccCity;
```

Please note that due to implicit output symbol loading, see $offECImplicitLoad, the symbols `mccCountry` and `mccCity` are not only used to define `country` and `city` but are also loaded into GAMS because the embedded code uses `gams.set` to write to these symbols.

See also chapter Embedded Code Facility for more details.

$(on|off)Empty ($onEmpty)

Syntax:

```
$onEmpty
$offEmpty
```

`$onEmpty` allows empty data statements for list or table formats (default). When setting `$offEmpty`, empty data statements will cause a compilation error.

Example:

```
Set       i    / 1,2,3 / ;
Set       j(i) /       / ;
Parameter x(i)   "empty parameter" /  / ;
Table     y(i,i) "headers only"
     1   2   3
;
$offEmpty
*offEmpty from here on
Set       k(i) /  / ;
Parameter xx(i)   "empty parameter" /  / ;
Table     yy(i,i) "headers only"
     1   2   3
;
```

The resulting listing file will contain the following lines:

```
   1  Set       i    / 1,2,3 / ;
   2  Set       j(i) /       / ;
   3  Parameter x(i)   "empty parameter" /  / ;
   4  Table     y(i,i) "headers only"
   5       1   2   3
   6  ;
   8  *offEmpty from here on
   9  Set       k(i) /  / ;
****                    $460
**** 460  Empty data statements not allowed. You may want to use $ON/OFFEMPTY
  10  Parameter xx(i)   "empty parameter" /  / ;
****                                      $460
**** 460  Empty data statements not allowed. You may want to use $ON/OFFEMPTY
  11  Table     yy(i,i) "headers only"
  12       1   2   3
  13  ;
****  $462
**** 462  The row section in the previous table is missing
```

Empty data statements are most likely to occur when data is being entered into the GAMS model by an external program.

Note

The empty data statement may only be used with symbols which have a known dimension. If the dimension is also derived from the data, the option $phantom should be used to generate 'phantom' set elements.

The option `$onEmpty` in conjunction with the option $onMulti and the save and restart feature may be used to set up a model and add data later.

The default for dealing with empty data statements can be set using the command line parameter empty.

$(on|off)End ($offEnd)

Syntax:

```
$onEnd
$offEnd
```

This option offers an alternative syntax for flow control statements. The option `$onEnd` causes the following words to be regarded as keywords: `do`, `endLoop`, `endIf`, `endFor` and `endWhile`. They are used to close the language constructs `loop`, `if`, `for` and `while` respectively.

Example:

Note

The standard syntax is given as an end-of-line comment.

```
Set       i    / 1*3 /;
Scalar    cond / 0 /;
Parameter a(i) / 1 1.23, 2 2.65, 3 1.34/;

$eolCom //
$onEnd

loop i do                                  // loop (i,
  display a;                               //   display a;
endLoop;                                   // );

if (cond) then                             // if (cond,
  display a;                               //   display a;
else                                       // else
  a(i) = a(i)/2;                           //   a(i) = a(i)/2;
  display a;                               //   display a;
endIf;                                     // );

for cond = 1 to 5 do                       // for (cond = 1 to 5,
  a(i) = 2 * a(i);                         //   a(i) = 2 * a(i);
endFor;                                    // );

while cond > 3 do                          // while (cond > 3,
  a(i) = a(i) / 2;                         //   a(i) = a(i) / 2;
  cond = cond-1;                           //   cond = cond-1;
endWhile;                                  // );
```

Observe that the alternative syntax is more in line with the syntax used in some of the popular programming languages.

Attention

Setting the option `$onEnd` will make the alternative syntax valid, and at the same time it will make the standard syntax invalid. Therefore the two forms of the syntax will never be valid simultaneously.

$(on|off)EolCom ($offEolCom)

Syntax:

```
$onEolCom
$offEolCom
```

This option acts as a switch to control the use of end-of-line comments. Note that by default, the end-of-line comment symbol is set to `!!` but the processing is disabled.

Example:

```
$onEolCom
Set       i    /1*2/ ;  !! set declaration
Parameter a(i)       ;  !! parameter declaration
```

Observe that after the option `$onEolCom` has been specified, comments may be entered on the same line as GAMS code.

Note

The option $eolCom automatically sets `$onEolCom`.

The default for dealing with end-of-line comments can be set using the command line parameter eolCom.

See also section Comments.

$(on|off)Eps ($offEps)

Syntax:

```
$onEps
$offEps
```

This option is used to interpret zero values as `EPS` when non-scalar parameters, variable attributes, or equation attributes are defined or loaded (e.g. from GDX or embedded code) at compile time. This can for example be useful if the value of zero is overloaded with existence interpolation.

See option zeroToEps for the execution-time equivalent.

Example 1:

Consider the following example where zero values are defined for a one dimensional parameter, once under `$onEps` and `$offEps`.

```
Set       i    / one, two, three, four /;
Parameter a(i) /
$onEps
      one     0
$offEps
      two     0
      three   EPS   /;
Display a ;
```

The outcome generated by the display statement follows:

```
----      8 PARAMETER a

one   EPS,    three EPS
```

Note that only those entries specifically entered as 0 are treated as EPS.

Example 2:

Consider the following example where zero values are loaded from a GDX file, once under $offEps (default) and $onEps (derived from test library model [**ZEROTOEPS1**]).

```
Set s,t;
Parameter p(s<,t<)
          pOnEps(s,t);
$gdxIn zeropar.gdx
$onText
P looks like this:
   t1        t2        t3
s1 0         0.843267
s2 0.224053  0.349831  0
s3           0         5
$offText
$load p
$onEps
$load pOnEps=p
display p, pOnEps;
```

The outcome generated by the display statement follows:

```
----     15 PARAMETER p

            t1          t2          t3

s1                    0.843
s2        0.224       0.350
s3                                5.000


----     15 PARAMETER pOnEps

            t1          t2          t3

s1          EPS       0.843
s2        0.224       0.350         EPS
s3                      EPS       5.000
```

Example 3:

Consider the following example where zero values are loaded from embedded Python code, once under `$offEps` (default) and `$onEps` (derived from test library model [**ZEROTOEPS1**]).

```
Set s / s1*s3 /;
Parameter pEC(s)
          pECOnEps(s);
$onEmbeddedCode Python:
p = [('s1',1), ('s3',0)]
gams.set('pEC',p)
$offEmbeddedCode pEC
$onEps
$onEmbeddedCode Python:
p = [('s1',1), ('s3',0)]
gams.set('pECOnEps',p)
$offEmbeddedCode pECOnEps

display pEC, pECOnEps;
```

The outcome generated by the display statement follows:

```
----     14 PARAMETER pEC

s1 1.000


----     14 PARAMETER pECOnEps

s1 1.000,    s3   EPS
```

$(on|off)EpsToZero ($offEpsToZero)

Syntax:

```
$onEpsToZero
$offEpsToZero
```

This option is used to write `EPS` values as zero when parameters, or variable and equation levels get written to GDX at compile time.

See option EpsToZero for the execution-time equivalent.

Example:

Usually GAMS squeezes out 0 values when writing to GDX. To get them written anyway, one can define them as `EPS` (e.g. by using `$onEps`) and set `$onEpsToZero`:

```
Set       i    / zero, one, two /;
$onEps
Parameter a(i) / zero    0
                 one     1
                 two     2  /;

$onEpsToZero
$gdxUnload 'zero.gdx' a
```

$(on|off)Expand ($offExpand)

Syntax:

```
$onExpand
$offExpand
```

This option changes the processing of macros that appear in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch `$onExpand` enables the recognition and expansion of macros in the macro argument list and `$offExpand` will restore the default behavior.

Example:

```
variable x(*,*);
$macro  f(i)   sum(q, x(i,q))
$macro equ(x)  equation equ_&x; equ_&x.. &x =e= 0;
equ(f(i))
```

The macro expansion of the code above will result in an equation definition that reads as follows:

```
equation equ_f(I); equ_f(i).. f(i) =e= 0;
```

If we compile the code under `$onExpand` the argument `f(i)` is expanded before the macro `equ()` gets expanding resulting in the following (incorrect) code:

```
equation equ_sum(q, x(i,q)); equ_sum(q, x(i,q)).. sum(q, x(i,q)) =e= 0;
```

For further information, see section Macros in GAMS below.

$(on|off)ExternalInput ($offExternalInput)

Syntax:

```
$onExternalInput
$offExternalInput
```

`$onExternalInput` and `$offExternalInput` mark the beginning and end of a well defined section in which data is loaded implicitly from an external data source, namely from a GDX file specified by the command line parameter IDCGDXInput.

Note

- If none of the command line parameters IDCGDXInput, IDCGenerateGDXInput, IDCJSON, IDCGenerateJSON, or IDCGenerateGDX is set, nothing - but a few checks for consistency - will happen by setting `$onExternalInput` and `$offExternalInput`.
- If at least one of the before mentioned command line parameters is set, an additional license check will be triggered: Without the `MIRO Connector License` not more than 10 indexed symbols can be declared as external input or external output.

For symbols, which are declared, while `$onExternalInput` is active, data in data statements of the declaration is ignored. Instead, the data is loaded from the GDX file specified by IDCGDXInput. Also for those symbols, if they are referenced with $load* or $offEmbeddedCode while `$onExternalInput` is active, the data is loaded from the GDX file specified by IDCGDXInput instead of the defined data source and symbol renaming as well as the projection operator are ignored in this case.

In addition, the following dollar control options are ignored, while `$onExternalInput` is active and IDCGDXInput is set:

- $batInclude
- $call
- $call.Async[NC]
- $call.checkErrorLevel
- $callTool
- $callTool.checkErrorLevel
- $gdxIn
- $hiddenCall
- $hiddenCall.Async[NC]
- $hiddenCall.checkErrorLevel
- $hiddenCallTool
- $hiddenCallTool.checkErrorLevel
- $include
- $libInclude
- $onEmbeddedCode[S|V] (including the following embedded code until `$offEmbeddedCode` is hit)
- $sysInclude

Example:

Lets look at a modified version of the simple example presented here:

```
Set
   i 'canning plants'
   j 'markets';

$onExternalInput
$call mdb2gms I=Sample.accdb Q="SELECT city1, city2, distance FROM distances" O=distances.inc
Parameter d(i<,j<) 'distance in thousands of miles' /
$include distances.inc
/;
$offExternalInput

display d;
```

Without setting a command line parameter, this runs in the same way as described in the other chapter, so by default, setting `$onExternalInput` and `$offExternalInput` does not make much of a difference. Now, lets run it with IDCGDXInput being set to `transsol`, which is a GDX file generated by running the transportation model [**TRNSPORT**]. There will be some new messages in the log:

```
--- GDXin=transsol.gdx
--- Distances1.gms(7) 2 Mb
--- Ignoring $call in externalInput section
--- Distances1.gms(9) 2 Mb
--- Ignoring $include in externalInput section
```

First, we notice, that the file `transsol.gdx` will be opened for input. This happens implicitly with `$onExternalInput`. Then, both the `$call` as well as the `$include` lines were ignored, since we do not want to query the data source explicitly defined in the model, but the GDX file defined by IDCGDXInput. Note, that the data statement in the declaration of `d` is actually empty when we ignore the `$include` statement. While `$onExternalInput` this does not lead to an error, but behaves as if $onEmpty was set. Looking at the output of the `display` statement in the `lst` file, one can see that the expected data was loaded:

```
----     13 PARAMETER d  distance in thousands of miles

             new-york      chicago       topeka

seattle         2.500        1.700        1.800
san-diego       2.500        1.800        1.400
```

If any of the options IDCGDXInput, IDCGDXOutput, IDCJSON or their `Generate` versions, e.g. IDCGenerateGDXInput, has been set, a MIRO connector license is required.

See also $(on|off)ExternalOutput.

$(on|off)ExternalOutput ($offExternalOutput)

Syntax:

```
$onExternalOutput
$offExternalOutput
```

`$onExternalOutput` and `$offExternalOutput` mark the beginning and end of a well defined section in which data is written implicitly to an external data file, namely to a GDX file specified by the command line parameter IDCGDXOutput or IDCGenerateGDXOutput.

Note

- If none of the command line parameters IDCGDXOutput, IDCGenerateGDXOutput, IDCJSON, IDCGenerateJSON, or IDCGenerateGDX is set, nothing - but a few checks for consistency - will happen by setting `$onExternalOutput` and `$offExternalOutput`.
- If at least one of the before mentioned command line parameters is set, an additional license check will be triggered: Without the `MIRO Connector License` not more than 10 indexed symbols can be declared as external output or external input.

Example:

This is a slightly modified code snippet of the transportation model [**TRNSPORT**]:

```
...
$onExternalOutput
Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

Variable
   x(i,j) 'shipment quantities in cases'
```

```
$offExternalOutput
   z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
$onExternalOutput
   cost      'define objective function'
$offExternalOutput
   supply(i) 'observe supply limit at plant i'
   demand(j) 'satisfy demand at market j';
...
```

Without setting a command line parameter, this runs in the same way as the original model, so by default, setting `$onExternalOutput` and `$offExternalOutput` does not make much of a difference. Now, lets run it with IDCGDXOutput being set to `transOut.gdx`. This generate the file `transOut.gdx` with the following content:

Entry	Name	Type	Dim	Records	Text
1	c	Parameter	2	6	transport cost in ...
3	cost	Equation	0	1	define objective ...
2	x	Variable	2	6	shipment quanti...

i	j	Level	Marginal	Lower	Upper	Scale
seattle	new-york	50	0	0	+INF	1
seattle	chicago	300	0	0	+INF	1
seattle	topeka	0	0.036	0	+INF	1
san-diego	new-york	275	0	0	+INF	1
san-diego	chicago	0	0.009	0	+INF	1
san-diego	topeka	275	0	0	+INF	1

So, this implicit generation of the output file is basically equivalent to the following explicit statement:

```
execute_unload 'transOut.gdx', c, x, cost;
```

If any of the options IDCGDXInput, IDCGDXOutput, IDCJSON or their `Generate` versions, e.g. IDCGenerateGDXInput, has been set, a MIRO connector license is required.

See also $(on|off)ExternalInput.

$(on|off)Filtered ($onFiltered)

Syntax:

```
$onFiltered
$offFiltered
```

This dollar control option controls how GAMS loads data from GDX with gdxLoad, $load, $loadR, and $loadM. With `$onFiltered` the `$load*` calls behave like $loadFiltered* calls. With `$offFiltered` the `$load*` calls behave like $loadDC* calls. This dollar control option also controls how GAMS loads symbols on the $offEmbeddedCode line. With `$onFiltered` data is read filtered while `$offFiltered` triggers a domain check of the data. The gams.set Python function allows to overwrite the setting of this dollar control option with the argument `domCheck=DomainCheckType.CHECKED` or `domCheck=DomainCheckType.FILTERED`.

The default of how GAMS loads data from GDX can be set with the command line parameter filtered.

$(on|off)Global ($offGlobal)

Syntax:

```
$onGlobal
$offGlobal
```

When an include file is inserted, it inherits the dollar control options from the higher level file. However, the dollar control option settings specified in the include file do not affect the higher level file. This convention is common among most scripting languages or command processing shells. In some cases, it may be desirable to break this convention. This option allows an include file to change the options of the parent file as well.

Example:

```
$include 'inc.inc'
$hidden after first call to include file
$onGlobal
$include 'inc.inc'
$hidden after second call to include file
```

The file `inc.inc` contains the following lines:

```
$onDollar
$hidden text inside include file
```

The the echo print of the resulting listing file follows:

```
INCLUDE    D:\GAMS\INC.INC
    2  $onDollar
    3  $hidden text inside include file
INCLUDE    D:\GAMS\INC.INC
    7  $onDollar
    8  $hidden text inside include file
    9  $hidden after second call to include file
```

Note that the dollar control option $onDollar inside the include file does not affect the parent file until `$onGlobal` is set. The text following the option $hidden is then echoed to the listing file.

$(on|off)IDCProtect ($onIDCProtect)

Syntax:

```
$onIDCProtect
$offIDCProtect
```

This controls if symbols which are declared as external input can be changed at execution time. With `$onIDCProtect` external input symbols are protected from being changed; `$offIDCProtect` allows it.

The initial state of this can be set with the command line parameter IDCProtect.

$(on|off)ImplicitAssign ($offImplicitAssign)

Syntax:

```
$onImplicitAssign
$offImplicitAssign
```

This controls if the compiler issues or ignores the 141 error in case an unassigned symbol in the code is referenced (e.g. on the right hand side of an assignment statement). With `$onImplicitAssign` the error is ignored.

The initial state of this can be set with the command line parameter ImplicitAssign.

$(on|off)Include ($onInclude)

Syntax:

```
$onInclude
$offInclude
```

This option controls the listing of the expanded include file name in the listing file.

Example:

```
$include 'inc.inc'
$offInclude
$include 'inc.inc'
```

We assume that the file `inc.inc` contains the following lines:

```
$onDollar
$hidden Text inside include file
```

The resulting listing file will contain the following lines:

```
INCLUDE    C:\tmp\inc.inc
   2  $onDollar
   3  $hidden Text inside include file
   6  $onDollar
   7  $hidden Text inside include file
```

Note that the include file name is echoed the first time the include file is used. However, the include file name is not echoed after `$offInclude` has been set. Also, the include file summary and the creation of an Expanded Include File is suppressed if `$offInclude` is set.

$(on|off)Inline ($offInline)

Syntax:

```
$onInline
$offInline
```

This option acts as switch to control the use of in-line comments. Note that by default, the in-line comment symbols are set to the two character pairs /* and */ but the processing is disabled. In-line comments may span several lines till the end-of-comment characters are encountered.

Example:

```
$onInline
Set i /* The default comment symbols are now
         active. These comments can continue
         to additional lines till the closing
         comments are found.  */ / i1*i3 / ;
```

Note

- The option $inlineCom automatically sets `$onInline`.
- *Nested* in-line comments are illegal unless the option $onNestCom is set.

The default for dealing with inline comments can be set using the command line parameter inlineCom.

See also section Comments.

$(on|off)Listing ($onListing)

Syntax:

```
$onListing
$offListing
```

This option controls the echoing of input lines to the compilation output of the listing file. Note that suppressed input lines do not generate entries in the symbol and reference sections that appear at the end of the compilation output. Lines with errors will always be listed.

Example:

```
Set i /0234*0237/
    j /a,b,c/     ;
Table x(i,j) "very long table"
        a   b   c
 0234   1   2   3
$offListing
 0235   4   5   6
 0236   5   6   7
$onListing
 0237   1   1   1
;
```

The resulting listing file will contain the following lines:

```
    1  Set i /0234*0237/
    2      j /a,b,c/     ;
    3  Table x(i,j) very long table
    4          a   b   c
    5   0234   1   2   3
   10   0237   1   1   1
```

Note that the lines in the source file between the options `$offListing` and `$onListing` are not echoed to the listing file.

Note

For some projects the listing file can become huge and can take significant time to be written. This time can be saved by setting `$offListing` at the beginning of the input file and `$onListing` just before the parts one is interested in, or not at all, if one does not look at the listing file anyway.

The default for dealing with the compilation output of the listing file can be set using the command line parameter listing.

$(on|off)Local ($onLocal)

Syntax:

```
$onLocal
$offLocal
```

The suffix `.local` attached to the name of a controlling set will use an *implicit alias* within the scope of the indexed operation or on the left-hand side of an assignment statement. This feature is particularly useful in the context of nested macros.

Example:

```
Set i /1*3/; alias(i,j);
Parameter xxx(i,j) / 1.1 1, 2.2 2, 3.3 3, 1.3 13, 3.1 31 /;
display xxx;

Parameter p(i);
p(i.local) = sum(j, xxx(i,j));
display p;
```

Note that in the assignment statement the set `i` on the right-hand side is controlled by `i.local` on the left-hand side. Thus we have the following values for the two parameters:

```
----      3 PARAMETER xxx
            1           2           3
1       1.000                  13.000
2                   2.000
3      31.000                   3.000

----      7 PARAMETER p
1 14.000,    2  2.000,    3 34.000
```

In the example above, the suffix `.local` appeared one time on the left-hand side. The option `$onLocal` allows the suffix `.local` to appear more that one time attached to the same symbol. Consider the following example that extends the example above:

```
Parameter g(i,i);
g(i.local-1,i.local) = xxx(i,i);
display g;
```

Note that in the assignment statement of `g` the suffix `.local` attached to the set `i` appears two times on the left-hand side. The question arises whether the reference to the set `i` on the right-hand side refers to the first or the second instance of `.local` on the left-hand side. The assignment statement may alternatively be written in the following way using an *explicit* alias statement:

```
alias (i,i1,i2);
g(i1-1,i2) = xxx(i2,i2);
```

Thus is becomes clear that the symbol on the right-hand side refers to the controlling index that enters last (here the second one). The output generated by the display statement follows:

```
----      10 PARAMETER g
            1           2           3
1       1.000       2.000       3.000
2       1.000       2.000       3.000
```

Observe that the multiple use of the suffix `.local` on the same symbol is considered an error with the option `$offLocal`.

Note that it is also allowed to combine the original index with an index suffixed with `.local`. Consider the following alternative formulation:

```
g(i.local-1,i) = xxx(i,i);
```

Note that in this case the index suffixed with `.local` takes precedence and the reference of `i` on the right-hand side refers to the index `i.local` even though `i` is entered last. Observe that this statement even works with `$offLocal` as the suffix `.local` appears only once.

See also section Macros in GAMS below.

$(on|off)Log ($onLog)

Syntax:

```
$onLog
$offLog
```

This option acts as a switch that controls logging information about the line number and memory consumption during compilation. This is scoped like the option $on/offListing applying only to included files and any subsequent included files but reverting to the setting `$onLog` in the parent files (if it was not changed there as well).

Example:

```
Set i /i1*i20000000/;
$include inc.inc
Set l /l1*l20000000/;
```

The file `inc.inc` looks like this:

```
Set j /j1*j20000000/;
$offLog
Set k /k1*k20000000/;
```

The generated log will contain the following lines:

```
--- test.gms(1) 1602 Mb  5 secs
--- test.gms(2) 1602 Mb
--- .inc.inc(1) 3122 Mb  6 secs
--- test.gms(3) 6161 Mb  14 secs
```

Note that the first line of both the parent and the include file got logged, but not the third line of the include file, after `$offLog` was set. The last line of the parent file got logged again.

$(on|off)Macro ($onMacro)

Syntax:

```
$onMacro
$offMacro
```

Enables or disables the expansion of macros defined by $macro.

Example:

```
$macro oneoverit(y) 1/y
$offMacro
y = oneoverit(x1);
display y;
```

causes an error because the macro `oneoverit` in line 3 can not be expanded.

$(on|off)Margin ($offMargin)

Syntax:

```
$onMargin
$offMargin
```

This option controls margin marking, that means if margins set by the options $minCol and $maxCol, should be marked in the lst file.

Example:

```
$onmargin mincol 20 maxcol 51
Now we have        Set       i "plant" / US, UK /; This defines I
turned on the      Scalar    x         / 3.145  /; A scalar example.
margin marking.    Parameter a, b;                  Define some
                                                    parameters.
$offmargin
```

The lst file will contain this:

```
  2 Now we have        |Set       i "plant" / US, UK /; |This defines I
  3 turned on the      |Scalar    x         / 3.145  /; |A scalar example.
  4 margin marking.    |Parameter a, b;                 |Define some
  5                    |                                |parameters.
```

Note that any statements between columns 1 and 19 and any input beyond column 52 are treated as comments. These margins are marked with | on the left and right.

See also section Comments.

$(on|off)Multi ($offMulti)

Syntax:

```
$onMulti
$offMulti
```

This option controls multiple data statements or tables. By default, GAMS does not allow data statements to be redefined. If this option is activated the second or subsequent data statements are merged with entries of the previous ones. Note that all multiple data statements are performed before any other statement is executed.

Note

- There is also $onMultiR, which behaves similarly, but replaces existing data instead of merging into it. Compare the example here and there to see the difference.
- When `$onMulti` is active $load behaves like a $loadM.
- The initial state of this dollar control option can be set through the command line parameter multi.

Example:

Consider the following slice of code. The list after the end of line comment describes the complete content of the symbol x after the data statement has been processed:

```
$eolCom //
Set i / i1*i10 /;
Parameter x(i) / i1*i3  1 /   // /i1 1,i2 1,i3 1/
$onMulti
Parameter x(i) / i7*i9  2 /   // /i1 1,i2 1,i3 1,i7 2,i8 2,i9 2/
Parameter x(i) / i2*i6  3 /   // /i1 1,i2 3,i3 3,i4 3,i5 3,i6 3,i7 2,i8 2,i9 2/
Parameter x(i) / i3*i5  0 /   // /i1 1,i2 3,i6 3,i7 2,i8 2,i9 2/
$offMulti
display x;
```

Note that the repeated parameter statements would have resulted in a compilation error without the presence of the option `$onMulti`. The result of the display statement in the listing file follows:

```
----      8 PARAMETER x

1 1.000,    2 3.000,    6 3.000,    7 2.000,    8 2.000,    9 2.000
```

Note that `x("i1")` is assigned the value of 1 with the first data and is not affected by any of the subsequent data statements. `x("i3")` on the other hand is reset to 3 by the third data statement and wiped out with 0 in the fourth data statement.

Attention

The two-pass processing of a GAMS file may lead to seemingly unexpected results. Dollar control options and data initialization are both done in the first pass and assignments in the second, irrespective of their relative locations. This is an issue particularly with the option `$onMulti` since it allows data initializations to be performed more than once. See section GAMS Compile Time and Execution Time Phase for details.

Consider the following example:

```
Scalar a /12/;
a=a+1;
$onMulti
Scalar a /20/;
display a;
```

Note that the two `scalar` data initialization statements and the option `$onMulti` are processed before the assignment statement `a=a+1`. As a result, the final value of `a` will be 21. The output of the display statement follows:

```
----      5 PARAMETER a                   =       21.000
```

Observe that the option $onEmpty in conjunction with the option `$onMulti` and the save and restart feature may be used to set up a model and add data later. See example in section Advanced Separation of Model and Data for details.

$onMultiR|offMulti ($offMulti)

Syntax:

```
$onMultiR
$offMulti
```

This option controls multiple data statements or tables. By default, GAMS does not allow data statements to be redefined. If this option is activated the second or subsequent data statements replace the previous ones. Note that all multiple data statements are performed before any other statement is executed.

Note

- There is also $onMulti, which behaves similarly, but *merges* into existing data instead of replacing it. Compare the example here and there to see the difference.
- In contrast to $onMulti, `$onMultiR` also allows to redefine an equation and a macro.
- The initial state of this dollar control option can be set through the command line parameter multi.

Example:

Consider the following slice of code. The list after the end of line comment describes the complete content of the symbol x after the data statement has been processed:

```
$eolCom //
Set i / i1*i10 /;
Parameter x(i) / i1*i3  1 /   // /i1 1,i2 1,i3 1/
$onMultiR
Parameter x(i) / i7*i9  2 /   // /i7 2,i8 2,i9 2/
Parameter x(i) / i2*i6  3 /   // /i2 3,i3 3,i4 3,i5 3,i6 3/
Parameter x(i) / i3*i5  0 /   // ( ALL       0.000 )
$offMulti
display x;
```

Note that the repeated parameter statements would have resulted in a compilation error without the presence of the option `$onMultiR`. The result of the display statement in the listing file follows:

```
----      9 PARAMETER x

( ALL       0.000 )
```

Note

- When using a redefinition with `$onMulitR` to remove elements from a set, which was used as domain of another symbol, also the dependent symbol gets reduced. This can happen through data statements, but also with $clear and $load/$loadDC.
- When `$onMultiR` is active $load behaves like a $loadR with the exception of the previous note. So, the only way to allow for a redefiniton of a domain set is to activate `$onMultiR`. If that is not active `$loadR` for a domain set would trigger an error.

Example:

```
Set i          / i1*i5 /;
Parameter p(i) / #i 3  /;

$onMultiR

Set i          / i3*i6 /;
Display p;
```

Looking at the output from the `Display` statement one can see, that the entries for `i1` and `i2` were removed by the second definition of `i`:

```
----      7 PARAMETER p

i3 3.000,    i4 3.000,    i5 3.000
```

Attention

> The two-pass processing of a GAMS file may lead to seemingly unexpected results. Dollar control options and data initialization are both done in the first pass and assignments in the second, irrespective of their relative locations. This is an issue particularly with the option `$onMultiR` since it allows data initializations to be performed more than once. See section GAMS Compile Time and Execution Time Phase for details.

Consider the following example:

```
Scalar a /12/;
a=a+1;
$onMultiR
Scalar a /20/;
display a;
```

Note that the two `scalar` data initialization statements and the option `$onMultiR` are processed before the assignment statement `a=a+1`. As a result, the final value of `a` will be 21. The output of the display statement follows:

```
----      5 PARAMETER a                  =       21.000
```

Observe that the option $onEmpty in conjunction with the option `$onMultiR` and the save and restart feature may be used to set up a model and add data later. See example in section Advanced Separation of Model and Data for details.

$(on|off)NestCom ($offNestCom)

Syntax:

```
$onNestCom
$offNestCom
```

This option controls nested in-line comments. It makes sure that the open-comment and close-comment characters match.

Example:

```
$inlineCom { } onNestCom
  { nesting is now possible in comments { braces
    have to match } }
```

See also $inlineCom, $onInline and section Comments.

$(on|off)Order ($onOrder)

Syntax:

```
$onOrder
$offOrder
```

Lag and lead operations and the ord operator require the referenced set to be ordered and constant. In some special cases users might want to use those operations on dynamic and/or unordered sets. The option `$on/offOrder` has been added to locally relax the default requirements. The use of this option comes with a price, the system will not be able to diagnose odd and incorrect formulations and data sets.

Example:

```
Set       t1    / 1987, 1988, 1989, 1990, 1991 /
          t2    / 1983, 1984, 1985, 1986, 1987 /;
Parameter p(t2);
$offOrder
p(t2) = ord(t2);
display t2,p;
```

Without the `$offOrder` the compilation of the line `p(t2) = ord(t2);` would have triggered a compilation error. The ordinal numbers assigned here are probably not what one expects. The element `1987` gets ordinal number 1 although it seems to be last last in the set. The ordinal numbers are assigned in the order the set is stored internally in GAMS. This order is also used when displaying the set `t2`:

```
----      6 SET t2
1987,    1983,    1984,    1985,    1986

----      6 PARAMETER p
1987 1.000,    1983 2.000,    1984 3.000,    1985 4.000,    1986 5.000
```

$(on|off)Put[S|V][.tag]

Syntax:

```
File myputfile;
put myputfile;
$onPut[S|V][.tag]
text
{text}
$offPut[.tag]
```

The pair `$onPut[S|V]` - `$offPut` causes a block of text to be placed in a put file at run-time. The is one of the few dollar control options that operate at run time. The `$` in the first column usually indicates action at compile time.

Note that parameter substitutions are not permitted with `$onPut`. The option `$onPut` has two more variants: `$onPutS` and `$onPutV`. `$onPutS` allows parameter substitutions while the option `$onPutV` does not allow parameter substitutions, like `$onPut`, so it is just a synonym which makes it more obvious that the text is written verbatim with the appended `V`.

Example:

```
$set it TEST
File myputfile;
put myputfile;
$onPutS
Line 1 of text "%it%"
Line 2 of text %it%
$offPut
```

This code generates the put file `myputfile.put` with the following content:

```
Line 1 of text "TEST"
Line 2 of text TEST
```

Note that the compile-time variable `%it%` was replaced by `TEST`. However, if the option `$onPutV` is used instead, then `%it%` will not be substituted:

```
$set it TEST
File myputfile
put myputfile
$onPutV
Line 1 of text "%it%"
Line 2 of text %it%
$offPut
```

The resulting file `myputfile.put` will contain the following lines:

```
Line 1 of text "%it%"
Line 2 of text %it%
```

One may add a `tag` to the `$onPut[S|V]` and `$offPut`. This allows to use nested put blocks like this:

```
File myputfile / outer.gms /;
put myputfile;
$onPut.outer
File myputfile;
put myputfile;
$onPut.inner
Line 1 of text
Line 2 of text
$offPut.inner
$offPut.outer
```

The resulting file `outer.gms` looks like this:

```
File myputfile;
put myputfile;
$onPut.inner
Line 1 of text
Line 2 of text
$offPut.inner
```

$(on|off)Upper ($offUpper)

Syntax:

```
$onUpper
$offUpper
```

GAMS code echoed to the listing file is written in upper case after `$onUpper`. The default of mixed code echoing is restored with `$offUpper`.

Note

The default case of the echo print can be set using the command line option case.

Example:

```
Set
   i 'Canning plants' / seattle,  san-diego /
   j 'Markets'        / new-york, chicago, topeka /;

$onupper
Parameter
   a(i) 'Capacity of plant i in cases'
        / seattle    350
          san-diego  600 /

$offupper
   b(j) 'Demand at market j in cases'
        / new-york   325
          chicago    300
          topeka     275 /;
```

The generated listing file looks like this:

```
   1  Set
   2     i 'Canning plants' / seattle,  san-diego /
   3     j 'Markets'        / new-york, chicago, topeka /;
   4
   6  PARAMETER
   7     A(I) 'CAPACITY OF PLANT I IN CASES'
   8          / SEATTLE    350
   9            SAN-DIEGO  600 /
  10
  12     b(j) 'Demand at market j in cases'
  13          / new-york   325
  14            chicago    300
  15            topeka     275 /;
```

$(on|off)Recurse ($offRecurse)

Syntax:

```
$onRecurse
$offRecurse
```

This option controls whether it is permitted for a file to include itself.

Example:

The following GAMS program result in a recursive inclusion of the program itself:

```
$onRecurse
$include "%gams.input%"
```

Note that the maximum include nesting level is 40 and if it is exceeded an error is triggered.

In the following example that prints a string and then the reverse string the nesting level is less that 40 and one get some kind of recursion at compile time:

```
$onEchoV > reverse.gms
$ifthene %1=%3+1
  put ' '
$ exit
$endif
loop(map(chars,code)$(code.val=ord("%2",%1)), put chars.tl:0);
$eval posPlus1 %1+1
$batInclude reverse %posPlus1% %2 %3
loop(map(chars,code)$(code.val=ord("%2",%1)), put chars.tl:0);
$offEcho

set chars / A*Z /, code / 65*90 /, map(chars,code) / #chars:#code /;
file fx /''/; put fx;
$onRecurse
$batInclude reverse 1 RACECAR 7
put /;
$batInclude reverse 1 LAGER 5
```

The log will print the following lines:

```
--- Starting execution: elapsed 0:00:00.067
RACECAR RACECAR
LAGER REGAL
*** Status: Normal completion
```

$(on|off)StrictSingleton ($onStrictSingleton)

Syntax:

```
$onStrictSingleton
$offStrictSingleton
```

If the option `$onStrictSingleton` is active, a compilation error is triggered if a data statement for a singleton set contains more than one element. After activating the option `$offStrictSingleton` GAMS will take the *first* element of a singleton set that was declared with multiple elements as the valid element, the other elements are disregarded and there is no error. The option to change the initial state of `$(on|off)StrictSingleton` and/or to control this behavior at runtime is strictSingleton.

Example:

The *first* element is not always the one that appears in the data statement first as the following example shows:

```
set i /1,2/
$offStrictSingleton
singleton set ii(i) /2,1/;
display ii;
```

The set `ii` contains the element `1` because it is the first in the GAMS label order as the display statements shows:

```
----      4 SET ii
1
```

$(on|off)SuffixDLVars ($offSuffixDLVars)

Syntax:

```
$onSuffixDLVars
$offSuffixDLVars
```

This option controls if it is allowed to use suffixes on variables with limited domains in a model. By default this is not allowed but this can be changed by setting `$onSuffixDLVars`.

Example:

```
Set n        / n1*n100 /
    t        / t1*t10  /
    sub(n,t) / #n:#t   /;

Variable z, x(n,t);
Equation obj;

x.up(n,t) = uniformInt(7,13);
obj.. z =e= sum((n,t),x(n,t)/x.up(n,t));

Model m / all, x(sub) /;
Solve m min z use lp;
```

This causes the following error by default:

```
*** Problem in Model Generation: Variable "x" appears with suffix "Up".
*** Error at line 12: Cannot use domain limited variables with suffix in model - use $onSuffixDL
```

The purpose is to make the user aware of the fact, the she uses `x.l`, which would also be affected by the domain restriction of `x(sub)` in the model statement. So, if the user made sure, that this is all intended, setting `$onSuffixDLVars` before the solve statements overcomes this error. Though, with the example given here, this would actually lead to a "division by zero" error, since some `x.up` were excluded leaving a `0` as divisor.

The default for dealing with suffixes on variables with limited domains can be set using the command line parameter SuffixDLVars.

$(on|off)SuffixAlgebraVars ($onSuffixAlgebraVars)

Syntax:

```
$onSuffixAlgebraVars
$offSuffixAlgebraVars
```

This option controls if it is allowed to use suffixes in equations. By default this is allowed but it can be changed by setting `$offSuffixAlgebraVars`. Activating `$offSuffixAlgebraVars` can be useful to ensure, that no decision variable in a model was replaced by one of its attributes, for example if one is dealing with a model written by someone else.

Example:

```
Variable x / 1 0 /, z / 1 0 /;
Equation e;

e.. z =e= sqr(1-x.l);

Model m /e/;
solve m min z use nlp;
```

This runs, but might give a different solution than expected.

Activating `$offSuffixAlgebraVars` will point to a potential problem:

```
   5  e.. z =e= sqr(1-x.l);
****                     $917
**** 917  Cannot use variable with suffix in model algebra -
****         Use $onSuffixAlgebraVars to allow this
```

See also related command line parameter SuffixAlgebraVars.

$(on|off)SymList ($offSymList)

Syntax:

```
$onSymList
$offSymList
```

This option controls whether the symbol listing map appears in the compilation output of the listing file. The symbol listing map contains the complete listing of all symbols that have been defined and their explanatory text. The entries are in alphabetical order and grouped by symbol type.

Example:

The symbol listing map generated by running [**TRNSPORT**] with `$onSymList` is as follows:

```
Symbol Listing


SETS

i             canning plants
j             markets

PARAMETERS

a             capacity of plant i in cases
b             demand at market j in cases
c             transport cost in thousands of dollars per case
d             distance in thousands of miles
f             freight in dollars per case per thousand miles

VARIABLES

x             shipment quantities in cases
z             total transportation costs in thousands of dollars

EQUATIONS

cost          define objective function
demand        satisfy demand at market j
supply        observe supply limit at plant i

MODELS

transport
```

This serves as a simple description of the symbols used in a model and may be used in reports and other documentation. For further information, see section The Symbol Listing Map.

$(on|off)SymXRef ($offSymXRef)

Syntax:

```
$onSymXRef
$offSymXRef
```

This option controls the following:

- Collection of cross references for symbols like sets, parameters, variables, acronyms, equations, models and put files.
- Symbol cross reference report of all collected symbols in the compilation output of the listing file. For details, see section The Symbol Reference Map.
- Listing of all referenced symbols and their explanatory text by symbol type in listing file. This listing may also be activated with the option $onSymList.

Example:

```
$onSymXRef
Set i / 1*6 /, k;
$offSymXRef
Set j(i) "will not show" / 1*3 /;
$onSymXRef
k('1') = yes;
```

The resulting listing file will contain the following symbol reference map and symbol listing map:

```
SYMBOL      TYPE   REFERENCES

i            SET   declared        2  defined        2
k            SET   declared        2 assigned        6

SETS

i
k
```

Note that the set `j` does not appear in these listings because the listing was deactivated with the option `$offSymXRef` in line 3 of the code above.

$(on|off)Text

Syntax:

```
$onText
$offText
```

The pair `$onText` - `$offText` encloses comment lines. Line numbers in the compiler listing are suppressed to mark skipped lines.

Example:

```
* Standard comment line
$onText
Everything here is a comment
until we encounter the closing $offText
like the one below
$offText
* Another standard comment line
```

The echo print of the resulting listing file will contain the following lines:

```
1  * Standard comment line
   Everything here is a comment
   until we encounter the closing $offText
   like the one below
7  * Another standard comment line
```

Attention

GAMS requires that every `$onText` has a matching `$offText` and vice versa.

See also section Comments.

$(on|off)Troll ($offTroll)

Syntax:

```
$onTroll
$offTroll
```

This option controls whether to recognize Troll periodicity in set definitions using sequences (`id*id`). Leap years are taken into account. Supported formats are `yyyyA` (annual), `yyyyQq` (quartely), `yyyyMmm` (monthly) and `yyyyWww` (weekly).

Example:

```
$onTroll
Set
  trollw / 1990w49*1991w4 /
  trollm / 1990m10*1991m5 /
  trollq / 1990q1*1991q4 /
;

Display trollw, trollm, trollq;
```

The resulting listing file will contain the following output:

```
----     25 SET trollw

1990W49,    1990W50,    1990W51,    1990W52,    1991W1 ,    1991W2 ,    1991W3 ,    1991W4


----     25 SET trollm

1990M10,    1990M11,    1990M12,    1991M1 ,    1991M2 ,    1991M3 ,    1991M4 ,    1991M5


----     25 SET trollq

1990Q1,    1990Q2,    1990Q3,    1990Q4,    1991Q1,    1991Q2,    1991Q3,    1991Q4
```

$(on|off)UElList ($offUElList)

Syntax:

```
$onUElList
$offUElList
```

This option controls the complete listing of all set elements that have been entered in the compilation output of the listing file. For details see section The Unique Element Listing Map.

Example:

The unique element listing in the listing file generated by running the model [**TRNSPORT**] with `$onUElList` follows:

```
Unique Element Listing


Unique Elements in Entry Order

1  seattle     san-diego   new-york    chicago     topeka

Unique Elements in Sorted Order

1  chicago     new-york    san-diego   seattle     topeka
```

Note that the sorted order is not the same as the entry order. For more information, see section Ordered and Unordered Sets.

$(on|off)UElXRef ($offUElXRef)

Syntax:

```
$onUElXRef
$offUElXRef
```

This option controls the collection and listing of cross references of set elements in the compilation output. For more information, see section The Unique Element Listing Map.

Example:

```
Set i "set declaration" / one, two, three /, k(i);
$onUElXRef
k('one') = yes;
$offUElXRef
k('two') = yes;
$onUElXRef
k('three') = yes;
```

The resulting listing file will contain the following unique element reference report:

```
Unique Element Listing


ELEMENT   REFERENCES

one           index        3
three         index        7
```

Note that the element `two` does not appear in this listing because the listing was deactivated with the option `$offUElXRef` in line 4 of the code above.

$(on|off)UNDF ($offUNDF)

Syntax:

```
$onUNDF
$offUNDF
```

This option controls the use of the special value UNDF which indicates a result is undefined. For details see section Extended Range Arithmetic. By default, UNDF is not permitted to be used in assignments. This may be changed with the option `$onUNDF`.

Example:

```
Scalar x;
$onUNDF
x = UNDF;
Display x;
```

The output of the display statement follows:

```
----      4 PARAMETER x                   =          UNDF
```

Note that an error would have been triggered without the use of `$onUNDF`. The option `$offUNDF` will return the system to the default, where `UNDF` may not be used in assignments.

$(on|off)Uni ($offUni)

Syntax:

```
$onUni
$offUni
```

This controls whether the compiler checks the referential integrity (see section Domain Checking) of the code. This is an essential part of good GAMS programming and it is highly recommend to declare symbols with proper domains. With the universe as a domain the compiler does not help the user with easy-to-make mistakes, like swapping indexes, `a(i,j)` versus `a(j,i)`. By default something like this would generate an error, if `a` was declared as `a(i,j)`. Such an error could be ignored, by setting `$onUni`, which can be useful in few situations, when accessing a symbol with a set that is not the domain or a subset of the domain. For example, we could read data of a union of sets that already exist. We could use the universe as the domain for that symbol, but perhaps we need to protect the referential integrity of this symbol too.

Example:

```
Set       fruit   / apple, pear /
          veggie  / carrot, pea /
          produce / #fruit, #veggie /;
Parameter produceCalories(produce) "per 100g" / apple 52, pear 57, carrot 41, pea 81 /
          fc(fruit)  "calories per 100g"
          vc(veggie) "calories per 100g";
$onUni
fc(fruit)  = produceCalories(fruit);
vc(veggie) = produceCalories(veggie);
$offUni
display fc, vc;
```

So when assigning `fc` we only access `produceCalories` with `fruit`. We could reverse the order of declaration of `fruit`, `veggie` and `produce` and use a proper subdomain, but sometimes data flow and input don't allow that.

Attention

When the GAMS compiler operates under $onUni it treats all symbols as being declared over the universe. So all domain checking is gone. We can set elements in a symbols that normally can't be entered. This can also lead to strange effects:

```
set       i / 1*2 /
          j / a,b /;
parameter pi(i);
$onuni
pi(j) = 1;
$offuni
* We will see elements from j in pi
Display pi;
* The following should only clear the i-elements from pi, but it clears the
* entire symbol, because GAMS knows it's doing this to the entire domain and
* takes a shortcut.
pi(i) = no;
Display pi;
```

$(on|off)Verbatim

Syntax:

```
$onVerbatim
$offVerbatim
```

These options are used in conjunction with the GAMS command line parameter DumpOpt to suppress the input preprocessing for input lines that are copied to the dmp file. This feature is mainly used to maintain different versions of related models in a central environment.

Note

- The options `$on/offVerbatim` are only recognized for `DumpOpt` $\geq$ 10 and apply only to lines in the file between the two options
- If `$onVerbatim` is active, `DumpOpt = 11` behaves like `DumpOpt = 21` (comments are kept)

Observe that the use of the options $goto and `$on/offVerbatim` are incompatible and may produce unexpected results.

Example:

```
$set f 123
$log %f%
$onVerbatim
$log %f%
$offVerbatim
$log %f%
```

The corresponding dmp file will contain the following lines:

```
$log 123
$onVerbatim
$log %f%
$offVerbatim
$log 123
```

See also command line parameter DumpOpt.

$(on|off)Warning ($offWarning)

Syntax:

```
$onWarning
$offWarning
```

This option acts as a switch for data domain checking. In some cases it may be useful to accept domain errors in data statements that are imported from other systems and report warnings instead of errors. Data will be accepted and stored, even though it is outside the domain.

Attention

- This switch effects three types of domain errors usually referred to as error numbers 116, 170 and 171, see example below.
- This may have serious side affects and we recommend to exercise great care when using this feature.

Example:

```
Set i     / one, two, three /
$onWarning
    j(i)  / four, five /
    k     / zero /;
Parameter x(i) "Messed up Data" / one 1.0, five 2.0 /;
x('six') = 6;
x(j) = 10;
x('two') = x('seven');
j(k) = yes;
$offWarning
display i,j,x;
```

Note that the set `j`, although specified as a subset of `i`, contains elements not belonging to its domain. Similarly, the parameter `x` contains data elements outside the domain of `i`. The skeleton listing file that results from running this code follows:

```
 1  Set i     / one, two, three /;
   3      j(i)  / four, five /
****                 $170  $170
   4      k     / zero /;
   5  Parameter x(i) "Messed up Data" / one 1.0, five 2.0 /;
****                                          $170
   6  x('six') = 6; x(j) = 10; x('two') = x('seven');
****       $170                               $116,170
   7  j(k) = yes;
****     $171
   9  display i,j,x;

 Error Messages

116  Label is unknown
170  Domain violation for element
171  Domain violation for set

**** 0 ERROR(S)   7 WARNING(S)

E x e c u t i o n
```

```
----      9 SET i
one   ,    two   ,    three


----      9 SET j
four,    five,    zero


----      9 PARAMETER x  Messed up Data
one   1.000,    four 10.000,    five 10.000,    six   6.000
```

Observe that the domain violations are marked like normal compilation errors but are only treated as warnings and it is permitted to execute the code.

For an introduction to domain checking in GAMS, see section Domain Checking.

$phantom id

Syntax:

```
$phantom id
```

This option is used to designate `id` as a *phantom* set element. Syntactically, a phantom element is handled like any other set element. Semantically, however, it is handled like it does not exist. This is sometimes used to specify a data template that initializes the phantom records to default values.

Example:

```
$phantom null
Set i / null /
    j / a, b, null /;
display i,j;
```

The output generated by the display statement is shown below:

```
----      4 SET i
                                                    ( EMPTY )

----      4 SET j
a,    b
```

Note that `null` does not appear in the listing file.

Attention

Statements that assign values to phantom labels are ignored.

Consider the following extension to the previous example:

```
Parameter p(j) / a 1, null 23 /;
display p;
```

The output generated by the display statement is shown below:

```
----      6 PARAMETER p
a 1.000
```

The system attribute system.empty is an implicitly defined phantom element. The following code works even without specifying $phantom:

```
Set i / system.empty /
    j / a, b, system.empty /;
display i,j;
```

Another way to specify empty data statements makes use of on/offEmpty. The following example produces the same data as the data statement with the phantom label. In contrast to the example we $phantom we need to provide the dimensionality of the symbol i explicitly via the (*):

```
$onEmpty
Set i(*) / /
    j / a, b /;
display i,j;
```

$prefixPath

Syntax:

```
$prefixPath directoryPath
```

This option augments the search path in PATH environment variable. The effect is that the text directoryPath is added to the beginning of the search path.

Example:

```
$log %sysenv.PATH%
$prefixPath C:\somewhereelse\anotherpath
$log %sysenv.PATH%
```

The log contains the following two relevant lines:

```
C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1
C:\somewhereelse\anotherpath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\
```

The option setEnv and %sysEnv.VARNAME% allow to modify system environment variables but the length of the environment variable value is limited in GAMS to 255 characters. The PATH environment variable is often much longer and therefore this special $prefixPath option exists.

This works on all platforms but the path separator depends on the operating system (; for Windows and : for Unix).

$protect

Syntax:

```
$protect all | ident1 ident2 ...
```

This option creates a privacy setting: it freezes all values of identifiers with the result that modifications are no longer allowed but the parameters may still be used in model calculation (for example, equation definitions). Here ident1 and ident2 are specific GAMS identifiers previously defined in the program and the keyword all denotes all identifiers.

Note that this option is mainly used in the context of secure work files. The privacy restrictions may be removed with the options $expose or $purge.

$purge

Syntax:

```
$purge all | ident1 ident2 ...
```

This option removes the identifiers and all associated data in a privacy setting. With explicit identifiers the listed identifiers are removed, and with `all` all identifiers are removed.

Note that this option is used in the context of secure work files. A special license file is needed for this feature to work, the removal only takes effect in the restart files.

$remark

Syntax:

```
$remark text
```

This option performs a parameter substitution and writes a comment `text` to the compilation output of the listing file. Note that the line numbers of the comment are suppressed.

Example:

```
$set it TEST
$remark Write %it% to the listing file
```

The resulting listing file will contain the following line:

```
Write TEST to the listing file
```

$[x]save[.keepCode]

Syntax:

```
$[x]save[.keepCode] saveFileName
```

This option creates a work file with all of the GAMS components compiled up to this point. Without the suffix `keepCode` no execution code is written, hence a restart from this work file has nothing to execute. `$xsave` writes a compressed work file, while work files created with `$save` are uncompressed.

Example:

```
set i /1*5/; parameter p(i) / #i 1 /;
p(i) = 2;
$save mywork
$onEcho > create_put.gms
file f / p.txt /; put f;
loop(i, put i.tl p(i) /);
$offEcho
$call.checkErrorLevel gams create_put.gms lo=2 restart=mywork
```

The resulting file `p.txt` will contain the following lines:

```
1                   1.00
2                   1.00
3                   1.00
4                   1.00
5                   1.00
```

Adding the suffix `.keepCode` in this example, i.e. `$save.keepCode mywork` will also write out unexecuted execution code, so a restart from this work file will execute this code. This option with suffix cannot be used *inside* a loop or if statement. This will trigger a compilation error. The example with the `.keepCode` suffix will result in the `p.txt` file containing the following lines:

```
1                    2.00
2                    2.00
3                    2.00
4                    2.00
5                    2.00
```

$scratchFileName

Syntax:

```
$scratchFileName VARNAME [fileStem]
```

This option establishes or redefines the content of a *scoped* compile-time variable that is accessible in the code where the command appears and all code included therein. Here `VARNAME` is any user chosen variable name; `fileStem` is optional and used to built the name for a file in the scratch directory with a scratch extension (unless an extension is set explicitly). If it is omitted, the system will make up a random file name.

So,

```
$scratchFileName fn abc
```

is equivalent to

```
$set fn %gams.scrDir%abc.%gams.scrExt%
```

Example:

```
* Set fileStem explicitly
$scratchFileName fn1 abc
$log %fn1%

* Omit fileStem
$scratchFileName fn2
$log %fn2%
```

The log will show:

```
C:\Data\t m p\225a\abc.dat
C:\Data\t m p\225a\__sFN__-0_313342928588555.dat
```

See also $set and section Compile-Time Variables.

$set

Syntax:

```
$set VARNAME text
```

This option establishes or redefines contents of a *scoped* compile-time variable that is accessible in the code where the command appears and all code included therein. Here `VARNAME` is any user chosen variable name; `text` is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that scoped compile-time variables may be destroyed (removed from the program) with the option $drop.

Note that in contrast to the option $eval the option `$set` does not evaluate the expression at compile time.

Note that GAMS allows scoped, local and global compile-time variables to be defined with the same name and therefore in some cases needs to prioritize. When referencing a compile-time variable via `%VARNAME%` a *local* variable hides *scoped* and *global* variables and a *scoped* variables hides the *global* variable as the following example demonstrates.

Example:

```
$setLocal myvar this is a local variable
$set myvar this is a scoped variable
$setGlobal myvar this is a global variable
$log %myvar%
$droplocal myvar
$log %myvar%
$drop myvar
$log %myvar%
```

The log will look as follows:

```
this is a local variable
this is a scoped variable
this is a global variable
```

If one wants to set a compile-time variable in an include file that is visible to the program after the `$include` one need to use `$setglobal`:

```
$onEchoV > setvar.gms
$setArgs varname varvalue
$setglobal %varname% %varvalue%
$offEcho
$batInclude setvar MYVAR one
$log %MYVAR%
```

The log will show

```
one
```

An inventory of all defined compile-time variables and their type (local, scoped, and global) is available with the option $show.

See also $setGlobal, $setLocal, and section Compile-Time Variables.

$setArgs

Syntax:

```
$setArgs id1 id2 id3 ...
```

With this option parameters that may be substituted are defined as GAMS compile-time variables. Note that `$setArgs` may only be used in external files that are included with the option $batInclude, $libInclude, and $sysInclude.

Example:

```
Scalar a /2/, b /4/, c /5/;
$batInclude test3 a b c
```

The file `test3.gms` contains the following lines:

```
Scalar x;
x = %1 + %2 * %3 ;
display x;
$setArgs aa bb cc
x = %aa% - %bb% * %cc%  ;
display x;
x = %1 + %2 * %3 ;
display x;
```

The option `$setArgs` allows the batInclude file to use the more descriptive compile-time variables `%aa%` instead of `%1`, `%bb%` instead of `%2` and `%cc%` instead of `%3`. Note that the use of `%1`, `%2` etc. is still allowed. The program listing looks as follows:

```
   1  Scalar a /2/, b /4/, c /5/;
BATINCLUDE C:\Users\default\Documents\gamside\projdir\test3.gms
   3  Scalar x;
   4  x = a + b * c ;
   5  display x;
   7  x = a - b * c  ;
   8  display x;
   9  x = a + b * c ;
  10  display x;
```

and the output generated by the display statements follows:

```
----      5 PARAMETER x                  =       22.000
----      8 PARAMETER x                  =      -18.000
----     10 PARAMETER x                  =       22.000
```

See also $set, $batInclude.

$setComps

Syntax:

```
$setComps perioddelimstring id1 id2 id3 ...
```

This option establishes or redefines compile-time variables so they contain the components of a period delimited string.

Here `perioddelimstring` is any period delimited string like the set specification of a multi-dimensional parameter, `id1` is the name of a scoped compile-time variable that will contain the name of the set element in the first position, `id2` is the name of a scoped compile-time variable that will contain the name of the set element in the second position and `id3` is the name of a scoped compile-time variable that will contain the name of the set element in the third position. The items may be recombined back into the original filename string by using `%id1%.%id2%.%id3%`.

Example:

```
$setComps period.delim.string id1 id2 id3
$log id1=%id1%
$log id2=%id2%
$log id3=%id3%
$set name %id1%.%id2%.%id3%
$log name=%name%
```

The resulting log file will contain the following lines:

```
id1=period
id2=delim
id3=string
name=period.delim.string"
```

See also $set.

$setDDList[.Cont]

Syntax:

```
$setDDList.Cont id1 id2 ...
$setDDList      id3 id4 id5 ...
```

This option causes GAMS to look for misspelled or undefined double dash GAMS parameters.

Example: Consider the following example where three double dash GAMS parameters are defined on the command line:

```
> gams mymodel.gms --one=11 --two=22 --three=33  --four=44
```

The corresponding GAMS file follows:

```
$log %one%
$log %two%
$setDDList three
$log %three%
$log %four%
```

Note that the option `$setDDList three` checks if all double dash parameters have been used so far except for `three`. An error is triggered because `four` has not been used so far, the log file will contain the following:

```
*** 1 double dash variables not referenced
    --four=44
```

`$setDDList.Cont` can be used to define a list of parameters to be checked over multiple lines. So, the following two examples do the same thing:

```
$setDDList one two three four
```

```
$setDDList.Cont one
$setDDList.Cont two three
$setDDList      four
```

Note

`$setDDList.Cont` does not do anything without a `$setDDList` statement.

See also section Double Dash Parameters.

$setEnv

Syntax:

```
$setEnv VARNAME value
```

This option defines an operating system environment variable. Here `VARNAME` is a user chosen environment variable name and `value` may contain text or a number. Note that system environment variables are destroyed (removed from the program) with the option $dropEnv or when GAMS terminates.

Example:

```
$ondollar
$set env this is very silly
$log %env%
$setenv verysilly %env%
$log %sysenv.verysilly%
$if not "%env%"=="%sysenv.verysilly%" $error "$setEnv did not work"

$dropenv verysilly
$if setenv verysilly $error should not be true
```

The following output is echoed to the log file:

```
--- Starting compilation
this is very silly
this is very silly
```

See also $dropEnv and section Environment Variables in GAMS.

$setGlobal

Syntax:

```
$setGlobal VARNAME text
```

This option establishes or redefines contents of a *global* compile-time variable that is accessible in the code where the command appears and all code included therein and all parent files. Here `VARNAME` is any user chosen variable name; `text` is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that global compile-time variables may be destroyed (removed from the program) with the option $dropGlobal.

The difference between local, scoped, and global compile-time variable is explained with the option $set.

See also $set, $setLocal, $dropGlobal and section Compile-Time Variables.

$setLocal

Syntax:

```
$setLocal VARNAME text
```

This option establishes or redefines contents of a *local* compile-time variable that is accessible only in the code module (source file) where it is defined. Here `VARNAME` is any user chosen variable name; `text` is optional and may contain any text. The text may contain spaces. The text can not be longer than 255 characters otherwise a compilation error is triggered. Observe that local compile-time variables may be destroyed (removed from the program) with the option $dropLocal.

The difference between local, scoped, and global compile-time variable is explained with the option $set.

See also $set, $setGlobal, $dropLocal and section Compile-Time Variables.

$setNames

Syntax:

```
$setNames file filepath filename fileextension
```

This option establishes or redefines three scoped compile-time variables so they contain the drive subdirectory, filename and extension of a file named with full path. Here `file` is any filename, `filepath` is the name of a scoped compile-time variable that will contain the name of the subdirectory where the file is located, `filename` is the name of a scoped compile-time variable that will contain the root name of the file and `fileextension` is the name of a scoped compile-time variable that will contain the extension of the file.

Example:

```
$setNames "%gams.input%" filepath filename fileextension
$set name %filepath%%filename%%fileextension%
$log %name%
```

The log will show

```
C:\Users\default\Documents\gamside\projdir\
Untitled_1
.gms
C:\Users\default\Documents\gamside\projdir\\Untitled_1.gms
```

Note that `file` is separated into its three components placing `C:\Users\default\Documents\gamside\projdir\` into `filepath`, `Untitled_1` into `filename` and `.gms` into `fileextension`. The three items may be recombined back into the original filename by using `%filepath%%filename%%fileextension%` as shown in the example.

If the file is missing a path, name, or extension the corresponding variable is defined but remains empty as demonstrated in the following example:

```
$onEchoV > showfileparts.gms
$setNames "%1" filepath filename fileextension
$log path=%filepath%
$log name=%filename%
$log ext=%fileextension%
$offEcho
$batInclude showfileparts "C:\tmp\"
$batInclude showfileparts "Untitled_1"
$batInclude showfileparts "Untitled_1.gms"
$batInclude showfileparts "Untitled_1.gms.txt"
```

The log shows:

```
--- Untitled_1.gms(7) 2 Mb
path=C:\tmp
name=
ext=
--- .showfileparts.gms(4) 2 Mb
--- Untitled_1.gms(8) 2 Mb
path=
name=Untitled_1
ext=
--- .showfileparts.gms(4) 2 Mb
--- Untitled_1.gms(9) 2 Mb
path=
name=Untitled_1
ext=.gms
--- .showfileparts.gms(4) 2 Mb
--- Untitled_1.gms(10) 2 Mb
path=
name=Untitled_1.gms
ext=.txt
```

Note that if a file contains multiple `.` the last one will be assigned to the `fileextension` as shown in the example with `Untitled_1.gms.txt`.

$shift

Syntax:

```
$shift
```

This option is similar to the command.com/cmd.exe shift operator (see [en.wikipedia.org/wiki/COMMAND.COM::Bat
(https://en.wikipedia.org/wiki/COMMAND.COM#Batch_file_commands). It shifts the order of all parameters passed once to the *left*. This effectively drops the lowest numbered parameter in the list.

Example:

```
Scalar a, b, c ; a = 1 ;
$batInclude inc.inc a b c
display a, b, c ;
```

The batch include file `inc.inc` follows:

```
%2 = %1 + 1 ;
$shift
%2 = %1 + 1 ;
```

The resulting listing file will contains the following echo print:

```
    1  Scalar a, b, c ; a = 1 ;
BATINCLUDE C:\Users\default\Documents\gamsdir\projdir\inc.inc
    3  b = a + 1 ;
    5  c = b + 1 ;
    6  display a, b, c ;
```

Note that in the first statement in the include file, `%1` is the first argument in the $batInclude call and in this case it is interpreted as `a`. `%2` is the second argument in the $batInclude call and is interpreted as `b`. This leads to the overall assignment being interpreted as `b=a+1`. The dollar control option `$shift` shifts the arguments to the left. As a result, `%1` is interpreted as `b`, and `%2` is interpreted as `c`. This leads to the second assignment being interpreted as `c=b+1`.

Therefore the outcome generated by the display statement in the input file is as follows:

```
----      6 PARAMETER a                       =        1.000
            PARAMETER b                       =        2.000
            PARAMETER c                       =        3.000
```

See also $batInclude.

$show

Syntax:

```
$show
```

This option causes current values of the compile-time variables plus a list of the macros and active input and include files to be shown in the compilation output, where "active" means, that `$show` was either directly in that file or that file is a parent that includes the file with `$show`.

Example:

```
$set it 1
$setLocal yy
$setGlobal gg what
$include myinclude
$macro addx(x)  x+x
$show
```

The file `myinclude.gms` follows:

```
$set inincs
$setLocal inincsl
$setGlobal inincsg
$macro multx(x)  x*x
$show
```

The resulting listing file will contain the following *environment reports* in the compilation output:

```
---- Begin of Active File List
Level Type         Line  Filename
-----------------------------------
    1 INCLUDE         5  C:\Users\default\Documents\gamside\projdir\myinclude.gms
    0 INPUT           4  C:\Users\default\Documents\gamside\projdir\Untitled_1.gms
---- End of Active File List

---- Begin of Compile-time Variable List
Level SetVal                          Type       Text
-------------------------------------------------------
    1 inincsl                         LOCAL
    1 inincs                          SCOPED
    0 yy                              LOCAL
    0 it                              SCOPED     1
    0 gg                              GLOBAL     what
    0 inincsg                         GLOBAL
---- End of Compile-time Variable List

---- Begin of Macro List
$macro multx(x)  x*x
---- End of Macro List
```

and

```
---- Begin of Active File List
Level Type         Line  Filename
---------------------------------
    0 INPUT           6  C:\Users\default\Documents\gamside\projdir\Untitled_1.gms
---- End of Active File List

---- Begin of Compile-time Variable List
Level SetVal                          Type       Text
-------------------------------------------------------
    0 yy                              LOCAL
    0 it                              SCOPED     1
    0 gg                              GLOBAL     what
    0 inincsg                         GLOBAL
---- End of Compile-time Variable List

---- Begin of Macro List
$macro multx(x)  x*x
$macro addx(x)  x+x
---- End of Macro List
```

Note that only the macros and the item defined with the option $setGlobal in the included file carries over.

If one needs only parts of this report, the dollar control options $showFiles, $showMacros and $showVariables can be used.

See also section Compile-Time Variables.

$showFiles

Syntax:

```
$showFiles
```

This option prints the active files to the lst file as described at $show.

$showMacros

Syntax:

```
$showMacros
```

This option prints a list of macros to the lst file as described at $show.

$showVariables

Syntax:

```
$showVariables
```

This option prints a list current values of the compile-time variables to the lst file as described at $show.

$single

Syntax:

```
$single
```

The lines following this option will be echoed single spaced in the compilation output. Note that this is the default. The option is only useful as a switch to deactivate the option $double.

Example:

```
Set    i / 1*2 / ;
Scalar a / 1   / ;
$double
Set    j / 10*15 / ;
Scalar b / 2     / ;
$single
Set    k / 5*10 / ;
Scalar c / 3    / ;
```

The echo print in the resulting listing file will look as follows:

```
1  Set i / 1*2 / ;
2  Scalar a /1/ ;

4  Set j / 10*15 / ;

5  Scalar b /2/ ;
7  Set k / 5*10 / ;
8  Scalar c /3/ ;
```

Note that lines between the options $double and `$single` are listed double spaced, while the lines after the option `$single` revert back to being listed single spaced.

See also $double.

$splitOption

Syntax:

```
$splitOption KEYVALPAIR optname optvalue
```

Establishes or redefines two scoped compile-time variables so they contain the name and value of an option key/value pair specified in various formats. `KEYVALPAIR` is a string formatted as `-opt=val` or `-opt val` (instead of `-` one can also use `/`). `optname` is the name of a scoped compile-time variable that will contain the name of the option and `optvalue` is the name of a scoped compile-time variable that will contain the value of the option. This is useful in particular in combination with batInclude files.

Example:

```
$onechoV > myinclude.gms
* Default values for named arguments
$setGlobal a1 1
$setGlobal a2 2
$setGlobal a3 nothing
$setGlobal positionalArgs
$label ProcessNamedArguments
$  splitOption "%1" key val
$  if x%key%==x $goto FinishProcessNamedArguments
$  ifThenI.NamedArguments %key%==a1
$    setGlobal a1 %val%
$  elseIfI.NamedArguments %key%==a2
```

```
$    setGlobal a2 %val%
$  elseIfI.NamedArguments %key%==a3
$    setGlobal a3 %val%
$  else.NamedArguments
$    error Unkown named argument "%key%"
$  endIf.NamedArguments
$  shift
$goTo ProcessNamedArguments
$label FinishProcessNamedArguments
$setGlobal positionalArgs %1 %2 %3
$offEcho
$batInclude myinclude "-a3=some things" -a2=3.14 i j k
$log Using named arguments a1:>%a1%< a2:>%a2%< a3:>%a3%< positionalArgs:>%positionalArgs%<
```

Now when calling this piece of code as a batInclude one can specify optionally some named arguments (in any order) right after the name of the batInclude file and before the positional arguments as demonstrated by the log output:

```
Using named arguments a1:>1< a2:>3.14< a3:>some things< positionalArgs:>i j k<
```

stars (****)

Syntax:

```
$stars char[char][char][char]
```

This option is used to redefine the **** marker in the GAMS listing file. By default, important lines like those that denote errors and the solver and model status are prefixed with ****. A new marker consists of one to four characters.

Example:

```
$stars *###*
garbage
```

The resulting listing file follows:

```
    2  garbage
*###*         $140
*###*  $36,299  UNEXPECTED END OF FILE (1)

Error Messages
 36  '=' or '..' or ':=' or '$=' operator expected
     rest of statement ignored
140  Unknown symbol
299  Unexpected end of file
```

$sTitle

Syntax:

```
$sTitle text
```

This option sets the subtitle in the page header of the listing file to `text`. Note that the next output line will appear on a new page in the listing file.

Example:

```
$sTitle Data tables for input/output
```

See also $title.

$stop

Syntax:

```
$stop [text]
```

This option stops program compilation without creating an error. Note there is a difference to the option $exit. If there is only one input file, `$stop` and `$exit` will have the same effect. In an include file the option $exit acts like an end-of file on the include file. However, the option `$stop` in an include file will cause GAMS to stop reading all input but continue the execution phase of the so far compiled program. The text followed by `$stop` is ignored.

Example:

```
$ifthen not set EXPORTEXCEL
$  stop No export to Excel
$else
$  call gdxxrw ...
$endif
```

See also $abort, $error, $exit, and $terminate.

$sysInclude

The syntax of this dollar control option is equivalent to the syntax of $batinclude:

Syntax:

```
$sysinclude external_file arg1 arg2 ...
```

However, if an incomplete path is given, the file name is completed using the system include directory. By default, the system include directory is set to the GAMS system directory. Note that the default directory may be reset with the command line parameter sysIncDir.

Example:

The only relevant include file in the GAMS system directory is `mpsgeset` for MPSGE models, see for example [**HARMGE**]:

```
$sysInclude mpsgeset KAMIYA
```

Note that this call will first look for the include file `[GAMS System Directory]/mpsgeset`. If this file does not exist, it will looks for `[GAMS System Directory]/mpsgeset.gms`. The argument `KAMIYA` is passed on to the include file and are interpreted as explained for the dollar control option $batInclude.

Consider the following example:

```
$sysInclude C:\Users\default\Documents\mpsgeset KAMIYA
```

This call will first look specifically for the include file `C:\Users\default\Documents\mpsgeset` and next for `C:\Users\default\Documents\mpsgeset.gms`.

See also $batInclude.

$terminate

Syntax:

```
$terminate [text]
```

This option terminates compilation and also does not execution to program compiled so far without giving an error.

Example:

```
$if set JUSTTERMINATE $terminate
```

See also $abort, $error, $exit, and $stop.

$title

Syntax:

```
$title text
```

This option sets the title in the page header of the listing file to `text`. Note that the next output line will appear on a new page in the listing file.

Example:

```
$title  Production Planning Model
$sTitle Set Definitions
```

See also $sTitle.

$unLoad

Syntax:

```
$unLoad [sym1[,] sym2=gdxSym2[,] ...]
```

This option unloads specified items to a GDX file. Note that `$unLoad` must be used in conjunction with the option $gdxOut: `$gdxOut` must precede `*$unLoad*`. More than one option `$unload` may appear in between. Symbols can be renamed via the `sym=GDXSym` syntax. A `$unLoad` without arguments unloads the entire GAMS database into the GDX file.

Example: Consider the following slice of code:

```
Sets i   'canning plants'   / seattle, san-diego /
     j   'markets'          / new-york, chicago, topeka / ;

Parameters
     a(i)  'capacity of plant i in cases'
       /    seattle     350
            san-diego   600  /

     b(j)  'demand at market j in cases'
       /    new-york    325
            chicago     300
            topeka      275  / ;

Table d(i,j)  'distance in thousands of miles'
                    new-york       chicago      topeka
     seattle           2.5           1.7          1.8
     san-diego         2.5           1.8          1.4  ;

$gdxOut tran
$unLoad i j=k
$unLoad b=dem a=sup
$unLoad d
$gdxout tranX
$unLoad
```

Note that the last lines will create a file named `tran.gdx` that contains `i`, `j` (now named `k`) and `d` and the parameters `a` and `b` which are now named `dem` and `sup`. The `$unLoad` in the very last line creates a GDX file `tranX.gdx` with all symbols (with their original names). The table of content (via `$gdxIn` and `$load` without parameters) of these two files looks as follows:

```
Content of GDX C:\Users\default\Documents\gamsdir\projdir\tran.gdx
              5 UELs

    Number Type       Dim       Count  Name
         1 Set          1           2  i           canning plants
         2 Set          1           3  k           markets
         3 Parameter    1           3  dem(j)      demand at market j in cases
         4 Parameter    1           2  sup(i)      capacity of plant i in cases
         5 Parameter    2           6  d(i,j)      distance in thousands of miles

Content of GDX C:\Users\default\Documents\gamsdir\projdir\tranX.gdx
              5 UELs

    Number Type       Dim       Count  Name

         1 Set          1           2  i           canning plants
         2 Set          1           3  j           markets
         3 Parameter    1           2  a(i)        capacity of plant i in cases
         4 Parameter    1           3  b(j)        demand at market j in cases
         5 Parameter    2           6  d(i,j)      distance in thousands of miles
```

Both listings show domain information for the various symbols but only the file `tranX.gdx` created with `$unLoad` without arguments has consistent or *full* domain information while `tran.gdx` may have inconsistent or *relaxed* domain information. Domain matching when loading with the `$load sym<[=]symGDX`, see $load for details, can be used in both cases but can become subtle with the relaxed domain information from `tran.gdx`.

Starting with GAMS 34 the behavior of `$gdxOut` and `$unload` changed compare to previous versions. In order to write consistent domain information and write aliases as such, the actual writing of the symbols to GDX is *delayed* until the GDX files is about to be closed either through an explicit `$gdxOut` (potentially with a another filename) or through the end of the compilation phase. The advantage of the delayed writing is that the compiler can now better analyze the symbols and e.g. write alias symbols as aliases with the aliased set is also written to the GDX file (aliases were always written as sets before GAMS 34).

This new behavior can result in different GDX files. Here are two examples:

```
Set i /1,2,3/; Parameter p(i);
$gdxOut x.gdx
$unload
Parameter p(i) /1 1, 2 2, 3 3/;
```

In the GAMS versions before 34 the export GDX happened when the compiler processed the `$unload` command. At this point the parameter `p` did not have any data yet, so the GDX symbol `p` had no records. With the new behavior, the actual writing happens when the GDX file will be closed. Since there is no explicit close via a `$gdxOut` the GDX file is closed at the end of the compilation phase. By then the compiler has processed the statement `Parameter p(i) /1 1, 2 2, 3 3/;` and has filled `p` with data that will show up in the GDX file. In such a case an explicit `$gdxOut` after the `$unload` helps to get the old behavior back.

```
Set i /1,2,3/; Parameter p(i) / 1 1/;
$gdxOut x.gdx
$unload p=p1
$onmulti
Parameter p(i) / 2 2/;
$unload p=p2
```

In the GAMS versions before 34 the GDX file `x.gdx` contains two parameters `p1` with record `1 1` and `p2` with record `1 1, 2 2`. With the new behavior the symbols are written again at the end of the compilation phase and by then the parameter `p` has the records `1 1, 2 2` and will write to GDX the parameters `p1` and `p2` with the two records each. The only way to accomplish the old behavior in this case is to write `p1` and `p2` to different GDX files:

```
Set i /1,2,3/; Parameter p(i) / 1 1/;
$gdxOut x1.gdx
$unload p=p1
$gdxOut x2.gdx
$onMulti
Parameter p(i) / 2 2/;
$unload p=p2
```

This delayed writing also influence other options like $onEpsToZero. The state of this becomes relevant when the file is actually written, not when a symbol is added to the list of symbols to be exported with `$unload`. This can be seen in the following example:

```
Set       i      / i1*i3 /;
Parameter p1(i)  / i1 0, i2 EPS, i3 1  /
          p2(i)  / i1 0, i2 EPS, i3 1  /;

$gdxOut out.gdx
$onEpsToZero
$unload p1
$offEpsToZero
$unload p2
$gdxOut

$call.checkErrorLevel gdxdump out.gdx
```

Here is the output from `gdxdump` at the end:

```
Parameter p1(*) /
'i2' Eps,
'i3' 1 /;

Parameter p2(*) /
'i2' Eps,
'i3' 1 /;
```

$use205

Syntax:

```
$use205
```

This option sets the GAMS syntax to the syntax of Release 2.05. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since Release 2.05. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow to run such models.

Example:

```
$use205
Set if /1.2.3/;
Scalar x ;
```

The word "if" is a keyword in GAMS that was introduced with the first version of Release 2.25. Setting option `$use205` allows "if" to be used as an identifier since it was not a keyword in Release 2.05.

$use225

Syntax:

```
$use225
```

This option sets the GAMS syntax to the syntax of the first version of Release 2.25. This is mainly used for backward compatibility. New keywords have been introduced in the GAMS language since the first version of Release 2.25. Models developed earlier that use identifiers that have since become keywords will cause errors when run with the latest version of GAMS. This option will allow to run such models.

Example:

```
$use225
Set for /1.2.3/;
Scalar x ;
```

The word "for" is a keyword in GAMS that was introduced with the later versions of Release 2.25. Setting option `$use225` allows "for" to be used as an identifier since it was not a keyword in the first version of Release 2.25.

$use999

Syntax:

```
$use999
```

This option sets the GAMS syntax to the syntax of the latest version of the compiler. Note that this setting is the default.

Example:

```
$use225
Set for /1.2.3/;
Scalar x ;
$use999
for (x=1 to 3, display x) ;
```

Note that the word "for" is used as a set identifier after setting the option `$use225` and later the keyword `for` is used in a looping construct after having set the language syntax to that of the latest version using the option `$use999`.

$version

Syntax:

```
$version n
```

This issues a compilation error if `n` is greater than the current GAMS version. This can be useful to ensure that a model is run only with new versions of GAMS, because, e.g., a particular feature which did not exist in older versions is needed.

Example:

```
* With GAMS 24.8.1 the function numCores was added to the system.
* Make sure, that we use this GAMS version or newer.
$version 248

Scalar nc "Number of cores";
nc = numCores;
Display nc;
```

$warning

Syntax:

```
$warning text
```

This dollar control option issues a compilation warning to the log and listing but continues compilation and execution.

Example

```
$ifthen not set INPUTFILE
$ set INPUTFILE default.txt
$ warning Using default INPUTFILE "default.txt". Use --INPUTFILE=myfile.txt to overwrite default
$endif
```

The GAMS log file will issue a warning:

```
*** Error 332 in C:\Users\default\Documents\gamsdir\projdir\myinput.gms
    $Warning encountered - see listing for details
```

with the details in the listing file:

```
   3  $ warning Using default INPUTFILE "default.txt". Use --INPUTFILE=myfile.txt to overwrite d
****       $332
```

4.41.4 Conditional Compilation

GAMS offers several dollar control options that facilitate conditional compilation. In this section we will first introduce the general syntax, present an overview of all relevant options and list the conditional expressions that may be used to perform tests. Then we will give several examples to illustrate how these options are used and to demonstrate their power. This section is meant as an introduction to conditional compilation in GAMS and complements the detailed descriptions of the dollar control options listed in Table 1 below.

4.41.4.1 Conditional Compilation: General Syntax and Overviews

The dollar control option `$if` and its variants provide a great amount of control over conditional processing of the input file(s). The syntax in GAMS is similar to the `IF` statement of the DOS Batch language:

```
$if [not] <conditional expression> new_input_line
```

The dollar control statement begins with `$if`. Note that `$if` may be replaced by one of its variants that are listed in Table 1 below. The operator `not` is optional and makes it possible to negate the `conditional expression` that follows. The conditional expression may take various forms, a complete list is given in Table 2. The result of the conditional test is used to determine whether to process or not the remainder of the line, `new_input_line`, which may be any valid GAMS input line.

Attention

The first non-blank character on the line following the conditional expression is considered to be the first column position of the GAMS input line. Therefore, if the first character encountered is the dollar control character, the line is treated as a dollar control line. Likewise, if the first character encountered is a comment character (default: *) the remainder of the line is treated as a comment line. If a multiplication is to be performed in the true clause instead of a comment, the $ifThen condition needs to be used:

```
$setglobal mult on
Scalar P_test;
P_test = 10
$iftheni %mult%==on
  * 5
$endif
;
```

Alternatively, the `new_input_line` may be placed in the next line. The corresponding syntax follows:

```
$if [not] <conditional expression>
new_input_line
```

Note that in this version the space after the conditional expression is left blank. If the conditional is found to be false, either the remainder of the line (if any) will be skipped or the next line will not be processed.

The overviews in Table1 and Table 2 conclude this subsection. Examples are given in the next subsection.

Table 1: $*if and Related Dollar Control Options*

Dollar Control Option	Description
$if	This option is used to do case sensitive comparisons. Several examples are given in the next subsection.
$ifE	This variant does the same as $if but allows numerical constant expression evaluation. For an example, see the detailed description of this option.
$ifI	This variant is the same as $if, but it is case insensitive.
$ifThen	This variant controls whether a block of statements will be processed or not. It is used to do case sensitive comparisons. Most often it is followed by one or more of the following dollar control options: $else, $elseIf, $elseIfI, $elseIfE. The option `$ifThen` must be matched with the option $endIf that marks the end of the block. An example is given below.
$ifThenE	This is a variant of `$ifThen` and is used for numerical comparisons. Like `$ifThen`, it is often followed by the option $else or one of its variants and must be matched with the option $endIf that marks the end of the construct.
$ifThenI	This is a variant of `$ifThen` and is used to do case insensitive comparisons. Like `$ifThen`, it is often followed by the option $else or one of its variants and must be matched with the option $endIf that marks the end of the construct.
$endIf	This option must be matched with a preceding option $ifThen, $ifThenE or $ifThenI and marks the end of the `if - then` construct. Note that the option `$endIf` is *not* followed by a conditional expression, but it may be follwowed by a `new_input_line`. This GAMS input is restricted to other dollar control statements. An example is given below.

Dollar Control Option	Description
$else	This option follows the option $ifThen, $ifThenE or $ifThenI. It is followed by an instruction which is executed if the conditional expression of the matchining `$ifThen` statement is not true. Note that therefore this dollar control statement does *not* contain a conditional expression. An example is given below.
$elseIf	This option follows the option $ifThen, $ifThenE or $ifThenI. It is follwed by another conditional expression and instruction. Note that this option is case sensitive. An example is given below.
$elseIfE	This is a variant of $elseIf that evaluates numerical values.
$elseIfI	This is a variant of $elseIf that is case insensitive.

Table 2: ***Conditional Expressions in Conditional Compilation***

Conditional Expression	Description
`acrType id`	True if `id` is an acronym.
`decla_OK`	True if a declaration statement is permitted in the current line. Note that declaration statements are not permitted within programming flow control structures like if statements or loop statements. An example is given below.
`declared id`	True if `id` was declared.
`defined id`	True if `id` was defined. An example is given below.
`dExist directoryname`	True if a directory with the name `directoryname` exists.
`dimension n id`	True if `id` has `n` dimensions. Note that `n` may take values from 0 to maximum number of possible indexes (see Dimensions).
`equType id`	True if `id` is an equation.
`errorFree`	True if compilation up to this point has been free of errors.
`errorLevel n`	True if the return code of a program called via $call is equal to or larger than `n`. For lists of GAMS return codes, see chapter GAMS Return Codes. An example is given below.
`exist filename`	True if a file with the name `filename` exists and is readable in the working directory or an input file directory. If no extension is specified, also `filename.gms` is checked. An example is given below.
`filType id`	True if `id` is the name of a put file.
`funType id`	True if `id` is a GAMS function.
`gamsVersion n`	True if current GAMS version is greater than or equal to `n`. GAMS versions are referenced with a single number. For example, if it should be tested whether the current GAMS version is 24.7 or newer, `n` will equal 247. Maintenance version numbers, e.g. 24.7.**4** do not count.
`gdxDimension n id`	True if `id` exists in a GDX file previously opened with $gdxIn and has `n` dimensions. Note that `n` may take values from 0 to maximum number of possible indexes (see Dimensions).
`gdxEquType id`	True if `id` exists in a GDX file previously opened with $gdxIn and is an equation.
`gdxParType id`	True if `id` exists in a GDX file previously opened with $gdxIn and is a parameter.
`gdxSetType id`	True if `id` exists in a GDX file previously opened with $gdxIn and is a set.
`gdxSymExist id`	True if `id` exists in a GDX file previously opened with $gdxIn.
`gdxVarType id`	True if `id` exists in a GDX file previously opened with $gdxIn and is a variable.
`macType id`	True if `id` is a macro.

Conditional Expression	Description
`modType id`	True if `id` is a model.
`onState key`	True if the state of `key` is `on` (i.e. activated by `$onKey`). An example is given below.
`parType id`	True if `id` is a parameter.
`preType id`	True if `id` is a one of the predefined symbols in GAMS. Details are given below.
`putOpen`	True if both a file statement and at least one put statement have been compiled. Note that this does not guarantee that a file will be open at runtime.
`readable id`	True if `id` was correctly initialized, i.e. the symbol has a data statement or appeared on the left-hand side of an assignment statement, and may therefore be used on the right-hand side of an assignment statement. An example is given below.
`set varname`	True if the scoped compile-time variable `varname` was set with the dollar control option $set, $setGlobal or $setLocal.
`setEnv varname`	True if the environment variable `varname` was set in the systems environment, e.g. with the dollar control option $setEnv.
`setGlobal varname`	True if the control variable `varname` was set with the dollar control option $setGlobal.
`setLocal var_name`	True if the control variable `var_name` was set with the dollar control option $setLocal.
`setType id`	True if `id` is a set.
`solver solver_name`	True if a solver named `solver_name` exists in the GAMS system. An example is given below.
`uelExist id`	True if unique element `id` exists in current GAMS database.
`varType id`	True if `id` is a variable.
`warnings`	True if the compilation until this point has been free of warnings.
`xxxType id`	True if `id` is an unknown type. For more information, see below.
`string1 == string2`	True only if `string1` matches `string2` exactly. Note that the strings may be quoted or unquoted. Null (empty) strings may be indicated by an empty quote: `""` or `''`. The case of the strings provided either explicitly or, more likely, through a parameter substitution, is preserved and therefore will affect the string comparison. Quoted strings with leading and trailing blanks are not trimmed and the blanks are considered part of the string. Note that the string may have the form `%VARNAME%`, where `VARNAME` refers to a compile-time variable including GAMS command line parameters and system attribute. An example for a string comparison with a command line parameter is given below. An example for a string comparison with a system attribute is given below.
`string`	True only if `string` is an empty string (`""` or `''`), otherwise False.

4.41.4.2 Conditional Compilation: Examples

File Operation Test

The operator `exist` may be used to test whether a given file name exists. Consider the following example:

```
$if exist myfile.dat $include myfile.dat
```

Observe that the effect of this dollar control statement is that the file `myfile.dat` is included if it exists. Note that the character `$` at the beginning of the option $include is the first non-blank character after the conditional expression `exist myfile.dat` and therefore it is treated as the first column position. The statement above may also be written as follows:

```
$if exist myfile.dat
$include myfile.dat
```

Conditional Compilation and Batch Include Files

In the next example we will illustrate how the option `$if` is used inside a batch include file where parameters are passed through the option $batInclude from the parent file:

```
$if not "%1a" == a $goto labelname
$if exist %1 file.ap=1;
```

Note that in the first line the `$if` condition uses the string comparison `"%1a" == a` to check if the parameter is empty. This test may also be done in the following way: `%1 == ""`. If the parameter is not empty, the option $goto is processed.

Note

> The option $label cannot be part of the conditional input line. However, if the option $label appears on the next line, the condition decides once if the label is placed or not and subsequent instances of `$goto` will find the label without reevaluating the condition.

The second line illustrates the use of standard GAMS statements if the conditional expression is valid. If the file name passed as a parameter through the $batInclude call exists already, the GAMS will execute the `file.ap=1;` statement which will append to the file.

The next example demonstrates how an unknown number of file specifications may be passed on to a batch include file that will include each of them if they exist. The batch include file could look as follows:

```
* Batch Include File - inclproc.gms
* include and process an unknown number of input files
$label nextfile
* Quote everything because file name might have blanks
$if exist "%1"  $include "%1"
$shift
$if not "%1a" == a $goto nextfile
```

The call to this file in the parent file could take the following form:

```
$batInclude inclproc "file 1.inc" file2.inc file3.inc file4.inc
```

Testing Whether an Item Has Been Defined

The next example shows how to test if a named item was declared and/or defined.

```
Set i;
$if defined  i $log First: set i is defined
$if declared i $log First: set i is declared
Set i /seattle/;
$if defined  i $log Second: set i is defined
$if declared i $log Second: set i is declared
```

Note that after the first declaration of `i` only `declared i` evaluates to true when after the second declaration with a data statement both `defined i` and `declared i` are true.

Testing the state of a flag set by a dollar control option

The expression `onState key` tests whether the state of `key` is `on`. So the following example could be done with any `key` available as a `$(on|off)Key` dollar control option:

```
$if onState listing $log Listing is active at checkpoint 1
$if onState listing $set reactivateListing
$offListing
* Do something which should not be seen in the listing
$if onState listing $log Listing is active at checkpoint 2
* Reactivate listing if it was active at the start only
$if set reactivateListing $onListing
$if onState listing $log Listing is active at checkpoint 3
```

Testing Whether an Item May Be Used in an Assignment

The expression `readable id` tests whether data were assigned to an item and therefore the item may be used on the right-hand side of an assignment statement. Consider the following example:

```
Scalar f;
$if not readable f $log f cannot be used on the right
Scalar f /1/;
$if readable f $log f can be used on the right
$kill f
$if not readable f $log f cannot be used on the right after clear
f = 1;
$if readable f $log f can be used on the right after assignment
```

Note that in the first test the set `f` was declared, but there was no data statement, hence it is not `readable`. After a declaration with a data statement the test `readable f` evaluates to `TRUE`. With $kill we can revert `f` to a data less state and hence `not readable f` is `TRUE` after the "$kill". The assignment statement `f = 1;` make the scalar `f` readable again.

Testing Whether an Identifier May Be Declared

In programming flow control structures, like if statements or loop statements declaration statements are not permitted. The test `decla_ok` may be used to test whether the current environment allows declaration statements. Consider the following example:

```
$if decla_ok $log declarations are possible
if(1,
$  if not decla_ok $log declarations are not allowed
);
```

Note that the conditional expression in the both `$if` tests will evaluate to `TRUE`. However, the second test of `decla_ok` itself will be `FALSE` because it is processed while compiling an `if` statement, but with the `not` the entire expression evaluated to `TRUE`. For more information, see chapter Programming Flow Control Features.

Comments in the Context of Conditional Compilation

In-line and end-of-line comments are stripped out of the input file before processing the `new_input_line`. If either of these forms of comments appear, they will be treated as blanks. Consider the following example:

```
Parameter a ;
a=10 ;
$eolCom // inlineCom /*  */
$if exist myfile.dat  /* in line comments */  // end of line comments
a = 4 ;
display a;
```

Note that the comments on line 3 are ignored and the fourth line with the assignment statement will be processed if the conditional expression is true Hence the outcome generated by the display statement will list `a` with a value of 4 if the file `myfile.dat` exists and a value of 10 if the file does not exist.

Error Level Test

Consider the following example:

```
$call gams mymodel.gms lo=2
$if errorlevel 1 $abort one or more errors encountered
```

Note that the errorlevel is retrieved from the previous system call via $call. The conditional statement `errorlevel 1` is true if the returned errorlevel is equal to or larger than 1. In case of calling GAMS this means that something was not quite right with the execution of GAMS (either a compilation or execution error or other more exotic errors, see GAMS return codes. If this is the case, this GAMS program will be aborted immediately at compilation time.

Usually programs return 0 on success and non-zero on failure. The `$if errorlevel 1` checks for strictly positive return codes. There are rare cases with failures and negative return codes (e.g. on Windows if some DLL dependencies of the program can't be resolved). In such a case `$if errorlevel 1` will evaluate to false and not continue with the `$abort` instruction. It might be better to access the program return code via the errorLevel function in the following way:

```
$call gams mymodel.gms lo=2
$ifE errorLevel<>0 $abort one or more errors encountered
```

Solver Test

The following example illustrates how to check if a solver exists.

```
$if solver ZOOM
```

Note that the conditional expression is false since the solver named ZOOM does not exists in the GAMS system (anymore).

Command Line Parameters in String Comparison Tests

Assume we include the following dollar control statements in a GAMS file called myfile.gms:

```
$if not '%gams.ps%'==''  $log Page size set to  %gams.ps%
$if not '%gams.pw%'==''  $log Page width set to %gams.pw%
$if not '%gams.mip%'=='' $log MIP solver default is %gams.mip%
```

Then we run the program with the following call:

```
> gams myfile pageSize=60  pageWidth=85 mip=cbc
```

Note that we specified values for the command line parameters pageSize, pageWidth, and MIP. We can either use the short or long name on the command line and in the compile-time variable. If we do not specify the option on the command line we will get the default value for option page size and page width. The MIP solver line will not show because %gams.mip% remains empty. The log with option setting on the command line will include the following lines:

```
Page size set to  60
Page width set to 85
MIP solver default is cbc
```

Command line parameters are introduced in chapter The GAMS Call and Command Line Parameters.

System Attributes in String Comparison Tests

Compile-time system attributes may also be used in string comparison tests. The system attribute that is most useful in this context is .fileSys. It identifies the name of the operating system being used. Consider the following example:

```
$ifthen not %gams.logOption%==3
$  ifi %system.fileSys%==UNIX  $set nullFile > /dev/null
$  ifi %system.fileSys%==MSNT  $set nullFile > nul
$  if not set nullFile $abort %system.fileSys% not recognized
$else
$  set nullFile
$endif
$call gamslib trnsport %nullFile%
```

These dollar control statements allow the definition of a NULL file destination that is dependent on the operating system that is being used. Note that the control variable nullFile contains an operating-system-dependent name. This is useful when making an external program that writes to STDOUT quiet in case the GAMS log does not go to STDOUT (logOption=3). This example could also have used the system attribute %system.nullFile% which contains the operating-system-dependent NULL file destination:

```
$set nullFile
$if not %gams.logOption%==3 $set nullfile > %system.nullFile%
$call gamslib trnsport %nullfile%
```

System attributes in general are introduced in chapter System Attributes.

Conditional Compilation with $ifThen and $else

Consider the following example which illustrates the use of `$ifThen`, `$elseIf`, `$else` and `$endif`:

```
$set x a
$label test
$ifThen %x% == a $set x 'c' $log $ifThen   with x=%x%
$elseIf %x% == b $set x 'k' $log $elseIf 1 with x=%x%
$elseIf %x% == c $set x 'b' $log $elseIf 2 with x=%x%
$else            $set x 'e' $log $else     with x=%x%
$endIf $if not %x% == e $goTo test
```

Note that the resulting log file will contain the following lines:

```
$ifthen   with x=a
$elseif 2 with x=c
$elseif 1 with x=b
$else     with x=k
```

Observe that the options `$else` and `$endIf` are not followed by conditional expressions and the instruction following the option `$endIf` contains a dollar control statement. Moreover, note that the '$set x 'c' has the text to be set in quotes. GAMS needs to know where the text ends and the next dollar control option (in this case $log`) starts.

Type of Identifiers

The type of a symbol can be retrieved via `$if ...Type`. Consider the following example:

```
Set diag / 1*3 /;
Parameter p(diag) / 1 1, 2 4, 3 8 /;
$if     setType diag    $log diag   is a set
$if not varType diag    $log diag   is not a variable
$if     preType diag    $log diag   is a predefined type
$if     parType p       $log p      is a parameter
$if     setType sameAs  $log sameAs is a set
$if     preType sameAs  $log sameAs is a predefined type
```

Note that for predefined symbols more than one type applies (e.g. `sameAs` is of set and predefined type). Please also note that `diag` is a set even though there is a predefined symbol named `diag` but that becomes invisible with a user defined symbol with the same name.

Normally there is no way to get a symbol into the GAMS symbols table without a proper type. However, if the dollar command line parameter multiPass is set to a value larger than zero, then the compiler will just check for some integrity and will try to deduce the symbol type from the context. If it is not able to do so, the symbol type will remain unknown. For example, compiling the following lines with `multiPass=1`

```
display x;
$if xxxType x $log x is of unknown type
```

result in the line `x is of unknown type` in the GAMS log.

4.41.5 Macros in GAMS

Macros are widely used in computer science to define and automate structured text replacements. The GAMS macro processors function similarly to the popular C/C++ macro preprocessor. Note that the GAMS macro facility has been inspired by the [GAMS-F] (http://www.mpsge.org/inclib/gams-f.htm) preprocessor for function definition developed by Michael Ferris, Tom Rutherford and Collin Starkweather, 1998 and 2005. The GAMS macro facility incorporates the major features of the GAMS-F preprocessor into the standard GAMS release as of version 22.9. GAMS macros act like a standard macro when defined. However, their recognition for expansion is GAMS syntax driven.

4.41.5.1 Syntax and Simple Examples

The definition of a macro in GAMS takes the following form:

```
$macro name  macro_body
$macro name(arg1,arg2,arg3,...) macro_body with tokens arg1, ...
```

The dollar symbol `$` followed by `macro` indicate that this line is a macro definition. The name of the macro has to be unique, similar to other GAMS identifiers like sets and parameters. The macro name is immediately followed by a list of replacement arguments `arg1,arg2,arg3,...` that are enclosed in parentheses. The macro body is not further analyzed after removing leading and trailing spaces.

The recognition and following expansion of macros is directed by GAMS syntax. The tokens in the macro body to be replaced by the actual macro arguments follow the standard GAMS identifier conventions. Consider the following simple example of a macro with one argument:

```
$macro reciprocal(y) 1/y
```

Here the name of the macro is `reciprocal`, `y` is the argument and the macro body is `1/y`. This macro may be called in GAMS statements as follows:

```
$macro reciprocal(y) 1/y
scalar z, x1 /2/, x2 /3/;
z = reciprocal(x1) + reciprocal(x2);
```

As GAMS recognizes `reciprocal(x1)` and `reciprocal(x2)` as macros, the assignment statement will expand to:

```
z = 1/x1 + 1/x2;
```

Note that the macro use cannot be spread over multiple lines. Single line use with more complex arguments as in

```
z = reciprocal(x1+x2);
```

is recognized and will expanded to:

```
z = 1/x1+x2;
```

Note that z will be equal to 3.5 in this example as division takes precedence over addition. But breaking the argument of the macro over multiple lines as follows will trigger a macro expansion error:

```
z = reciprocal(x1
              +x2);
```

The next example illustrates macros with multiple arguments:

```
$macro ratio(x,y) x/y
scalar z, x1 /2/, x2 /3/;
z = ratio(x1,x2);
```

The assignment above will expand to:

```
z= x1/x2;
```

Note that the macro definition may extend over several lines with the symbol '\' acting as a continuation string. Consider the following example:

```
$macro myxor(a,b)  (a or b) \
               and (not a or not b)
scalar z;
z = myxor(1,0);
display z;
```

The z assignment expands to

```
z = (x1 or x2) and (not x1 or not x2);
```

Note that although the macro has been defined over two lines, the expansion happens by combining the lines after stripping leading white spaces of the second line as demonstrated in the next example (because and has a higher precedence than or we can omit the parenthesis):

```
$macro myxor(a,b)  not a and b \
                or a and not b
scalar z;
z = myxor(1,0);
display z;
```

The z assignment expands to this:

```
z = not 1 and 0 or 1 and not 0;
```

The &, explained in more detail in the next section can be used to preserve (some of the) leading white spaces (but not the line breaks) if that is desired:

```
$macro myxor(a,b)  not a and b \
       &        or a and not b
```

4.41.5.2 Nested Macros

Macros may be nested. Consider the following example:

```
$macro product(a,b) a*b
$macro addup(i,x,z) sum(i,product(x(i),z))
set j /j1*j10/;
Parameter a1(j) / #j 1 /, z, x1 /5/;
z = addup(j,a1,x1);
```

Observe that the macro `product` is nested in the macro `addup`. The assignment will expand to:

```
z = sum(j,a1(j)*x1);
```

Note that nested macros may result in an expansion of infinite length. An example follows.

```
$macro a  b,a
display a;
```

This will expand into:

```
display b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,b,...
```

In such a case GAMS will eventually refuse to do more substitutions and will issue a compilation error:

```
732  Too many edits on one single line - possible recursion in macro calls
        compilation will be terminated
```

4.41.5.3 Ampersands in Macro Definitions

The expansion of arguments may be more carefully controlled by the use of ampersands `&` in the macro body. A single ampersand `&` is used as a concatenation or separation symbol to indicate tokens that are to be replaced. Consider the following example:

```
$macro  f(i)  sum(j, x(i,j))
$macro equ(q)  equation equ_&q;  \
               equ_&q.. q =e= 0;
set i /i/, j /j/;
variable x(i,j);
equ(f(i))
```

This will expand into:

```
equation equ_f(i);equ_f(i).. sum(j, x(i,j)) =e= 0;
```

Note that without the ampersand notation, GAMS would have recognized only the third occurrence of `q` and hence the expansion would have been:

```
equation equ_q;equ_q.. sum(j, x(i,j)) =e= 0;
```

Two ampersands `&&` immediately preceding a token will drop the most outer matching single or double quotes of the replacement argument. This makes it possible to include expressions with spaces, commas and unbalanced parentheses. The latter one is something users should really avoid doing. An example follows.

```
$macro d(q) display &&q;
$macro ss(q) &&q)
set i /i/, k /k/;
parameter a1(i) / i 1/, z;
d('"here it is" , i,k')
d('"(zz"')
z=ss('sum(i,a1(i)');
z=ss('prod(i,a1(i)');
```

Note that the expressions `d` contain quotes, spaces and commas and the expression `ss` has unbalanced parentheses within the quoted parts. In turn these expand to become:

```
display "here it is" , i,k;
display "(zz";
z=sum(i,a1(i));
z=prod(i,a1(i));
```

4.41.5.4 Additional Macro Features

Deeply nested macros may require aliased sets in indexed operations like `sum` and `prod`. A minor syntax extension allows the implicit use of aliases. The suffix `.local` on a controlling set will use an implicit alias within the scope of the indexed operation. Consider the following example:

```
$macro ratio(a,b) a/b
$macro total(q) sum(i,q(i))
set i /i1*i15/;
parameter a(i) / #i 1 /, b(i) / #i 2 /, r(i), asum;
asum = total(a);
r(i) = ratio(total(a), b(i));
```

The assignment statement will expand to:

```
asum = sum(i,a(i));
r(i) = sum(i,a(i))/b(i);
```

The second line will not compile because the `i` in the sum is already controlled from the `i` on the left. The intention was the `total` macro is to add up the elements of a parameter indexed over `i`. As in the `r(i)` assignment the macro might be used in a statement where `i` is already controlled hence when doing the `sum` in the macro we want to use an alias of `i`. If we change the macro definition to

```
$macro total(q) sum(i.local,q(i))
```

The code works as expected because the `i` in the sum refers to the `i.local` and not the outside `i`.

Note that the the modifier `.local` is not limited to macros and may be used in any context. For further details and more examples, see the detailed description of the dollar command option $on/offLocal.

Another feature of macros is the implicit use of the suffix `.L` in report writing and other data manipulation statements. This allows using the same algebra in model definitions and assignment statements. The following code illustrates this feature:

```
$macro sumIt(i,term) sum(i,term)
cost ..        z  =e=  sumIt((i,j), (c(i,j)*x(i,j))) ;
supply(i) ..   sumIt(j, x(i,j))  =l=  a(i) ;
demand(j) ..   sumIt(i, x(i,j))  =g=  b(j) ;
Model transport /all/ ;
solve transport using lp minimizing z ;
Parameter tsupply(i) total demand for report
          tdemand(j) total demand for report;
$onDotL
tsupply(i)=sumIt(j, x(i,j));
tdemand(j)=sumIt(i, x(i,j));
```

The option $onDotL enables the implicit suffix `.L` for variables. This feature was introduced for macros with variables to be used in equation definitions as well as assignment statements. The matching option $offDotL will disable this feature. Similarly, $offDotScale will access the `.scale` suffix of a variable or equation in an assignment statement.

Three more switches are relevant to macros. The option $show will list any GAMS macros defined. The option $on/$offMacro will enable or disable the expansion of macros; the default is `$onMacro`. Finally, the option $on/offExpand will change the processing of macros appearing in the arguments of a macro call. The default operation is not to expand macros in the arguments. The switch `$onExpand` enables the recognition and expansion of macros in the macro argument list. The option `$offExpand` will restore the default behavior.

Note that macro definitions are preserved in a save/restart file and are available again for a continued compilation.

Summarizing, macros shares the name space of GAMS symbols, like sets, parameters, variables, etc. Macros are recognized and expanded anywhere a proper GAMS identifier may be used. This may be suppressed with the option $on/offMacro. The body of macros is only used during expansion. Hence, macro definitions are not order dependent. Variables in macro bodies will have an implicit suffix `.L` when they are used in assignment statements. This GAMS feature needs to be activated with the option $onDotL.

4.41.6 Compressing and Decompressing Files

GAMS provides two dollar control options for compressing and decompressing GAMS input files:

Dollar Control Option	Description
$compress <source> <target>	The file `source` is compressed into the packed file `target`.
$decompress <source> <target>	The compressed file `source` is decompressed into the unpacked file `target`.

Attention

Spaces are interpreted as separators between the source and target file names, hence quotes (single or double) have to be used if the file names contain spaces.

Note that GAMS will recognize whether a file is compressed and will processes it accordingly.

Note

Like any other GAMS input files, all compressed files are platform-independent.

4.41.6.1 Compressing and Decompressing Files: A Simple Example

We use the well-known transportation model [**TRNSPORT**] to illustrate. First we copy the model from the GAMS Model Library and then we create a compressed version of the original:

```
> gamslib trnsport
> echo $compress trnsport.gms t1.gms > t2.gms
> gams t2
```

Alternatively, the following code snippet may be used from within a GAMS file:

```
$call 'gamslib trnsport'
$compress trnsport.gms t1.gms
$include t1.gms
```

Note that the compressed input file `t1.gms` can be treated like any other GAMS input file. If it is executed, the listing file will be identical to the listing file of the original input file `trnsport.gms`, since a *decompressed* input is reported in the echo print. As usual, the parts of the model that are marked with the dollar control option $on/offListing will not appear in the echo print.

The compressed file `t1.gms` can be decompressed as follows:

```
> echo $decompress t1.gms. t3.gms > t4.gms
> gams t4
```

Alternatively, from within a GAMS file:

```
$decompress t1.gms t3.gms
```

Observe that the decompressed file `t3.gms` is identical to the original file `trnsport.gms`. This can easily be tested with the following command:

```
> diff trnsport.gms t3.gms
```

4.41.6.2 Compressing and Decompressing Files: The Model CEFILES

The following more elaborate example is self-explanatory. It is adapted from model [**CEFILES**] and can easily be modified to test the use of compressed files.

```
* --- get model
$call gamslib -q trnsport

* --- compress and run model
$compress   trnsport.gms t1.gms
$decompress t1.gms       t1.org
$call       diff trnsport.gms t1.org > %system.nullFile%
$if         errorLevel 1 $abort files trnsport and t1 are different

* --- check to see if we get the same result
$call gams trnsport gdx=trnsport lo=%gams.lo%
$if   errorLevel 1 $abort model trnsport failed
$call gams t1       gdx=t1       lo=%gams.lo%
$if   errorLevel 1 $abort model t1 failed
$call gdxdiff trnsport t1 %system.reDirLog%
$if   errorLevel 1 $abort results for trnsport and t1 are not equal

* --- also works with include files
$echo $include t1.gms > t2.gms
$call gams t2 gdx=t2 lo=%gams.lo%
$if   errorLevel 1 $abort model t2 failed
$call gdxdiff trnsport t2 %system.reDirLog%
$if   errorLevel 1 $abort results for trnsport and t2 are not equal
$terminate
```

4.41.7 Encrypting Files

When models are distributed to users other than the original developers, issues of privacy, security, data integrity and ownership arise. To address these concerns, secure work files may be used and GAMS input files may be encrypted. Note, that the encryption follows the work file security model and requires special licensing.

Note

Like any other GAMS input files, all compressed and encrypted files are platform-independent.

Encryption is only available if a system is licensed for secure work files. There are two way to encrypt a file: Encryption for a particular user group that is identified by a target license file or for a general wide audience. In the prior case the key for encryption and decryption is extracted from the target license (specified by command line parameter pLicense) while for the latter the key for encryption and decryption is passed to GAMS via the command line parameters encryptKey and decryptKey. Note that once a file has been encrypted it cannot be decrypted any more. GAMS provides the following dollar control option to encrypt an input file:

```
$encrypt <source> <target>
```

Here the name of the input file to be encrypted is `source` and the name of the resulting encrypted file is `target`.

4.41.7.1 Encrypting Files: A Simple Example

We use again the transportation model [**TRNSPORT**] to illustrate. First we copy the model from the GAMS Model Library and then we create an encrypted version of the original via a target license:

```
> gamslib -q trnsport
> echo $encrypt trnsport.gms t1.gms > t2.gms
> gams t2 pLicense=target lo=%gams.logOption%
```

Note that the first two lines are similar to the directives that we have used to compress the model above. In the third line, the command line parameter pLicense specifies the target or privacy license to be used as a user key for encrypting. Thus the new encrypted file `t1.gms` is locked to the license key `target` and it can only be executed with the license file `target:`

```
> gams t1 license=target dumpOpt=11
```

Note that the command line parameter license is used to override the default GAMS license file `gamslice.txt` that can be located in various system wide and user locations. Note further that the command line parameter dumpOpt is usually used for debugging and maintenance. The value 11 causes a clean copy of the input to be written to the file `t1.dmp`, where all include files and macros are expanded. Observe that if some lines have been marked with the dollar control options $on/offListing in the original file, then these lines will be suppressed in the file `t1.dmp`.

An alternative approach to encrypt GAMS source file without a target license is done by the pair of command line parameters encryptKey and decryptKey:

```
> gamslib -q trnsport
> echo $encrypt trnsport.gms t1.gms > t2.gms
> gams t2 encryptKey=ThisIsAPasswordSomeoneNeedsToRunTheModel lo=%gams.logOption%
```

In the third line, the command line parameter encryptKey specifies the encryption key. Thus the new encrypted file `t1.gms` is locked and requires this key to successfully decrypt via command line parameter decryptKey:

```
> gams t1 decryptKey=ThisIsAPasswordSomeoneNeedsToRunTheModel
```

Anyother key will result in a failure with the message `Integrity check failed`.

Note

> Once a file has been encrypted, it cannot be decrypted any more. There is no inverse mechanism to recover the original file from the encrypted file. An attempt to decompress it using $decompress will fail.

Observe that decrypting is done on the fly into memory when the GAMS system files are read. GAMS will recognize if a file is just plain text or compressed and/or encrypted and will validate and process the files accordingly.

4.41.7.2 Encrypting Files: The Model ENCRYPT

The following more elaborate example is self-explanatory; it is model [**ENCRYPT**] from the GAMS Model Library.

```
$title Input File Encryption Demo (ENCRYPT,SEQ=318)

$onText
Input files can be encrypted with an encryption key. Either the
privacy license file mechanism or the command line parameter pair
encryptKey/decryptKey can be for managing the password. Similar to
compression, we offer an $encrypt utility to lock any file to a
specific key. Once a file has been encrypted it can only be read
by a gams program that has the matching license file or the matching
decryption key. There is no inverse operation possible: you cannot recover
the original GAMS file from the encrypted version.

To create an encrypted file, we need a license file which has the
security option enabled.

Keywords: GAMS language features, input file encryption
$offText

$if not set MYPLICENSE $set MYPLICENSE "%gams.sysdir%plicense.txt"
$ifthen not exist "%MYPLICENSE%"
$   log *** Target license file "%MYPLICENSE%" does not exist.
$   log *** Specify via --MYPLICENSE=...
$   log *** Encryption only via en/decryptKey
$   drop MYPLICENSE
$endif

* --- get model
$onDollar
$call gamslib -q trnsport

$ifthen.MYPLICENSE set MYPLICENSE
* --- encrypt and try to decrypt
$call rm -f t1.gms
$echo $encrypt trnsport.gms t1.gms > s1.gms
$call gams s1 plicense="%MYPLICENSE%" lo=%gams.lo%
$ifE  errorLevel<>0     $abort encryption failed

$eolCom  //
$if not errorFree $abort pending errors
$decompress t1.gms t1.org  // this has to fail
$if     errorFree $abort decompress did not fail
$clearError

*-- execute original and encrypted model
$call gams trnsport gdx=trnsport lo=%gams.lo%
$ifE  errorLevel<>0 $abort model trnsport failed
$call gams t1 license="%MYPLICENSE%" gdx=t1 lo=%gams.lo%
$ifE  errorLevel<>0 $abort model t1 failed
$call gdxdiff trnsport t1 %system.reDirLog%
$ifE  errorLevel<>0 $abort results for trnsport and t1 are not equal

* --- use the encrypted file as an include file
```

```
$onEcho > t2.gms
$offListing
* this is hidden
option limRow = 0, limCol = 0, solPrint = off;
$include t1.gms
$onListing
* this will show
$offEcho
$call gams t2 license="%MYPLICENSE%" lo=%gams.lo%
$ifE  errorLevel<>0 $abort model t2 failed

* --- protect against viewing
*     now we will show how to protect parts of an input
*     file from viewing and extracting original source
*     via the gams DUMPOPT parameter. We just need to
*     encrypt again

* --- encrypt new model
$call rm -f t3.gms
$echo $encrypt t2.gms t3.gms > s1.gms
$call gams s1 plicense="%MYPLICENSE%" lo=%gams.lo%
$ifE errorLevel<>0  $abort encryption failed
$call gams t3 license="%MYPLICENSE%" gdx=t3 dumpopt=19 lo=%gams.lo%
$ifE errorLevel<>0 $abort model t3 failed
$call gdxdiff trnsport t3 %system.reDirLog%
$ifE errorLevel<>0 $abort results for trnsport and t3 are not equal

* --- check for hidden output
$call grep "this is hidden" t3.lst > %system.nullfile%
$if not errorLevel 1 $abort did not hide in listing
$call grep "this is hidden" t3.dmp > %system.nullfile%
$if not errorLevel 1 $abort did not hide in dump file
$endif.MYPLICENSE

* Same with encryptKey/decryptKey
* --- encrypt and try to decrypt
$call rm -f t1.gms
$echo $encrypt trnsport.gms t1.gms > s1.gms
$call gams s1 encryptKey=ThisIsMyPassword lo=%gams.lo%
$ifE  errorLevel<>0     $abort encryption failed

$eolCom  //
$if not errorFree $abort pending errors
$decompress t1.gms t1.org  // this has to fail
$if     errorFree $abort decompress did not fail
$clearError

*-- execute original and encrypted model
$call gams trnsport gdx=trnsport lo=%gams.lo%
$ifE  errorLevel<>0 $abort model trnsport failed
$call gams t1 decryptKey=ThisIsMyPassword gdx=t1 lo=%gams.lo%
$ifE  errorLevel<>0 $abort model t1 failed
$call gdxdiff trnsport t1 %system.reDirLog%
$ifE  errorLevel<>0 $abort results for trnsport and t1 are not equal

* --- use the encrypted file as an include file
$onEcho > t2.gms
$offListing
```

```
* this is hidden
option limRow = 0, limCol = 0, solPrint = off;
$include t1.gms
$onListing
* this will show
$offEcho
$call gams t2 decryptKey=ThisIsMyPassword lo=%gams.lo%
$ifE  errorLevel<>0 $abort model t2 failed

* --- protect against viewing
*     now we will show how to protect parts of an input
*     file from viewing and extracting original source
*     via the gams DUMPOPT parameter. We just need to
*     encrypt again

* --- encrypt new model
$call rm -f t3.gms
$echo $encrypt t2.gms t3.gms > s1.gms
$call gams s1 encryptKey=ThisIsMyPassword lo=%gams.lo%
$ifE errorLevel<>0  $abort encryption failed
$call gams t3 decryptKey=ThisIsMyPassword gdx=t3 dumpopt=19 lo=%gams.lo%
$ifE errorLevel<>0 $abort model t3 failed
$call gdxdiff trnsport t3 %system.reDirLog%
$ifE errorLevel<>0 $abort results for trnsport and t3 are not equal

* --- check for hidden output
$call grep "this is hidden" t3.lst > %system.nullfile%
$if not errorLevel 1 $abort did not hide in listing
$call grep "this is hidden" t3.dmp > %system.nullfile%
$if not errorLevel 1 $abort did not hide in dump file
```

4.42 The Put Writing Facility

4.42.1 Introduction

While the GDX facility is widely used to exchange bulk GAMS data with other programs the put writing facility allows to generate sophisticated reports in GAMS. The result are external ASCII files that are structured using information that is stored by the GAMS system. The put writing facility offers users numerous ways to control three format layers: the format of the external file that is written to, the format of the pages of the external files and the format of output items. Hence the structure of the put writing facility is more complex and requires more programming than the display statement, but there is much more flexibility and control over the output. The put writing facility generates external files automatically when the GAMS program is executed. The files are written sequentially, a single page at a time. The current page is stored in a buffer, which is automatically written to an external file when the page is full. Thus, the put writing facility has only control over the current page and does not have the ability to go back to alter former pages. However, while a particular page is current, information placed on it may be overwritten or removed at will.

This chapter is organized as follows. We will first introduce the file statement and the put statement, which are at the core of the put writing facility. Then we will present a simple example that will serve as illustration. We will also introduce some widely used features as we comment on the example. The remainder on the chapter will cover in detail the external report files, also called put files, the structure of put file pages and ways to control their format, types of output items and their formatting controls, and the put_utility statement, a variant of the put statement that allows to special communication of formatted elements with the outside

world. In addition, we will briefly discuss exception handling in the context of put statements and GAMS errors that are specific to put statements. We will conclude the chapter with an elaborate example. Note that put file attributes play a crucial role, since they are used for most formatting controls. A complete list of all file attributes is given in section Put File Attributes.

4.42.2 The Syntax

The basic structure of the put writing facility in its simplest form is as follows:

```
File file_name {,file_name};
put file_name;
put item {,item};
```

Note that the first line is a file statement. File statements define one or more external files that will be written to and specify internal names for them. These internal names wil be used in the GAMS model to reference the external files when they are written to. The second line is a put statement that assigns the file with the name `file_name` as the current file. The third line is a put statement that writes one or more items to the current file. Items are any type of output like explanatory text, set labels, parameters, and variable, equation values and model attributes.

Next, we will present more details on the file statement and the put statement in the following two subsections and then we will turn to a first example that will illustrate how the the put writing facility works.

4.42.2.1 The File Statement: Defining Put Files

External files that are written to with the put writing facility are called *put files*. They are defined with a file statement. The syntax for a file statement is as follows:

```
File[s] file_name ["text"] [external_file_name]
  {,file_name ["text"] [external_file_name] } ;
```

The keyword `file` or `files` indicates that this is a file statement. It is follwed by the internal name for the put file, `file_name`. The internal file name is a handle for the put file, it is used in the GAMS model to refer to the put file. The optional explanatory text may be used to describe the put file for future reference and to ease readability. The final part of the file statement is the actual name of the put file. Naming the external file is optional. In case it is omitted, by default, GAMS will create a name by appending the extension `.put` to the internal name. Consider the following example:

```
File results;
```

Note that in this statement no external file name is specified. Thus GAMS will create the external file `results.put` in the current working directory. Observe that by default, all put files are stored in the current working directory. There are several ways to specify alternative directories for put files. For details see section Choosing Where Put Files are Saved below.

Note

Multiple put files may defined with one file statement.

Consider the following example:

```
File  class1
      class2   "this defines a specific external file"  /report.txt/
      log      "this defines access to the GAMS log file" /''/;
```

Observe that the internal name of the first file is `class1`. As no external name is specified, GAMS will assign the absolute name `class1.put` to this file. The second file will be referenced in the model with the name `class2` and it corresponds to the external file `report.txt`. The third file is special: the internal name `''` is reserved for writing output to the GAMS log. Note that writing to the GAMS log can be useful to monitor how the solution process of the model is progressing.

For further details on put files, see section Put Files below.

4.42.2.2 The Put Statement

The put statement is at the core of the put writing facility. It has two different functions: it specifies which of the previously defined put files is the current file and it writes output to that file. The syntax for the first function is simple:

```
put file_name;
```

The keyword `put` indicates that this is a put statement, `file_name` is the internal name of a put file that was previously defined with a file statement. This put statement has the effect that the specified file is now ready to be written to. Note that this put statement is necessary even if only one put file has been previously defined. For an example of how it is used when output is written to several different files, see the next section.

The basic syntax for a put statement that is used to write output to a file is as follows:

```
put item {,item};
```

The statement begins with the keyword `put` followed by one or more items. Items may be a text (like a quoted text, an explanatory text, the name of a set element), a numerical value (like the value of a parameter, the value of an attribute, the solution status of the model) or a set value (`YES` or `NO` indicating whether a label is an element of a specific set). These items are discussed in detail in section Output Items below. In addition, GAMS facilitates writing a *text block* to a put file and including the content of an external file in a put file. For details see sections Text Items: Text Blocks and The Put_Utility Statement respectively.

Note that it is also allowed to use only one put statement to assign one file as the current file *and* write to it. The syntax follows:

```
put file_name item {,item};
```

In addition, it is permitted to use just one put statement to write to multiple files sequentially. Thus the most general form of a put statement is as follows:

```
put file_name item {,item} {,file_name item {,item}};
```

Note that only one file is current at a time. When a file is current, the output items following the name of this file will be written to the file. After this has been completed, the current file is reassigned to the next internal file name in the statement. The last internal file name used in a put statement continues to be the current file until a subsequent put statement uses an internal file name.

Observe that the keyword `put` has several variants: putclose, puttl, puthd, putpage and put_utility.

4.42.3 A First Example

We will use a small example to introduce the basic features of the put writing facility. The example is based on the well-known transportation model [**TRNSPORT**]. The following code segment could be placed at the end of the transportation model to create a report:

```
File factors /factors.dat/,
     results /results.dat/;
put factors;
put 'Transportation Model Factors' / /
    'Freight cost  ', f,
    @1#6, 'Plant capacity'/;
loop(i, put @3, i.tl, @15, a(i)/);
put /'Market demand'/;
loop(j, put @3, j.tl, @15, b(j)/);

put results;
put 'Transportation Model Results' / / ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);
```

In the first line, the file statement defines the internal file names `factors` and `results` and connects them to the external files `factors.dat` and `results.dat` respectively. These internal file names are used inside the model to reference files which are external to the model.

In the second line, the put statement assigns the file `factors` as the current file, that is the file which is currently available to be written to.

In the third line, the put statement starts the actual writing to the put file. The first item that is written is the quoted text string `'Transportation Model Factors'`. The item is followed by two slashes. A slash instructs the cursor to move to the first column of the next line. Two slashes have the effect that the cursor is moved to the first column of the second line, thus introducing a blank line.

Attention

> Two slashes `//` are a popular end of line comment character sequence. So an intended blank line can result in a comment in the GAMS code. A safe way to use slashes in put statement is to separate the slashes by a space.

The next item is the string `'Freight cost '` followed by the value of the scalar `f`. Note that these output items are separated by commas. Blanks and commas serve as delimiters for separating different output items. These delimiters leave the cursor at the next column position in the file following the last item written. In most cases, a blank and a comma may be used interchangeably. However, the comma is the stronger form and will eliminate any ambiguities.

In the fifth line, the code above starts with the instruction `@1#6`. In the context of put statements, the symbols `@` and `#` serve to reposition the cursor to a specific column and row respectively. Thus in our case, the cursor is repositioned to column 1 of row 6 of the put file. Then another text string is written and a new line is started. The semicolon terminates the put statement. More details on cursor positioning are given in section Controlling the Cursor On a Page.

In line 6, the next put statement is embedded within a loop statement. Note that the set `i` is the looping set. The put statement writes at column position 3 the set label name and at column position 15 the value of the parameter `a(i)` for each element of the set `i`. Observe that set element labels are referenced with the name of the set and the suffix `.te`. For more information on identifier suffixes, see section Text Items below. Note that the put statement has to be placed within a looping structure, since only one element of the index set may be written with a put statement.

In line 7, the first symbol after the keyword `put` is a slash, that has the effect that a blank line is inserted before the text string `'Market demand'` is written.

In line 8, we have again a put statement within a loop structure: the values of the parameter `b` are written in a similar way to those of parameter `a` in line 6.

After execution, the put file `factors.dat` will look as follows:

```
Transportation Model Factors

Freight cost          90.00

Plant capacity
  Seattle            350.00
  San-Diego          600.00

Market demand
  New-York           325.00
  Chicago            300.00
  Topeka             275.00
```

Note that this output has been formatted using the default file format values. GAMS offers several ways to customize the format, see sections Global Item Formatting Controls and Local Item Formatting Controls below for further information.

In the last three lines of the code above, the file `results.dat` is made current and the level values associated with the variable `x` along with their corresponding set element index labels are written line by line. Note that the format of the output results of the variable `x` is customized by specifying a field width of 8 spaces with 4 of these spaces reserved for decimal places. This is an example of local formatting. The put file `results.dat` will contain the following lines:

```
Transportation Model Results

Seattle    New-York     50.0000
Seattle    Chicago     300.0000
Seattle    Topeka        0.0000
San-Diego  New-York    275.0000
San-Diego  Chicago       0.0000
San-Diego  Topeka      275.0000
```

This small example has demonstrated the main features of the put writing facility. However, its surface has just barely been scratched. In the remainder of this chapter we will describe in detail the many features of the put writing facility. Observe that in section Creating a Report for the Model MEXSS at the end of this chapter we will present a more elaborate report.

4.42.4 Put Files

As mentuioned earlier, external files that are written to with the put writing facility are called *put files*. They are defined with a file statement and are made current with a put statement. Once they are current, they may be written to. By default, put files are saved in the current working directory. In this section we will cover more details on put files. We will discuss ways to specify other directories for put files, introduce the putclose statement, a variant of the put statement that closes the current file, and we will show how to append to an existing external file instead of replacing (overwriting) it. In addition, we will introduce the notion of put file attributes including a list of all attributes.

4.42.4.1 Choosing Where Put Files are Saved

Recall that by default, put files are saved in the current working directory. GAMS offers several ways to specify other directories for put files. The easiest way to specify another directory is by including the absolute or relative path in the file statement. Consider the following example:

```
File report / C:\Documents\GAMS\Output\report.dat /;
```

Note that the put file `report.dat` with the internal name `report` will be stored in the directory specified in the file statement instead of the current working directory.

An alternative directory for *all* put files in a model may be specified with the command line parameter PutDir. Assume we wish to write some ouput from running the well-known transportation model [**TRNSPORT**] to the file `results` defined above. Consider the following GAMS call:

```
> gams trnsport PutDir=c:\output
```

Note that the file `results.put` will be stored in the directory specified in the GAMS call. Observe that storing put files in the directory specified with `PutDir` is applies to all file that are not specified via an absolute file name. For more information on command line parameters, see chapter The GAMS Call and Command Line Parameters. scratchdir as putdir option (`.pdir`) lets GAMS write put files into the scratch directory.

The content of a put file will be sent to the GAMS log file if the put file is specified in the following way:

```
File name / '' /;
```

For example, the following code snippet may be added to the transportation model [**TRNSPORT**]:

```
File name / '' /;
put name ;
put 'instructions that will go to the GAMS log' /;
put 'more instructions that will go to the GAMS log' /
```

As a result, the respective GAMS log will contain the following lines:

```
...
instructions that will go to the log file
more instructions that will go to the log file
*** Status: Normal completion
--- Job trnsport.gms Stop 01/25/17 15:46:09 elapsed 0:00:00.320
```

4.42.4.2 The Putclose Statement: Closing a Put File

Recall that the file statement and the put statement are at the core of the put writing facility. The putclose statement is a variant of the put statement, it is used to close a put file. The syntax is as follows:

```
putclose [file_name]  {item} {file_name item {item}};
```

If the `file_name` is missing, `putclose` will close the current file. Often the putclose statement is used in this simple form. Consider the following example:

```
File report;
put report;
put "This is a report."
putclose;
```

Note that the last line has the effect that the file `report.put` is closed. The file may be used again later. As usual, it has to be assigned as the current file and then it may be written to. By default, an existing file is overwritten (replaced). Alternatively, it may be appended to. For details see section Writing to an Existing Put File below. The briefest version of the above code is

```
File report; putclose report "This is a report."
```

Closing a put file is useful for example when writing a solver option file from within the GAMS model. Consider the following example, where we will create and close an option file for the solver MINOS.

```
File opt 'Minos option file'  / minos.opt /;
put opt 'Iteration limit          500'/
        'Feasibility tolerance    1.0E-7'/ ;
putclose opt;
```

Note that the file `minos.opt` is closed with a simple putclose statement. This will make the file available to be used by the solver. Observe that the code snippet above has to be placed *before* the respective solve statement.

Now, the last three lines in the example above may be reduced to two lines in the following way:

```
putclose opt 'Iteration limit          500'/
             'Feasibility tolerance    1.0E-7'/ ;
```

In this formulation, the putclose statement makes the file `minos.opt` current, writes to it and then closes the file after the last item has been written. Even though this is shorter, many users prefer the first formulation, since it is clearer.

Similarly to the put statement, the putclose statement may also be used to write to several files subsequently. The only difference to the put statement is, that the last current file will be closed after is has been written to.

Observe that GAMS automatically closes the put files of a model when it exits, even without a putclose statement.

4.42.4.3 Writing to an Existing Put File

When the put writing facility is used to write to a file that is not empty, by default, the existing content is overwritten (replaced). However, GAMS provides the option to append to the file instead via the file attribute append option (`.ap`). Consider the following example:

```
File append /append.dat/
put append;
put "This is the first line." /;
putclose;
append.ap = 1;
put append;
put "This is the second line.";
```

Note that the putclose statement in line 4 closes the file `append.dat`. The assignment in the next line instructs the put writing facility to append to the file if the file is opened again and written to. Thus the following output is generated:

```
This is the first line.
This is the second line
```

For information on file attributes in general, see section Put File Attributes below.

4.42.4.4 Put File Attributes

Put files have attributes that are mainly used to customize the format of put files, put file pages and output items. Put file attributes are accessed in the

following way:

```
file_name.attribute
```

Here `file_name` is the internal name of the put file and `.attribute` is the specific attribute that is to be accessed. We can also access the specific attribute of the current file by `file.attribute`. This useful in particular in batincludes that are used for multiple put files. Put file attributes may be used on the left-hand side and right-hand side of assignments. Consider the following example:

```
factors.nw = 10;
scalar x; x = factors.nw;
```

In the first line, the attribute numeric field width (`.nw`) of the put file with the internal name `factors` is assigned the value of 10. In the second line, the value of the attribute `.nw` is assigned to a scalar parameter `x`. In addition, put file attributes may be used as output items. For an example, see section Controlling the Cursor with File Attributes below. A complete list of put file attributes is given in Table 1.

Table 1: Put File Attributes

Put File Attribute	Symbol	Description	Default Value	Optional Values
Append option	.ap	Allows to append to an existing file, instead of replacing (overwriting) it. For more information, see section Writing to an Existing Put File.	0	0: Overwrite 1: Append
Bottom margin	.bm	Number of blank lines to be placed in the bottom margin of the page. The bottom margin lines are in adition to the lines specified with the file attribute page size (.ps). Note that this attribute is functional only if the value of the suffix print control (.pc) is zero.	0	
Alphabetic case	.case	Specifies the case in which alphabetic characters are displayed in the put file, regardless of the input. Note that the meaning of 1 and 2 differs from the one for lcase.	0	0: Mixed case 1: Upper case 2: Lower case
Current column	.cc	This attribute may be used to return or set the current cursor column in the main window. For more information, see Current Cursor Column.	0	Value may be between 1 and the page width.
Current row	.cr	This attribute may be used to return or set the current cursor row in the main window. For more information, see Current Cursor Row.	1	Value may be between 1 and the page size minus any header rows, title rows and margins.
Number of put errors	.errors	See section Put Errors for details and an example.	0	
Header current column	.hdcc	This attribute may be used to return or set the current cursor column in the header block. For more information, see Current Cursor Column.	0	Value may be between 1 and the page width.

Put File Attribute	Symbol	Description	Default Value	Optional Values
Header current row	`.hdcr`	This attribute may be used to return or set the current cursor row in the header block. For more information, see Current Cursor Row.	1	Value may be between 1 and the size of the header.
Header last line	`.hdll`	This attribute may be used to return the number of the last line of the page header or reset the last row written in the page header. For more information, see Last Line Control.	0	
Alphabetic label case	`.lcase`	Specifies the case in which alphabetic characters of a label are displayed in the put file, regardless of the original casing.	0	0: Mixed case 1: Lower case 2: Upper case
Label justification	`.lj`	Alignment of set labels.	2	1: Right, 2: Left, 3: Center
Last line	`.ll`	This attribute may be used to return the number of the last line of the main window or reset the last row written in the main window. For more information, see Last Line Control.	0	
Left margin	`.lm`	Number of empty columns to be placed on the left of the page.	0	
Last page	`.lp`	Returns the number of pages that are already in the put file. Note that setting this attribute to zero does *not* delete the pages that have been written to the file.		
Label field width	`.lw`	Field width of set label output. For more information, see section Global Item Formatting Controls.	12	Values may be 0 or larger.

Put File Attribute	Symbol	Description	Default Value	Optional Values
Number of decimals	.nd	Sets the number of decimals that are displayed for numeric items. A value of zero entails that fixed-format output will show the number rounded to the nearest integer. For an example, see section Global Item Formatting Controls.	2	Values may be between 0 and 20.
Numeric justification	.nj	Alignment of numeric output.	1	1: Right, 2: Left, 3: Center
Numeric round format	.nr	Selects the formatting used for numeric output. For example, one can choose to use scientific notation (i.e. E-format) for all numbers: this is especially useful for numbers so small that they display as only zeros in fixed format with the specified suffix number of decimals (.nd). For more information and an example, see section Global Item Formatting Controls.	1	0: Item is displayed in F or E format to fit given width and decimals. 1: Item is rounded to fit given width and decimals. 2: Item is displayed in E format to fit given width and decimals. 3: Item is rounded to fit given width. 4: Item is displayed in F or E format to fit given width.
Numeric field width	.nw	Field width of numeric output. For more information, see section Global Item Formatting Controls.	12	Values may be 0 or larger.
Numeric zero tolerance	.nz	Sets the tolerance level for which a number will be rounded to zero for display purposes. Note that in case this attribute is set to zero, rounding will be determined by the field width.	1.0e-10	
PutDir becomes scratch directory	.pdir	Setting this to nonzero will result in resetting the PutDir to the scratch directory. Since the scratch directory is unknown while parameters are provided, this allows to use the scratch directory as PutDir.	0	

Put File Attribute	Symbol	Description	Default Value	Optional Values
Print control	.pc	Specifies the format of the put file. The options 4, 5, 6 and 8 create delimited files, which are especially useful when preparing output for the direct importation into other computer programs such as spreadsheets. For an example, see section Creating a Report for the Model MEXSS.	2	0: Standard paging based on the current page size. Partial pages are padded with blank lines. Note that the file suffix bottom margin (.bm) is only functional when used with this print control option. 1: FORTRAN page format. This option places the numeral one in the first column of the first row of each page in the standard FORTRAN convention. 2: Continuous page. This option is similar to .pc option zero, with the exception that partial pages in the file are not padded with blank lines to fill out the page. 3: ASCII page control characters inserted. 4: Formatted output; non-numeric output is quoted and each item is delimited with a blank space. 5: Formatted output; non-numeric output is quoted and each item is delimited with commas. 6: Formatted output; non-numeric output is quoted and each item is delimited with tabs. 7: Fixed width, fills up line with trailing blanks. 8: Formatted output; each item is delimited with a blank space.

Put File Attribute	Symbol	Description	Default Value	Optional Values
Page size	.ps	Number of rows (lines) that may be placed on a page of the put file. It may be reset by the user at any place in the program.Observe that the specification of this attribute is only effective if the attribute print control (.pc) has a value other than its default value. Note that a put error will result if it is set to value that is smaller than the number of rows which have already been written to the current page.	60	Any value between 1 and 130.
Page width	.pw	Number of columns (characters) that may be placed on a single row of the page. It may be reset by the user at any place in the program. Note that a put error will result if it is set to value that is smaller than the number of columns which have already been written to the current page. Observe that if a value is specified that is above the upper limit, then the value will be reset to the upper limit.	32767	Any value between 1 and 32767.
Set value justification	.sj	Alignment of set values.	1	1: Right, 2: Left, 3: Center
Set value field width	.sw	Field width of set values. For more information, see section Global Item Formatting Controls.	12	Values may be between 0 and 20.

Put File Attribute	Symbol	Description	Default Value	Optional Values
Text fill	.tf	Controls what content will be displayed, if there is no explanatory text for a set element, but a text item with suffix .te was specified. Note that options 4, 5 and 6 are useful if the output is intended to be included in a GAMS model at a later time.	2	0: Blanks are displayed. 1: Blanks will be displayed if the specified element does not exist, otherwise the name of the element will be displayed. 2: The name of the element is displayed. 3: The name of the element is displayed even if an explanatory text exists. 4: Like option 3, but displays output in quotes with comma separators. 5: Like option 4, but with periods as separators. 6: Like option 4, but with blanks as separators.
Text justification	.tj	Alignment of quoted text and explanatory text.	2	1: Right, 2: Left, 3: Center
Title current column	.tlcc	This attribute may be used to return or set the current cursor column in the title block. For more information, see Current Cursor Column.	0	Value may be between 1 and the page width.
Title current row	.tlcr	This attribute may be used to return or set the current cursor row in the title block. For more information, see Current Cursor Row.	1	Value may be between 1 and the size of the header.
Title last line	.tlll	This attribute may be used to return the number of the last line of the page title or reset the last row written in the page title. For more information, see Last Line Control.	0	

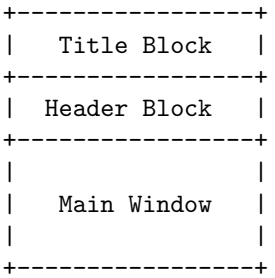
Put File Attribute	Symbol	Description	Default Value	Optional Values
Top margin	.tm	Number of blank lines to be placed at the top margin of the page. These lines are in addition to the number of lines specified with the file attribute page size (.ps).	0	
Text field width	.tw	Field width for quoted text and explanatory text. Note that a value of zero forces the field width to match the exact size of the item being displayed. For more information, see section Global Item Formatting Controls.	0	Value may be 0 or larger.
Window size	.ws	Returns the number of rows that may be written to the main window given the number of rows in the title and header blocks and the page size. Note that this attribute is computed by GAMS and cannot be changed by the user.		

4.42.5 Put File Pages

In this section we will discuss the structure of put file pages, in particular how to write titles and headers. In addition, we will give details on how users may control many aspects of the format of a put file page.

4.42.5.1 Adding Titles and Headers

Titles and headers are widely used when there are sections of a page that remain relatively constant throughout a document. There are three independent writing areas on each page of a put file: the title block, the header block and the main window. The layout of a page is shown in the following diagram.



Note that the put statement writes to the main window. There are two variants of the put statement that write to the title block and header block respectively: the puttl statement and the puthd statement.

Observe that every page must have an entry in the main window. If a page has no output in its main window, the page will not written, regardless of whether there are output items in the title or header blocks. To force a page that has an empty window out to file, we recommend to simply write something innocuous to the window, like the following:

```
put ''; // space
```

This will initiate the main window of the page and thus the page will be written.

Further, note that the size of any area of a page is based on the number of lines written to it and the total number of lines has to be smaller than the specified page size. A new page is started automatically whenever a page is full. For details on *manual* pagination, see section The Putpage Statement: Controlling Paging. Each area of a page is maintained independently. For example, we may write to the title block first, then to the header block and then again to the title block. However once we have written to the main window, all subsequent entries to the title and header are written to the *next* page, not the current page.

The Puttl Statement: Writing to the Title Block

The puttl (or `putTitle` which can be used as a synonym) statement is a variant of the put statement that writes to the title block of a page. The syntax is as follows:

```
puttl [file_name] item {item};
```

The keyword `puttl` indicates that this is a puttl statement. The internal name of the file, `file_name`, may be omitted if the desired file is already current. Like in the put statement, the content that is written is one or more `items`, that may be a text string, a numerical value or a set value. For more information on items, see section Output Items below. Consider the following example:

```
puttl factors 'GAMS Put Example';
```

If this line is placed *before* the put statement that writes to the main window in the example above, then `GAMS Put Example` will be written to the title block.

Observe that title blocks often contain the name of the model and the number of the put page. Once content is placed in the title block, it will be repeated on every page unless it is modified or deleted. For details on how to write output like the page number to a put file, see section System Suffixes as Output Items below.

Note

> If content has already been written to the main window of a page, the items in the puttl statement will appear in the title block starting from the *next* page. Thus the title has to be written *before* any entries in the main window if it is to appear on the current page.

Observe that if the title and header blocks contain too many lines given the page size, the page will overflow resulting in a put error.

The Puthd Statement: Writing to the Header Block

Like the puttl statement, the puthd (or `putHeader` which can be used as a synonym) statement is a variant of the put statement. Puthd statements write to the header block of a page. The syntax is as follows:

```
puthd [file_name] item {item};
```

The keyword `puthd` indicates that this is a puthd statement. The internal name of the file, `file_name`, may be omitted if the desired file is already current. Like in the put statement, the content that is written is one or more `items`. See section Output Items below for more information on items.

Observe that title blocks often contain a disclaimer or an instruction that is meant to be repeated on every page. Once content is placed in the header block, it will be repeated on every page unless it is modified or deleted.

Note

> If content has already been written to the main window of a page, the items in the puthd statement will appear in the header block starting from the *next* page. Thus the header has to be written *before* any entries in the main window if it is to appear on the current page.

Observe that if the title and header blocks contain too many lines given the page size, the page will overflow resulting in a put error.

The Putclear Statement: Deleting Title and Header Blocks

The `putclear` (or `putfmcl` which can be used as a synonym) statement may be used to delete the title and header blocks. The syntax is simple:

```
putclear;
```

If the keyword `putclear` is inserted after a title and/or header block has been written, but *before* the main window has been written to, then the title and/or header blocks of the *current* page will be deleted. If the main window has already been written to, the the title and/or header block of the *next* and all subsequent pages will be deleted. Note that if the user wishes to delete either only the title or only the header block, the put file attributes .tlll and .hdll respectively may be set to zero. For more information, see Last Line Control.

4.42.5.2 The Putpage Statement: Controlling Paging

A new page is started automatically if the bottom of a page is reached. Alternatively, a new page may be started early using a variant of the put statement: the `putpage` statement. The syntax is as follows:

```
putpage [file_name] {item};
```

The keyword `putpage` indicates that this is a putpage statement. In its simplest form, the putpage statement consists of just the keyword. If only the keyword is used, the current page will be terminated. The optional `file_name` makes the put file with the internal name `file_name` current. If output items follow, they will be written to the current page and a new page will be made available for the next put statement.

Consider the following example:

```
File example / example.txt /;
putpage example "This text is placed in the main window and the page ends here.";
put "Here starts a new page in the same putfile."
```

Observe that three file attributes are helpful for manual paging: last page (`.lp`), window size (`.ws`) and last line (`.ll`).

4.42.5.3 Controlling the Format of a Put File Page

While GAMS established gracious defaults, there are numerous ways to customize the format of a put file page. In this section we will present the put file attributes that facilitate the control of various aspects of page formatting and ways to control the position of the cursor.

Controlling Page Size, Page Width and Margins

By default, 60 rows (lines) may be written on a put file page. At any point in the program the file attribute page size (`.ps`) may be used to reset this value. Note that the upper limit is 130. Note further, that a put error will result if the attribute is set to value that is smaller than the number of rows which have already been written to the current page. Observe that the specification of `.ps` is only effective if the attribute print control (`.pc`) has a value other than its default value.

By default, 255 characters (colums) may be written to each line of a put file page. Like the page size, the page width may be reset at any point. GAMS provides the file attribute page width (`.pw`) to do this. Note that the upper limit is 32767. Note further, that a put error will result if the attribute is set to value that is smaller than the number of columns which have already been written to the current page.

By default, put file pages do not have top, bottom and left margins. However, it is easy to specify blank lines for the margins with the file attributes top margin (`.tm`) and bottom margin (`.bm`). Note that the lines reserved for the margins are in addition to the value specified for the page size (`.ps`). Observe that the file attribute `.bm` is only functional if the value of the file attribute print control (`.pc`) is set to zero. In addition, the number of empty columns for left margins may be specified with the file attribute left margin (`.lm`)

Controlling the Print Format and the Use of Capital and Lower Case Letters

By default, GAMS prints continuous pages. However, the file attribute print control (`.pc`) offers several alternative options. For a full list, see the respective entry in Table Put File Attributes above. Note in particular that the options 4, 5, 6 and 8 create delimited files, which are useful for importing output to other applications like spreadsheets. For an example, see section Creating a Report for the Model MEXSS.

By default, alphabetic characters are displayed in the case they were inputted. Note that the value of 1 for the file attribute alphabetic case (`.case`) will display all output in capital letters, regardless of the input, and the value of 2 will result in only lower case letters being displayed.

Assume that we wish a put file `report.txt` to have pages with 72 characters to a row and 58 lines, ASCII page control characters to be inserted at the end of every page, an additional top margin of 6 lines and all output displayed in upper case. The following assignments will implement these specifications:

```
File report /report.txt/;
report.pw = 72; report.ps = 58; report.pc = 3;
report.tm = 6;  report.case = 1;
```

Controlling the Cursor On a Page

There are three ways to control the position of the cursor: with cursor control characters, with the system attribute `system.tab` and with file attributes. In this section we will give details on cursor control characters, while inserting tabs and file attributes that are relevant for cursor control will be discussed in the next two sections.

By default, the cursor is moved to the space immediately following the last character written. GAMS provides the control characters listed in Table 2 to specify another position for the cursor.

Symbol	Description
`#n`	Move cursor position to row `n` of current page.
`@n`	Move cursor position to column `n` of current line.
`/`	Move cursor to first column of next line.

Table 2: Cursor Control Characters

Note that the numeric value `n` that follows the characters `#` and `@` may be any expression or symbol with a numeric value. An example is given below. Note further, that the character `@` is commonly used to align columns if the column headings have different widths. For an example, see section Creating a Report for the Model MEXSS. Observe that the character `@` may be used to overwrite output items that were written earlier. Consider the following example:

```
File out /out.dat/;
put out;
put 'Good' @3 'morning everybody!';
```

This code snippet will generate the following output:

```
Gomorning everybody!
```

Observe that cursor control characters are also used and discussed in the example at the beginning of this chapter.

Controlling the Cursor with a System Attribute: Inserting Tabs

The system attribute `system.tab` may be used to insert tabs. Consider the following example:

```
File test / test.dat /;
put test;
put "1" system.tab "2" system.tab "3" ;
```

The put file will contain the following line, separating 1, 2, and 3 by tabs.

```
1    2    3
```

Controlling the Cursor with File Attributes

In addition to cursor control characters and tabs, the position of the cursor may be controlled using the file attributes that will be discussed in this section. These file attributes refer to the current column, the current row and the last line in the title block, the header block and the main window.

Current Cursor Column

As there are three independent writing areas on each put file page, there are three file attributes that refer to the current column. They are listed in Table 3.

Symbol	Description
.cc	Current cursor column in the *main window*.
.hdcc	Current cursor column in the *header block*
.tlcc	Current cursor column in the *title block*

Table 3: File Attributes for the Column Position of the Cursor

Note that the values for these attributes are numeric and they are updated only at the *end* of a put statement. Consequently, their values will remain constant during a put statement, even if multiple items or lines are displayed. Observe that the attributes may be used on the left-hand side and on the right-hand side of an assignment statement to set the current column position and return it respectively.

The following example illustrates the updating of the cursor control suffixes and the use of cursor control characters. The example is trivial but instructive:

```
Scalar  lmarg  'left margin'   /6/;
File out; put out;
put @(lmarg+2) 'out.cc = ', out.cc:0:0 '  ';
put @out.cc 'x'/ @out.cc 'y'/ @out.cc 'z  ';
put 'out.cc = ' out.cc:0:0;
```

The resulting file `out.put` will contain the following lines:

```
out.cc = 1  x
            y
            z  out.cc = 23
```

Note that the scalar `lmarg` is set to a specific value that will be used as an alignment tab. Symbols which hold common alignment values such as margins or tabs are often useful for large structured documents. The first put statement uses the cursor control character `@` to relocate the cursor to column 8 where the text item `out.cc` and the respective value for the file attribute `.cc` are displayed. Note that the numerical value of the file attribute is formatted locally with the effect that only integers are displayed. We observe that at the start of the first put statement the file attribute `out.cc` equals 1.

The second put statement illustrates the updating of the cursor control suffixes by writing the letters `x`, `y` and `z` on three different lines. Each is preceded by moving the cursor to the value of the file attribute `.cc`. Note that at the end of the first put statement the value of `out.cc` is updated to 20. Hence, `out.cc` is 20 at the start of the second put statement. As a single put statement is used to write all three letters, the value of `out.cc` remains constant and thus the letters are written in the same column. At the end of the second put statement the value of `out.cc` value is updated to 23 (observe that there are two blank spaces after the letter `z`).

The third put statement writes the current value of the file attribute `out.cc`, which is the value at the start of the put statement.

Current Cursor Row

As there are three independent writing areas on each put file page, there are three file attributes that refer to the current row. They are listed in Table 4.

Symbol	Description
.cr	Current cursor row in the *main window*
.hdcr	Current cursor row in the *header block*
.tlcr	Current cursor row in the *title block*

Table 4: File Attributes for the Row Position of the Cursor

Note that the values for these attributes are numeric and they updated only at the *end* of a put statement. Consequently, their values will remain constant during a put statement regardless of how many rows are displayed. Observe that the attributes may be used on the left-hand side and on the right-hand side of an assignment statement to set the current row and return it respectively. The files attributes for the row postion of the cursor behave similarly to those for the column position.

Last Line Control

Like the file attributes for the current column and row position, the file attributes for the last line have three variants, one for each writing area. They are given in Table 5.

Symbol	Description
`.ll`	Last line in the *main window*
`.hdll`	Last line in the *header block*
`.tlll`	Last line in the *title block*

Table 5: File Attributes for the Last Line

Note that unlike the row and column controls, the last line attributes are updated continuously. Last line attributes are especially useful for modifying the three writing areas of a page. They may be used on the left-hand side and on the right-hand side of an assignment statement to set the current last line and return it respectively. Rows will be deleted if the last line attribute is set to a value that is lower than the current row.

Attention

> The file attributes `.tlll` and `.hdll` may not hold values applicable to the current page, because when the title or header blocks are modified, they will correspond to the title or header blocks of the *next* page if the main window has been written to on the current page.

As mentioned above, in addition to determining the last line in a writing area, these file attributes may be used to delete lines within a writing area. In the following example, the header block will be completely deleted by resetting the attribute `.hdll` to zero.

```
File out;
puthd out 'This header statement will be eliminated';
out.hdll = 0;
```

Note that a header is written initially. However, by changing the attribute `.hdll` to zero, the cursor is reset to the top of the header block. As a result, the header will not be written unless something new is added to the header block.

4.42.6 Output Items

Output items are the items in the put statement that are written to the put file. They may be a text, a numerical value or a set value. In this section we will provide more information on these three types of items. Observe that system suffixes and command line parameters are either text strings or have a numerical value. They may also be used as output items.

4.42.6.1 Text Items

Text items may be a quoted text, an explanatory text of identifiers and labels or names of set elements. In this section we will give details on each of these text items. In addition, text items may be system attributes or command line parameters. For these text items, see sections System Suffixes as Output Items and Command Line Parameters as Output Items respectively. Details on default field widths and alignments for text items and global and local customizing controls are given in section Customizing the Format of Output Items.

Text Items: Quoted Text

The simplest text output item is a quoted text. A quoted text is any combination of characters or numbers set apart by a pair of single (') or double (") quotes with each the items needing to use a matching pair. Thus the following three lines are all exactly equivalent.

```
put 'Run on ' system.date ' using source file ' system.ifile;
put "Run on " system.date " using source file " system.ifile;
put 'Run on ' system.date " using source file " system.ifile;
```

Note that there is a limit on the length of all output items: output items may not exceed the page width. In case this limit is exceeded, a put error will be reported in the listing file. Put errors are introduced and discussed in section Put Errors.

Text Items: Identifier Attributes

Text items like the explanatory text of an identifier, the name of a set element and the explanatory text of a set element are specified with identifier attributes. The identifier attributes are listed in Table 6.

Identifier Attribute	Symbol	Description
Explanatory text of an identifier	`.ts`	Displays the descriptive text associated with any identifier (like a set, a parameter, a variable, an equation, a model).
Name of a set element	`.tl`	Displays the names of the individual elements of a set. Observe that the put statement must be embedded in a loop statement, where the respective set is the looping set or the set is a singleton set.

Identifier Attribute	Symbol	Description
Explanatory text of a set element	`.te(index_list)`	Displays the explanatory text associated with a set element. Note that this attribute requires the specification of a driving index. If the set is a simple set, the set name itself will have to be specified as index (see example below). If the set is defined over an index, then the index or a subset of the index may be specified. A special case arises in case users wish to gain access to the explanatory text that appears in the definition of another set (see example below). Observe that if there is no explanatory text for a set element in the set definition, then either the name of the set element (default) or a blank will be displayed, depending on the value of the file attribute text fill (`.tf`). Note that with `.tf=3` even if an explanatory text exists, the name of the element will be displayed. Observe that the put statement must be embedded in a loop statement, where the respective set is the looping set or the set is a singleton set.
Indexed identifier with label combination	`.tn`	Displays the name of the identifier with all individual label combinations. Observe that the put statement must be embedded in a loop statement, where the respective index is the looping set or the set is a singleton set. See example below.

Table 6: Identifier Attributes

Consider the following example:

```
Set  i     master set of sites / i1  Seattle, i2  Portland
                                 i3  San Francisco, i4  Los Angeles, i5 /
     j     subset of sites     / i3 * i5 / ;

File out; put out i.ts /;
loop(i, put i.tl, i.te(i) /);
```

The resulting put file `out.put` will contain the following output:

```
master set of sites
i1          Seattle
i2          Portland
i3          San Francisco
i4          Los Angeles
i5          i5
```

Note that the explanatory text of the set `i` is written first, followed by the label names of the set `i` and their respective explanatory texts. Note further that even though the set `i` does not have an index (it is a simple set), an index is required for the attribute `.te`. In this case we specify as index the set itself. Observe that the set element `i5` was defined without an explanatory text. By default, GAMS inserts the name of the element instead. This may be changed by alterning the code in the following way:

```
put out i.ts /;
out.tf = 0;
loop(i, put i.tl, i.te(i) /);
```

Note that we assign the value of zero to the file attribute text fill (`.tf`) *before* the put statement where one of the items refers to the explanatory text of a set element. As a result, a blank will be displayed where the explanatory text of the set element `i5` is supposed to appear:

```
master set of sites
i1          Seattle
i2          Portland
i3          San Francisco
i4          Los Angeles
i5
```

Assume we also wish to display the elements of the set `j` and their explanatory texts. We could just adapt the code above for the set `j` and add the following two lines:

```
put / j.ts /;
loop(j, put j.tl, j.te(j) /);
```

These lines will result in the follwing output to be appended to the put file:

```
subset of sites
i3
i4
i5
```

Note that the explanatory text associated with the set elements is missing, since there is no explanatory text for the elements in the definition of the set `j` and the value of zero for text fill is still valid. However, we may gain access to the explanatory text of the set elements in the definition of the set `i` in the following way:

```
put / j.ts /;
loop(j, put j.tl, i.te(j) /);
```

Note that the the specification `i.te(j)` singles out the explanatory texts of the elements of the set `i` that are also elements of the set `j`. Hence we will obtain the desired output:

```
subset of sites
i3          San Francisco
i4          Los Angeles
i5
```

The following example illustrates the use of the identifier attribute `.tn`. The model [**MEXSS**] is a simplified representation of the Mexican steel sector, where five steel plants have to satisfy the demand for steel in three markets. Each steel plant `i` has several productive units `m` and the capacity of every unit in every plant is specified with the parameter `k(m,i)`. Assume we wish to write to a put file a list of all nonzero capacities. We could add the following code to the model:

```
File out /out.dat/;
put out 'Capacity (in metric tons)' / /;
loop((m,i)$k(m,i),
   put k.tn(m,i), @30 '=' k(m,i) /;
)
```

Note that we restrict the loop statement with a logical condition to exclude entries of zero. The put file `out.dat` will contain the following lines:

```
Capacity (in metric tons)

k('blast-furn','ahmsa')      =        3.25
k('blast-furn','fundidora')  =        1.40
k('blast-furn','sicartsa')   =        1.10
k('openhearth','ahmsa')      =        1.50
k('openhearth','fundidora')  =        0.85
k('bof','ahmsa')             =        2.07
k('bof','fundidora')         =        1.50
k('bof','sicartsa')          =        1.30
k('direct-red','hylsa')      =        0.98
k('direct-red','hylsap')     =        1.00
k('elec-arc','hylsa')        =        1.13
k('elec-arc','hylsap')       =        0.56
```

Observe that a more elaborate report for the model [**MEXSS**] is given in section Creating a Report for the Model MEXSS at the end of this chapter. The identifier attribute `.tn` is particularly useful when creating scalar EMP info files.

Text Items: Text Blocks

The easiest way to write blocks of text to a put file is with the dollar control options $onPut and $offPut. Consider the following example:

```
File fx; put fx;
$onPut
We will write four
lines of text to the put file,

including a blank line.
$offPut
```

The put file `fx.put` will contain the following lines.

```
We will write four
lines of text to the put file,

including a blank line.
```

Note that these dollar control options have a variant that allows the substitution of compile-time variables. For details see the description of $on/offPut.

4.42.6.2 Numeric Items

Numeric items may be values of parameters, variable and equation attributes or model attributes and expressions of such elemens. These numeric items will be discussed in this section. In addition, numeric items may be command line parameters. For these numeric items, see section Command Line Parameters as Output Items. Details on default field widths and alignments for numeric items and global and local customizing controls are given in section Customizing the Format of Output Items.

Numeric Items: Parameters and Functions

In our first example in this chapter, one of the output items was the parameter `f`. For convenience, we repeat the respective code below:

```
put 'Freight cost  ', f;
```

Note that the simple name of the parameter is sufficient. In the same example we also had two parameters that were defined over an index: `a(i)` and `b(i)`. The respective lines of code follow:

```
loop(i, put @3, i.tl, @15, a(i)/);
* ...
loop(j, put @3, j.tl, @15, b(j)/);
```

Note that indexed parameters must be specified with their index and a put statement with an indexed parameter has to be embedded within a loop structure.

Numeric Items: Variable and Equation Attributes

Recall that data associated with variables and equations is stored in variable attributes and equation attributes respectively. A full list is given in sections Variable Attributes and Equation Attributes.

Suppose we wish to generate a report with the demand for each market in the transportation model [**TRNSPORT**], the satisfied demand after solution of the model and the marginal cost of meeting the demand. Recall that the demand data for each market is saved in the parameter `b(j)` and the relationship between shipment quantities and demand is encoded in equation `demand(j)`. The following code will generate the desired report:

```
File report /report.dat/; put report;
loop(j,
   put 'Report for ' j.tl /
       'Demand'                          @35 b(j):10:0        /
       'Demand satisfied'                @35 demand.l(j):10:0 /
       'Marginal Cost of meeting demand' @35 demand.m(j):10:2 / /;
);
```

Note that we use three numeric output items: the parameter `b(j)`, the level value of the equation `demand.l(j)` and the marginal value of the equation `demand.m(j)`. Note further, that all three items are indexed over the set `j` and thus the put statement has to be placed within a loop statement. Observe that we customize the formatting of the items. For details see section Local Item Formatting Controls. The put file `report.dat` will contain the following output:

```
Report for New-York
Demand                                  325
Demand satisfied                        325
Marginal Cost of meeting demand        0.23

Report for Chicago
Demand                                  300
Demand satisfied                        300
Marginal Cost of meeting demand        0.15

Report for Topeka
Demand                                  275
Demand satisfied                        275
Marginal Cost of meeting demand        0.13
```

Numeric Items: Model Attributes

GAMS models have many model attributes. An introduction and a complete list is given in section Model Attributes. While in principle all model attributes may be used as output items, the attributes `.modelStat` and `.solveStat` with their string valued counter parts `.TModStat` and `.TSolStat` are used most frequently. They refer to the model status and solver termination condition after solution respectively. For a complete list of their values, see sections Model Status and Solver Status.

Suppose we wish to generate a report that contains the time it took to execute the solve statement of the transportation model [**TRNSPORT**], the objective value and the corresponding string valued model and solver status. Recall that the name of the model is `transport`. The following code will generate the desired report:

```
File report /report.dat/; put report;
put 'elapsed time in seconds of solve statement : ' transport.etSolve:0 /
    'objective value:                             : ' transport.objVal:0  /
    'model status:                                : ' transport.tmodstat  /
    'solver status:                               : ' transport.tsolstat  ;
```

Observe that we customize the formatting of the first two items. For details see section Local Item Formatting Controls. The put file `report.dat` will contain the following output:

```
elapsed time in seconds of solve statement : 0.11
objective value:                             : 153.68
model status:                                : 1 Optimal
solver status:                               : 1 Normal Completion
```

4.42.6.3 Set Value Items

There are only two set values: `YES` and `NO`. The set value will be `YES` if a label is an element of a specific set and `NO` otherwise. Consider the following example which is adapted from the example in section Text Items: Identifier Attributes above:

```
Set  i     master set of sites / i1  Seattle, i2  Portland
                                 i3  San Francisco, i4  Los Angeles, i5 /
     j     subset of sites     / i3 * i5 / ;

File out2 / out2.dat /;
put out2 j.ts /;
out2.tf = 0;
loop(i, put i.tl, j(i), '   ',  i.te(i) /);
```

Note that within the loop structure, the put statement writes each element of the set `i`, determines whether it is a member of the set `j` and displays the respective set value, and adds the explanatory text for the label. Hence the resulting put file `out2.dat` will look as follows:

```
subset of sites
i1      NO  Seattle
i2      NO  Portland
i3     YES  San Francisco
i4     YES  Los Angeles
i5     YES
```

Observe that the set values in the second column reflect set membership of the set `j`. Observe further, that the missing explanatory text for label `i5` is displayed as a blank, since the value of the file suffix text fill (`.tf`) equals zero.

4.42.6.4 System Suffixes as Output Items

System suffixes contain information about a GAMS run, they are introduced and discussed in chapter System Attributes. System suffixes may be used as output items in the context of put statements. They are accessed in the following two ways:

```
system.attribute
%system.attribute%
```

Here `system` is a keyword and `.suffix` is a specific system suffixes. Note that `system.suffix` references the *execution-time* version (which can be of type string or numeric) of the system suffixes and `%system.suffix%` (which is interpreted by the compiler as part of the input string) references the *compile-time* version. For further details on the difference between execution-time and compile-time system suffixes, see chapter System Attributes. A complete list of all system suffixes is given in section List of all System Suffixes, in Table 7 we present a list of the most common system suffixes.

System Attribute	Description
`.date`	Program execution date
`.ifile`	Input file name
`.ofile`	Output file name
`.rdate`	Restart file date
`.rfile`	Restart file name
`.sfile`	Save file name
`.title`	Title of the model as specified by $title

Table 7: A Selection of System Suffixes

To illustrate how system suffixes are used, assume we wish to include the program execution date, the name of the input (GAMS) file, and the page number of the current put statement in the input file to the report `results.dat` in section A First Example above. We will modify the code in the following way:

```
File factors /factors.dat/, results /results.dat/;
* ...
put results;
puthd 'Program Execution Date:', @26, system.date /
    'Source File:', @26, system.ifile /;
    'Page Number:', @26, system.page / /;
put 'Transportation Model Results' / / ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 /);
```

The file `results.dat` will then contain the following lines:

```
Program Execution Date:  01/13/17
Source File:             C:\Documents\GAMS\Models\trnsport.gms
Page Number:             1

Transportation Model Results

Seattle    New-York     50.0000
Seattle    Chicago     300.0000
Seattle    Topeka        0.0000
San-Diego  New-York    275.0000
San-Diego  Chicago       0.0000
San-Diego  Topeka      275.0000
```

Note that the date is given in the American format: `month/day/year` and can be reset via the command line parameter DFormat. Also note that including the page number in the header block of a put file will have the effect that the pages of the put file will be numbered. Of course, this is especially useful for longer put files.

It is also possible to read environment variables at execution time in a put context using `sysEnv.name` as in the following example:

```
$setEnv hello world
put_utility 'log' / sysEnv.hello;
```

If the environment variable queried is not defined, the result depends on the value of the command line parameter stringChk.

4.42.6.5 Command Line Parameters as Output Items

Command line parameters may also be used as output items. They are introduced and discussed in chapter The GAMS Call and Command Line Parameters. For an overview of all GAMS command line parameters, see section List of Command Line Parameters. Like system suffixes , command line parameters are referenced in the context of a put statement either compile-time strings or in an execution time version as follows:

```
put "%gams.parameter%";
put gams.parameter;
```

Here `parameter` is a GAMS command line parameter. To illustrate how command line parameters are used, assume we wish to include the page size of the input file, the name of the input file and the name of the restart file in the report `results.dat` in section A First Example above. We will modify the code in the following way:

```
File factors /factors.dat/, results /results.dat/ ;
* ...
put results;
put 'Transportation Model Results'// ;
loop((i,j), put i.tl, @12, j.tl, @24, x.l(i,j):8:4 //);
put / "Page size         = %gams.ps%"
    / "GAMS input file   = %gams.input%"
    / "GAMS restart file = %gams.restart%";
```

The last three lines of the put file `results.dat` follow:

```
Page size         = 58
GAMS input file   = C:\Documents\GAMS\Models\trnsport.gms
GAMS restart file =
```

Note that there was no restart file in the GAMS run, thus the value for `%gams.restart%` is the empty string.

In this example it actually makes no difference if one uses the compile- or execution time version to query the value of a command line parameter. This might be different, when doing the compile and execution phase separately, e.g. when using remote execution. Here is a toy example showing the difference:

```
$onEchoV > logScrDir.gms
put_utility 'log' / '%gams.scrDir%';
put_utility 'log' / gams.scrDir;
$offEcho

$call gams logScrDir.gms a=c s=1 keep=1
$call gams logScrDir.gms a=e r=1
```

In the log, one will see that `%gams.scrDir%` is evaluated in the first run at compile time already, while `gams.scrDir` is evaluated at execution time in the second run:

```
--- logScrDir.gms(1) 3 Mb
    C:\Data\t m p\225d\
--- logScrDir.gms(2) 3 Mb
    C:\Data\t m p\225e\
```

4.42.6.6 Customizing the Format of Output Items

GAMS provides global and local controls to modify the default format of output items. *Global* controls are set with file attributes and apply to all output items in a put file that follow the assignment of a file attribute. *Local* controls are used to change the format of only one specific output item.

Global Item Formatting Controls

The alignment (justification) of the field and the width of the field may be modified for all types of output items. The attributes that control field alignment are listed in Table 8. Note that possible values are 1 (right), 2 (left) and 3 (center).

Type of Output Item	Symbol	Default Value
Text items: set labels	`.lj`	2: Left
Text items: quoted and explanatory text	`.tj`	2: Left
Numeric values	`.nj`	1: Right
Set values	`.sj`	1: Right

Table 8: File Attributes for Field Alignment

The width of the field is specified with the number of spaces to be allocated. The attributes that control field width are liste in Table 9.

Type of Output Item	Symbol	Default Value
Text items: set labels	`.lw`	12
Text items: quoted and explanatory text	`.tw`	0
Numeric values	`.nw`	12
Set values	`.sw`	12 (maximum 20)

Table 9: File Attributes for Field Width

Note that the value of zero signifies a *variable* field width, matching the the exact size of the item being displayed. If a text output item does not fit within the specified field, truncation will occur to the right. For numeric output items, the decimal portion of a number is rounded or scientific notation is used to fit the number within the given field. If a number is still too large, asterisks will replace the value in the output file.

For example, the following assignment will set the field width for numeric items in the file `out.put` globally to 4:

```
out.nw = 4;
```

In addition to field alignment and field width, there are further global controls that apply to numeric output items only. They are given in Table 10.

File Attribute	Symbol	Description	Default Value	Optional Values
Number of decimals	`.nd`	Sets the number of decimals that are displayed for numeric items. A value of zero entails that fixed-format output will show the number rounded to the nearest integer.	2	Values may be between 0 and 20.

File Attribute	Symbol	Description	Default Value	Optional Values
Numeric round format	.nr	Selects the formatting used for numeric output. For example, one can choose to use scientific notation (i.e. E-format) for all numbers: this is especially useful for numbers so small that they display as only zeros in fixed format with the value specified for number of decimals (.nd). The default rounding format will treat small values in this way, but in many situations, it is important to be aware that such small values exist.	1	0: Item is displayed in F or E format to fit given width and decimals. 1: Item is rounded to fit given width and decimals. 2: Item is displayed in E format to fit given width and decimals. 3: Item is rounded to fit given width. 4: Item is displayed in F or E format to fit given width.
Numeric zero tolerance	.nz	Sets the tolerance level for which a number will be rounded to zero for display purposes. Note that in case this attribute is set to zero, rounding is determined by the field width.	1.0e-10	

Table 10: File Attributes for Global Format Control Specific to Numeric Items

Note that numeric items are always formatted to fit within the specified width limit, but their display may use fewer characters. Similarly, for some values of .nr, the number of digits shown is limited by the choice of the .nd parameter, but fewer digits may be shown. At most 17 significant digits are included - this is enough to represent any double-precision value unambiguously, i.e. as a base-10 string that converts back into this same double. With this in mind, any number can be represented fully in E-format using at most 24 digits, e.g. -1.25E+102.

With .nr=0, the formatting routines honor the decimals limit. Values closer to 1 are displayed in fixed format. Values so small they would display only zeros in fixed format are displayed in E-format, as are values too large for the width limit. Formatting with .nr=1 is similar, but small values are shown in fixed format even if they show all zeros. With .nr=2 E-format is used for all values, with the number of trailing digits respecting the decimals limit .nd. The decimals limit is ignored for .nr=3, and fixed format is used unless the value is so large that E-format output is shorter. Finally, .nr=4 is intended to be used with larger output widths, e.g. 18 or 24, in order to show values of all magnitudes in as full a precision as possible: it ignores the decimal limit and uses whatever format allows more significant digits in the specified width.

The following example illustrates the result of different combinations of numeric file attributes. Note that we will use five combinations of the file attributes .nd, .nz, .nr and .nw to display three numerical values.

```
Set  c  suffix combinations  / comb1 * comb9 /
```

```
     v  value indices        / value1 * value5 / ;

Table   suffix(c,*)   numeric suffix combinations
           nd      nz      nr      nw
comb1       3       0       0      12
comb2       3    1e-5       0      12
comb3       3    1e-5       1      12
comb4       8       0       0      10
comb5       6    1e-5       1      12
comb6       0    1e-5       1      12
comb7      10       0       2      18
comb8       0       0       3      14
comb9       0       0       4      18 ;

Parameter  value(v)   test values
      / value1     1.2345678901e14
        value2     [pi]
        value3      -0.1234567
        value4       0.0001234567
        value5       1.234567e-13 / ;

File out; put out;
out.nj=2; out.lw=10; out.cc=11;
loop (v, put v.tl:21);
loop (c,
  out.nd=suffix(c,"nd");
  out.nz=suffix(c,"nz");
  out.nr=suffix(c,"nr");
  out.nw=suffix(c,"nw");
  put / c.tl;
  loop (v,
    put @(ord(v)*21-10), value(v)
  )
);
```

Observe that we have chosen to align the values to the left. This will enhance readability as the model loops through the suffix combinations that entail different field widths. The resulting output file `out.put` follows:

```
          value1               value2               value3               value4               value5
comb1     1.234568E+14         3.142                -0.123               1.2345670E-4         1.2345
comb2     1.234568E+14         3.142                -0.123               1.2345670E-4         0.000
comb3     1.234568E+14         3.142                -0.123               0.000                0.000
comb4     1.2346E+14           3.14159265           -0.1234567           0.00012346           1.2346
comb5     1.234568E+14         3.141593             -0.123457            0.000123             0.0000
comb6     1.234568E+14         3                    -0                   0                    0
comb7      1.2345678901E+14     3.1415926536E+00    -1.2345670000E-01     1.2345670000E-04     1.2345
comb8     1.23456789E+14       3.141592653590       -0.12345670000       0.000123456700       0.0000
comb9     123456789010000      3.141592653589793    -0.12345678          1.234567E-4          1.2345
```

Note that in `comb1` the display of values switches to the exponential notation when the value becomes smaller than the number of decimal places allowed. This is a result of `.nr=0`. Note further, that `value4` is greater than the zero tolerance level (`.nz`), but smaller than the number of decimals allowed (`.nd`) in both, `comb2` and `comb3`. However, `.nr=0` results in the exponential notation in the display of `comb2`, while `.nr=1` has the effect that this small value is rounded to zero. Observe that in `comb6` fixed-format is used whenever allowed by the width limit, and in these cases the output is rounded to integer because `.nd` is

set to zero. With `comb7` E-format is used in all cases: a leading space is added for positive values and a minus sign to negative values, and the magnitude is indicated with an explicit plus/minus and two digits (three if necessary). With `comb8` we see fixed-format used whenever possible and as many digits (significant or not) as allowed by the width. With `comb9` we see only significant digits, with the format chosen to allow the display of as many digits as possible.

Local Item Formatting Controls

The local item formatting controls allow to format specific output items. Note that local formatting overrides global format settings. The syntax is as follows:

```
item:{<>}width:decimals;
```

Here `item` is the output item, followed by a colon, an alignment (justification) symbol, the field width, a colon and the number of decimals to be displayed. Note that the limit on the number of the decimal places is only valid for numeric output. Note further, that if a component of the local formatting feature is omitted, then the respective global formatting settings will be used. The local alignment symbols are listed in Table 11.

Symbol	Alignment
>	Right
<	Left
<>	Center

Table 11: Local Alignment Symbols

Observe that similar to global formatting, a field width of zero means that the field width will be variable, depending on the item to be displayed.

The following example serves as illustration. Observe that we use end-of-line comments to annotate the code.

```
File out; put out;

$eolCom //
Set       i    / i1*i3 /;
Parameter d(i) / i1 1426, i2 1347, i3 900 /;
Scalar    f    / 17.6745 /;

loop(i, put d(i):0:0 /);  // default justification and a field width
                          // of variable size with no decimals

put / 'Right Justified Comment':>50
    / 'Center Justified Truncated Comment':<>20;

put / / f:<6:2;           // left aligned scalar with 6 spaces for field width
                          // and two decimals
```

4.42.7 The Put_Utility Statement

The `put_utility` statement is a variant of the put statement that may be used to execute external programs. The syntax is as follows:

```
put_utility [file_name] 'command' / 'arguments' {/ 'command' / 'arguments'};
```

The keyword `put_utility` and its synonym `put_utilities` indicate that this is a put_utility statement. The keyword is followed by the internal name of an external file, `file_name`. Note that `file_name` may be omitted. It is not required for the `put_utility` statement but might be used to activate a file to be used with following `put` statements. `Command` denotes one of the commands listed in Table 12 below. Commands are followed by a slash and their respective `arguments`. Observe that a put_utility statement may contain multiple command / argument pairs. An example is given below.

The following simple example illustrates the put_utility statement:

```
File test / test.txt /;
put test "This is the original file."

put_utility 'ren' / 'test.dat';
putclose "This is the renamed file.";
test.ap = 1;
put "Write to the renamed file.";
```

Note that first the put file `test.txt` with the internal name `test` is defined, made current and written to. Then the put_utility statement uses the command ren to *rename* the current put file. The new name is `test.dat` and the internal name `test` will from now on reference the new file `test.dat`. However, the original file `test.txt` is not deleted. Hence, the code above will create two external files: `test.txt` and `test.dat`. The file `test.txt` will contain the following line:

```
This is the original file.
```

The file `test.dat` will have the following content:

```
This is the renamed file.
Write to the renamed file.
```

Note

If Put File Attributes are specified, they are also considered for the `put_utility` commands and arguments. This could lead to surprising results in some cases, like in this example:

```
File test / test.txt /;
test.pc = 5;
put test "This is the original file."

put_utility 'ren' / 'test.dat';
putclose "This is the renamed file.";
```

This will trigger the following errors:

```
*** Error at line 5: Put_Utilities: Unknown request ""ren""
*** Error at line 5: Put_Utilities: Unknown request ""test.dat""
```

The problem is, that `test.pc = 5;` puts quotes around non-numeric output, and thus `"ren"` (including the quotes) is not recognized as command anymore. To overcome this issue `test.pc` should be set back to default for the `ren` command like this:

```
File test / test.txt /;
test.pc = 5;
put test "This is the original file.";

test.pc = 2;
put_utility 'ren' / 'test.dat';
test.pc = 5;
putclose "This is the renamed file.";
```

We will present a list of all commands and their arguments in Table 12 and give examples for most commands below.

Table 12: List of Commands and their Arguments

Command	Description of Command	Description of Arguments
`assignText`	Allows to set the explanatory or label text of a singleton set element. See example below.	Name of singleton set symbol and text.
`click`	Adds a clickable file reference to the process window of the IDE. See example below.	File name of the file to which the reference will point.
`dropEnv`	Removes an environment variable at execution time.	Name of the environment variable.
`ECArguments`	Allows to set the arguments for execution time embedded code execution. See example in section Embedded Code at Execution Time.	Text for the embedded code argument.
`exec`	Passes a command to the operating system for execution. GAMS will wait until the command is executed. See example below. Note that the distinction between `exec` and shell is technical and may be operating system specific. Typically, the ability to use redirect of standard input output and the error console is involved.	Command to be executed with arguments.
`exec.aSync`	Passes a command to the operating system for execution. However, GAMS will *not* wait until the command is executed.	Command to be executed with arguments. Job control is handled identical to jobs spawned via `execute.async`.
`exec.aSyncNC`	Passes a command to the operating system for execution using a different console than the parent process (Windows only). GAMS will *not* wait until the command is executed, thus using multiple processors is possible. Job control is handled identical to jobs spawned via `execute.async`.	Command to be executed with arguments.
`exec.checkErrorLevel`	Passes a command to the operating system for execution. GAMS will wait until the command is executed, raises an execution error and aborts the execution, if the errorLevel returned is not 0.	Command to be executed with arguments.

Command	Description of Command	Description of Arguments
gdxIn	Accesses the GDX file specified in the argument. A subsequent directive execute_load or execute_loadpoint without a specified file name will unload data from the GDX file thus accessed. See example below.	Name of the GDX file that data will be loaded from.
gdxOut	Creates a new GDX file or accesses an existing GDX file specified in the argument. A subsequent directive execute_unload without a specified file name will write to the GDX file thus created or accessed. Note that if an existing GDX file is accessed, it will be overwritten. See example below.	Name of the GDX file to which data will be unloaded.
glb	Used by GAMS to facilitate building the model library - not intended for users.	–
htm	Used by GAMS to facilitate building the model library - not intended for users.	–
inc	Includes the contents of an external file in the currently active put file. See example below.	File name of external file.
incLog	Includes the contents of an external file in the currently log file . See example below.	File name of external file.
incMsg	Includes the contents of an external file in the currently listing file . See example below.	File name of external file.
incMsgLog	Includes the contents of an external file in both, the log file and the listing file. See example below.	File name of external file.
log	Sends a message to the log file. See example below.	Text of message.
msg	Sends a message to the listing file. See example below.	Text of message.
msgLog	Sends a message to both, the log file and the listing file. See example below.	Text of message.
ren	Creates a new external name for the *current* put file. The internal name will reference the new external file. Any subsequent put statements will write to the new file. See the simple example above and a more complex example below.	External file name.
save	Writes a save file of the current state of execution.	Name of save file.
setEnv	Sets an environment variable at execution time.	Name of the environment variable and its value.

Command	Description of Command	Description of Arguments
`shell`	Passes a command to the command shell processor, where it is processed. The processed form of the command is then passed to the operating system for execution. See example below. Note that the distinction between `shell` and exec is technical and may be operating system specific. Typically, the ability to use redirect of standard input output and the error console is involved.	Command to be executed with arguments.
`shell.checkErrorLevel`	Passes a command to the command shell processor, where it is processed. The processed form of the command is then passed to the operating system for execution. GAMS checks the errorLevel returned implicitly, raises an execution error and aborts the execution, if is not 0.	Command to be executed with arguments.
`solver`	Selects a solver for a given or all model types (use $*$) by name. See example below.	Model type or $*$ and solver name.
`stdOut`	Sends a message to stdOut independent of the log file.	Text of message.
`stdErr`	Sends a message to stdErr independent of the log file.	Text of message.
`title`	Changes the title on the DOS window.	New name for window.
`winMsg`	Sends a message to a window on a Windows machine. For examples, see models [**ASYNNTRP**] and [**MRW01**] in the GAMS Test Library.	Window name and message.
`xsave`	Writes a compressed save file of the current state of execution.	Name of compressed save file.

In the remainder of this section we will present examples.

Exec: Creating Empty Files by Executing External Program touch

Consider the following example:

```
Set i / 1*3 /;
loop(i, put_utility 'exec' / 'touch ' i.tl:0 '.txt');
```

Note that the command to be executed is `touch`. Thus this code snippet will create three empty files called `1.txt`, `2.txt` and `3.txt`.

Exec and Ren: Creating Directories and Renaming Files

Consider the following example:

```
File test / test.txt /; put test;
Set i / i01*i07 /;

loop(i,
   put_utilities 'exec' / 'mkdir ' 'test-':0 i.tl:0;
   put_utilities 'ren' / 'test-':0 i.tl:0 '%system.dirSep%test-':0 i.tl:0 '.txt':0 ;
   put 'this should be ' i.tl );
```

Observe that as the loop is executed, the first put_utitily statement will create seven subdirectories called `test-i01`, ..., `test-i07`. The second put_utility statement will create a text file for each of the new subdirectories. The text file in subdirectory `test-i01` is called `test-i01.txt`, the text file in subdirectory `test-i02` is called `test-i02.txt`, etc. The put statement in the last line will write to each text file. For example, the following line will be generated for the text file `test-i01.txt`:

```
this should be i01
```

Note that at the end of the loop, the external file associated with the internal file name `test` is the put file `test-i07.txt`, since this file was the last current put file.

Inc: Including the Content of a File

In this example, we first create the external file `recall.txt`, write to it and close it. Then we create a new external file called `report.dat`. In a third step we include the content of the first file in the second file.

```
File recall /recall.txt/;
putclose recall "I am the external content."

File report /report.dat/;
put report "Here we include content from an external file.";
put_utility 'inc' / 'recall.txt' ;
```

Note that the file `report.dat` will contain the following lines:

```
Here we include content from an external file.
I am the external content.
```

IncMsg, IncLog and IncMsglog: Including the Content of a File to the Log File and Listing file

In this example, we first create the external file `recall.txt`, write to it and close it. Then we include the content of `recall.txt` into the log and listing file.

```
File recall /recall.txt/;
putclose recall "I am the external content." / "I go over more than one line.";

display 'before the include';
put_utility 'incMsg' / 'recall.txt' ;
display 'after the include';
```

The listing file of this model shows the content of `recall.txt`:

```
----      4 before the include
*** Start of include of file recall.txt
I am the external content.
I go over more than one line.
*** End of include of file recall.txt

----      6 after the include
```

Msg, Log and Msglog: Writing to the Log File and Listing file

Consider the following example:

```
put_utility 'msg'    / 'This message is for the lst file.'
          / 'log'    / 'This message is for the log file.'
          / 'msgLog' / 'And this message is for the lst and the log file.' ;
```

Note that the following two lines will be generated in the log file:

```
This message is for the log file.
And this message is for the lst and the log file.
```

In addition, the listing file will contain the following two lines just before the report file summary.

```
This message is for the lst file.
And this message is for the lst and the log file.
```

Gdxout: Creating GDX Files and Unloading Data to them

Consider the following example:

```
Set j / 2005*2007 /;
Scalar random;

loop(j,
   put_utility 'gdxOut' / 'data' j.tl:0;
   random = uniform(0,1);
   execute_unload random;
);
```

This code will create the GDX files `data2005.gdx`, `data2006.gdx` and `data2007.gdx`. Note that each GDX file will contain a value between zero and 1 for the scalar `random`. For example, the file `data2005.gdx` (exported to an ASCII via `gdxdump`) will have the following content:

```
Scalar random / 0.171747132 /;
```

Gdxin: Loading Data from GDX Files

Note that the following example is an extension of the previous example that demonstrated the use of the command `gdxout`.

```
loop(j,
   put_utility 'gdxIn' / 'data' j.tl:0 ;
   execute_load random;
   display random;
);
```

This code loads the values of the scalar `random` from the GDX files `data2005.gdx`, `data2006.gdx` and `data2007.gdx` and displays them in the listing file of the GAMS input file:

```
----     19 PARAMETER random                =        0.172
----     19 PARAMETER random                =        0.843
----     19 PARAMETER random                =        0.550
```

Shell: Writing to Various Files

Consider the following example:

```
Set j / j1*j5 /;
Scalar random;

loop(j,
   random = uniformint(0,100);
   put_utility 'shell' / 'echo ' random:0:0 ' > ' j.tl:0;
);
```

Observe that the shell script command `echo` outputs an integer between zero and 100 to the files `j1`, ..., `j5`.

Click: Adding a Clickable Link

Assume that there is a file called `sets.html` in our working directory. Consider the following code snippet:

```
put_utility 'click' / 'sets.html' ;
```

If we run this code snippet with the GAMS IDE, the following clickable link will appear in the IDE process window:

```
>>> File sets.html
```

Solver: Select a solver by name at execution time

The `solver` keyword allows to select a solver at execution time by name. Normally, the option statement `option solver=xpress, lp=cplex;` is used to select the current solver, but in case where you want to programatically change the solver as in this example, this put_utility keyword can be useful. Consider the following code snippet which extends the [**DICE**] model:

```
...
set slv 'MIP solvers to run' / cplex, cbc, gurobi, mosek, scip, xpress /;
parameter rep 'report status, time, objective value, and more';
option bratio=1;
loop(slv,
   put_utility 'solver' / 'mip' / slv.tl:0;
   solve xdice using mip max wnx;
   rep(slv,'sstat')     = xdice.solveStat;
   rep(slv,'mstat')     = xdice.modelStat;
   rep(slv,'obj')       = xdice.objVal;
   rep(slv,'time')      = xdice.etSolve;
   rep(slv,'solver id') = xdice.sysIdent;
);
display rep;
```

If we run this code we get a report like this:

```
----     82 PARAMETER rep  report status, time, objective value, and more

             sstat       mstat         obj        time   solver id

cplex        1.000       1.000      21.000       0.455      19.000
cbc          1.000       8.000      21.000       2.804       6.000
gurobi       1.000       1.000      21.000       0.631      32.000
mosek        1.000       1.000      21.000       4.425      48.000
scip         1.000       1.000      21.000       3.907      65.000
xpress       1.000       1.000      21.000       2.571      71.000
```

If one want's to set a solver for all possible model types (similar to `option solver=xpress;`) specific model type has to be replaced by $*$: `put_utility 'solver' / '*' / 'xpress';`.

AssignText: Assigns label text to an element of a singleton set

GAMS has no string data type. The explanatory or label text of an element can serve in several situations as a poor man's string type and used in `put` and `put_utility` statements. The example below extends the [**TRNSPORT**] model and disables flow on a particular connection. The solution point file created by savePoint gets renamed to e.g. `sol_seattle_new-york.gdx`. The filename is assembled using the `assigntext` keyword. This has been packaged for better readability in some `STRING` macros:

```
...
$macro STRINGDEF(sym)         singleton set sym / sym /
$macro STRING(sym)            sym.te(sym)
$macro STRINGASSIGN(sym,text) put_utility 'assignText' / 'sym' / text
$macro STRINGAPPEND(sym,text) put_utility 'assignText' / 'sym' / sym.te(sym) text
alias (i,ii), (j,jj);
transport.savePoint = 1;
STRINGDEF(fname);
loop((ii,jj),
   x.up(ii,jj) = 0;
   solve transport min z using lp;
   x.up(ii,jj) = inf;
   STRINGASSIGN(fname,'sol');
   STRINGAPPEND(fname,'_' ii.tl:0);
   STRINGAPPEND(fname,'_' jj.tl:0 '.gdx');
   put_utility 'shell' / 'mv transport_p.gdx ' STRING(fname);
);
```

4.42.8 Conditional Put Statements

In GAMS, shorthand notation for conditional statement are dollar conditions, as introduced in chapter Conditional Expressions, Assignments and Equations. Dollar conditions may also be used in the context of a put statement and its variants. The syntax in its most general form is as follows:

```
put $ logical_condition [file_name] item {item} {file_name item {item}};
```

Note that like all put statements, the *conditional* put statement begins with the keyword `put`. The keyword is followed by the dollar operator and a logical condition. If the logical condition is `TRUE` the put statement will be executed, otherwise the put statement will be ignored. For details on logical conditions in GAMS, see sections Logical Conditions and Filtering Sets. Observe that the remainder of the conditional put statement is identical to the regular put statement introduced in section The Put Statement.

The following example demonstrates how dollar conditions are used in the context of put statements.

```
put$(a > 10) 'Some output items';
```

Note that the quoted text will only be written to the put file if the scalar or variable `a` is greater than 10.

4.42.9 Errors Associated with Put Statements

There are two types of errors that may occur when the put writing facility is used: syntax errors and put errors. In this section we will discuss these errors.

4.42.9.1 Syntax Errors in Put Statements

Syntax errors are caused by the incorrect usage of the GAMS language, including unmatched parentheses, undefined identifiers, uncontrolled sets or the incorrect use of a keyword. These errors are detected during program compilation and are always fatal to program execution. For more information on compilation errors, see section Compilation Errors and the tutorials A GAMS Tutorial by Richard E. Rosenthal and Fixing Compilation Errors.

4.42.9.2 Put Errors

Put errors are unique to the put writing facility. They are detected during program execution and are caused when the specifications of file attributes are violated. Typical errors include assigning inappropriate values to file attributes and attempts to write outside a page, like moving the cursor with the cursor control character `@` to a location beyond the page width. Consider the following example:

```
File out /out.dat/;
out.pw = 8;
put out "Let's try this.";
```

Note that we specify the page width (`.pw`) to be just 8 characters. However, the quoted text has clearly more than 8 characters. In such a case the GAMS code will be compiled and the log file will report "Normal completion". At the appropriate position in the listing file the following error will be reported:

```
**** PUT ERROR FOR FILE out AT LINE 5: PUT LINE OVERFLOW - LOOK FOR **** ON PUTFILE, YOU CAN RESET .
```

The put file `out.dat` will contain the following line:

```
Let'****
```

As put errors are not fatal and are not emphasized in the log file, they may be easily overlooked. Especially in large put files, put errors may go undetected. GAMS provides the file attribute `.errors` that facilitates the display of the numer of put errors. Consider the following example:

```
File out /out.dat/;
out.pw = 8;
put out "Let's try this.";

abort$(out.errors) "Put errors in out:", out.errors;
```

Users may choose to output the number of put errors in a put file or display statement or even trigger an execution error as in the example above.

4.42.10 Creating a Report for the Model MEXSS

We started this chapter with a simple example and we will complete it with a more elaborate example. In this section we will show how the put writing facility may be used to create a report for the model [**MEXSS**]. The code for the report may be inserted at the end of the original model and is shown below in its entirety.

The model [**MEXSS**] analyzes the relative efficiency of five different plants for meeting the product requirements of ingot steel in three different markets. The model may be used to identify the major bottlenecks that constrain production in the system of plants. It will find the production levels in the steel mills and shipments from the mills to the markets that will meet the market requirements at least cost. We will create a report for this model that will present details on the available capacities of the productive units at the five plants, the unused capacities and the marginal values of the capacities. This report may be extended to include other data and results and the code may be reused to create new reports if the data in the model is changed. For models that are run periodically, say, weekly or monthly, reusable reports can be invaluable.

We will start the code for the report with defining the put file, setting the file attributes print control and page size and making the the put file current. Then we will specify some global formatting settings. We will continue with writing a title block and finally turn to the core of the report: a table with three subtables. Note that we will use in-line comments to annotate the code. The code follows:

```
$eolcom //
File out /out.dat/;
out.pc=3; out.ps=54;       // print control, page size
put out;

* Global Format Settings:
Scalars   indent1    indent to first column of units display
          indent2    indent to first column of field labels
          indent3    indent to first column of first numeric field
          textwide   wide text       / 80 /
          textnarr   narrow text     / 30 / ;
```

```
out.nr  = 0;               // numeric round format
out.tw  = textwide;        // width of text field
out.lw  = 11;              // width of label field
out.nw  = 11;              // width of numeric field
out.nd  = 2;               // number of decimals displayed
indent1 = 3;
indent2 = 30;
indent3 = 27;

* Title Block
puttl   'MEXICO STEEL - SMALL STATIC MODEL':<> /
        system.date:<>/ / /;

* Main Window
out.tj = 3;                // alignment of text: center
put
  '      This report is based on selected data and results from the model'/
  'MEXSS in the GAMS Library. This model analyzes the relative efficiency'/
  'of five different plants in meeting the product requirements for ingot'/
  'steel in three different markets. The model aims to find the pattern  '/
  'of production levels in the mills and shipments from the mills to the '/
  'markets that will meet the market requirements at the least cost.     '/
  /
  'Reference:  Kendrick, D. A., Meeraus, A., Alatorre J., The Planning '/
  'of Investment Programs in the Steel Industry, John Hopkins Universi-'/
  'ty Press, 1984.                                                      '/
  / / /;

out.tj = 2;                // alignment of text: left
out.tw = textnarr;         // width of text field

putpage$(out.ll+card(m)+7 > out.ws);   // manual paging

put 'Table 1.  Plant Data and Results':0 /'-------'/;    // Table 1
out.cc = indent2;          // current column
loop(i, put i.tl:<>);      // column headings

put / / 'CAPACITY (metric tons)';                        // Capacity
loop(m,
   put / @indent1, m.te(m), @indent3;  // row headings
   loop(i, put k(m,i));                // numeric values
);

* Header Block with column headings for next page (will be used only if necessary)
puthd 'Table 1 (continued).  Plant Data and Results':0 /
      '--------------------'/;
out.hdcc = indent2;
loop(i, put i.tl:<>);      // column headings
puthd /'';

* Main Window continued
if(out.ll+card(m)+sum((m,p)$b(m,p), 1)+3 > out.ws,
   putpage;                // manual paging
   else put / /;
);

put 'UNUSED CAPACITY (metric tons)';                      // Unused Capacity
```

```
loop(m,
   put / @indent1, m.te(m), @indent3;   // row headings
   loop(i, put (k(m,i)-cc.l(m,i)));     // numeric values
);

if(out.ll+card(m)+4 > out.ws,
   putpage;               // manual paging
   else put / /;
);

put 'MARGINAL VALUE OF CAPACITY'/ @indent1 '(US$/ton)';   // Marginal Value of Capacity
loop(m,
   put / @indent1, m.te(m), @indent3;   // row headings
   loop(i, put abs(cc.m(m,i)));         // numeric values
);
```

Note that we set print control (.pc) to ASCII formfeed. Below we illustrate how to use one of the print control settings that generate a delimited file. Observe that we group all global format parameters and settings near the top of the code. This way of organizing the code will make it easy to modify the structure of the report in the future as needed. Observe further, that the output items in the title block are locally formatted to be center aligned.

In the remainder of the code the actual report is written in the main window. The block of text at the beginning serves as a brief introduction to the report. Note that the core of the report is a table consisting of three subtables, where the subtables share the column headings. Before writing the table, we insert a test - the conditional putpage statement - to determine whether there is a sufficient number of lines on the current page to accommodate the size of the first subtable. Thus the put writing facility would start a new page if there were not enough rows. We repeat similar tests before the code for the other two subtables. Observe that we insert a header block with the column headings. If a new page had to be started since there were not enough rows for the second or the third subtable, the header block would contain the column headings of the table. (Users might want to experiment by reducing the file attribute page size (.ps) to say, 30.) The report that will be generated follows:

```
                    MEXICO STEEL - SMALL STATIC MODEL
                                 01/31/17


          This report is based on selected data and results from the model
     MEXSS in the GAMS Library. This model analyzes the relative efficiency
     of five different plants in meeting the product requirements for ingot
     steel in three different markets. The model aims to find the pattern
     of production levels in the mills and shipments from the mills to the
     markets that will meet the market requirements at the least cost.

      Reference:  Kendrick, D. A., Meeraus, A., Alatorre J., The Planning
      of Investment Programs in the Steel Industry, John Hopkins Universi-
      ty Press, 1984.


Table 1.  Plant Data and Results
-------
                                ahmsa    fundidora  sicartsa     hylsa     hylsap

CAPACITY (metric tons)
  blast furnaces                 3.25       1.40       1.10      0.00       0.00
  open hearth furnaces           1.50       0.85       0.00      0.00       0.00
```

```
  basic oxygen converters        2.07       1.50       1.30       0.00       0.00
  direct reduction units         0.00       0.00       0.00       0.98       1.00
  electric arc furnaces          0.00       0.00       0.00       1.13       0.56

UNUSED CAPACITY (metric tons)
  blast furnaces                 0.13       0.00       0.00       0.00       0.00
  open hearth furnaces           0.00       0.00       0.00       0.00       0.00
  basic oxygen converters        0.00       0.72       0.14       0.00       0.00
  direct reduction units         0.00       0.00       0.00       0.00       0.39
  electric arc furnaces          0.00       0.00       0.00       0.23       0.00

MARGINAL VALUE OF CAPACITY
  (US$/ton)
  blast furnaces                 0.00      69.62      71.69       0.00       0.00
  open hearth furnaces          53.76       1.72       2.09     138.03     145.02
  basic oxygen converters       64.57       0.00       0.00     138.03     145.02
  direct reduction units         0.00       0.00       0.00      80.08       0.00
  electric arc furnaces        136.46     138.03     140.00       0.00      94.28
```

Assume we need the data reported in the first subtable above in a delimited file format in order to import it to other applications, like spreadsheets or databases. Consider the following code:

```
file out2 / 'out2.csv' /;
out2.pc=5;
put out2 'capacity (metric tons)';
loop(i, put i.tl);
loop(m,
     put / m.te(m);
     loop(i, put k(m,i));
);
```

Note that we set the file attribute page control (`.pc`) to 5. This means that non-numeric output items will be quoted and each output item will be delimited with commas. Observe that field widths, alignments and horizontal cursor relocations were completely avoided. The put file `out2.put` will contain the following lines:

```
"capacity (metric tons)","ahmsa","fundidora","sicartsa","hylsa","hylsap"
"blast furnaces",3.25,1.40,1.10,0.00,0.00
"open hearth furnaces",1.50,0.85,0.00,0.00,0.00
"basic oxygen converters",2.07,1.50,1.30,0.00,0.00
"direct reduction units",0.00,0.00,0.00,0.98,1.00
"electric arc furnaces",0.00,0.00,0.00,1.13,0.56
```

While a comma is the most commonly used delimiting character, other delimiters like a blank space (`.pc=4`) and a tab (`.pc=6`) may also be used.

For other examples of code that uses the put writing facility, see the various models in the GAMS Model Library.

4.43 Solver Usage

For the novice GAMS user, solver usage can be very simple: one runs the model and inspects the listing file to see what the solution is. No knowledge of solver options or solver specific return codes is required. While this is enough for some users, most will quickly find they need some basic knowledge of how to control the solver and interpret the results. Section Controlling a Solver via GAMS Options describes how to set the GAMS options that control a solver. Further, most solvers allow the user to set additional, solver-specific options. These can be set via a solver specific options file, which will be discussed in Section The Solver Options File. However, use of generic GAMS options should be preferred, since a GAMS option setting applies to all solvers and is interpreted by the solvers in a consistent way.

A number of solvers can make use of an initialization of variable and equation values. This will be discussed in Starting Point and Initial Basis.

Further solver specific topics, which are more interesting for advanced users, are discussed in the Sections Solve trace and Branch-and-Cut-and-Heuristic Facility (BCH).

For some hints on how to select a solver, see Choosing an appropriate Solver.

4.43.1 Controlling a Solver via GAMS Options

GAMS options can be set on the GAMS command line, e.g.,

```
$ gams trnsport iterlim = 100
```

Additionally, they can be set by an option statement within a GAMS model, e.g.,

```
option iterlim = 100;
```

Finally, a model attribute can set a GAMS option for an individual model:

```
mymodel.iterlim = 100;
```

The model suffix takes precedence over the option statement, which takes precendence over the command line parameters. If none of these methods is used to set an option, default values apply.

Further, one can unset any model-specific option by assigning it the value `NA`:

```
mymodel.iterlim = NA;
```

Unfortunately, not every option can be via as command line parameter, option statement, and model attribute. We refer to

- Solver-Related Options for the list of solve-related options that can be set via the command line,
- Options that Control Solver-Specific Parameters and Options that Control the Choice of Solver for the list of solve-related options that can be set via the option statement, and
- Model Attributes Mainly Used Before Solve for the list of solve-related model attributes.

4.43.2 The Solver Options File

To specify solver-specific options, it is necessary to use a solver option file. Two things are required to do this: one must create an option file having a proper name, and one must tell the solver to read and use this option file.

To tell a solver to use an option file, one can set the `optfile` model attribute or the optfile option to a positive value. For example,

```
model mymodel /all/;
mymodel.optfile = 1;
solve mymodel using nlp maximizing dollars;
```

The option file takes its name from the solver being used: `solvername.XYZ`, where `solvername` is the name of the solver that is specified, and the suffix `XYZ` depends on the value to which `optfile` has been set. If its value is 1, the suffix is `opt`. For example, the option file when calling CONOPT would be called `conopt.opt`. See the documentation on optfile for more information.

The format of the options file can change marginally from solver to solver. The following illustrates some frequent features of the option file format. However, solvers may vary from this format. Thus, the solver-specific documentation should be checked before using an option file.

- Blank lines in an option file are ignored.
- A comment line might begin with an asterisk (∗) in the first column, is not interpreted by either GAMS or the solver, and is used purely for documentation.
- Each non-comment line contains only one option specification.
- The format for specifying options is as follows:

  ```
  keyword(s) [modifier] [value]
  ```

  The keyword may consist of one or more words and is not case sensitive. The value might be an integer, a real, or a string. Real numbers can be expressed in scientific format, e.g., 1e-6. Note that not all options require modifiers or values.
- Any errors in the spelling of keyword(s) or modifiers will lead to that option being misunderstood and therefore ignored. Errors in the value of an option can result in unpredictable behavior. When detected, errors are either ignored or pushed to a default or limiting value, but not all can or will be detected.

Consider the following CPLEX options file,

```
* CPLEX options file
barrier
crossover 2
```

The first line begins with an asterisk and therefore contains comments. The first option specifies the use of the barrier algorithm to solve the linear programming problem, while the second option specifies that the crossover option 2 is to be used. Details of these options can be found in Summary of CPLEX Options.

Options can also be defined by generating an options file within the GAMS source code. Consider the following code fragment for using MINOS options file,

```
Model m /all/;
Option NLP = MINOS;
* MINOS options file
$onecho > minos.opt
scale option 2
completion partial
$offecho
m.OptFile = 1;
Solve m maximizing z using nlp ;
```

The first option sets the scale option to a value of 2. In this case, the keyword 'scale option' consists of two words. In the second line, the 'completion' option is set to 'partial'. Details of these options can be found in Summary of MINOS Options.

4.43.2.1 Dot Options

Dot options in a solver option file allow users to associate values to variables and equations using the GAMS name of the variables and equations. The general syntax of a dot option in the option file is as follows:

```
(variable/equation name).optionname (value)
```

Dot options can be specified for `all`, a `block`, a `slice`, and a `single` variable and equation. Please note that a specific dot option may only apply to variables or equations (e.g. the GAMS/Gurobi dot option prior applies to variables only). The following example makes the use of the dot option clear.

For example, suppose one has a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
Variable v(i,j);
Equation e(i,j);
```

Consider the following lines in an option file with the imaginary option name `dotopt`:

Line in option file	Explanation
`variables.dotopt 1`	Sets the value of all variables to `1`
`equations.dotopt 2`	Sets the value of all equations to `2`
`v.dotopt 3`	Sets the value of the variables in block `v` to `3`
`e.dotopt(*,*) 4`	Sets the value of the equations in block `e` to `4`
`v.dotopt(*,'j2') 5`	Sets the value of the variables v that have `j2` in the second index position (slice) to `5`
`e.dotopt('i3',*) 6`	Sets the value of the equations e that have `i3` in the first index position (slice) to `6`
`w.dotopt('i2') 7`	Sets the value of the single variables `v('i2')` to `7`
`e.dotopt('i3','j3') 8`	Sets the value of the single equations `e('i3','i3')` to `8`

The values of the dot option are applied in correspondence to the sequence in which they appear in the option file. In the current example, the values of `dotopt` for the equation `e` would be as follows:

e.dotopt	i1	i2	i3
j2	4	4	6
j3	4	4	8
j4	4	4	6

4.43.3 Starting Point and Initial Basis

4.43.3.1 Starting Point

NLP solvers that search for a locally optimal solution of a NLP require an initial point to start their search. Further, the closer this initial point is to a local optimum, the less effort the solver may have to spend. The latter can also be true for solvers that search for global optimal solutions, such as most LP or MIP solver or global MINLP solvers.

Because of this immense importance of a starting point, GAMS always passes a starting point to a solver. By default, the point passed on by GAMS is given by the level and marginal attributes of variables and equations. If these values have not been set yet, default values are used. This default value is zero, except for variables whose bounds would forbid this value. In this case, the bound closest to zero is used.

In addition to setting these values explicitly in a GAMS model, a user can also load them from a save file or a GDX point file via execute_loadpoint. The latter may have been generated by running a related model and using option savepoint. Further, in models with several solve statements, the solution from one solve, if any, is used to initialize the starting point for a succeeding solve. This happens automatically since solutions from a solve statements are also stored in the level and marginal values of variables and equations. Finally, note that model attribute defpoint can be used to force sending the default starting point to a solver.

For some solvers, in particular for MIP or MINLP, an option may have to be set to make use of the starting point. Further, some solvers offer the possibility to make use of a partial starting point or use the starting point as a guide for the search. For details, see the specific solver manuals and look for parameters like `mipstart` and the use of the GAMS parameter tryint.

4.43.3.2 Initial Basis

While for some solvers, the values of a starting point are sufficient to initialize the search, active-set algorithms make use of a different form of starting information. An active-set algorithm tries to identify which constraints are active (or binding) in a feasible or optimal solution, that is, which variables are at one of their bounds and for which equations the activity equals the right-hand-side and the marginal is non-zero. For example, the simplex method for linear programs is an active-set algorithm. The classification of constraints as either active or inactive is closely linked to a *basis*. Active constraints are called *nonbasic* and inactive constraints are called *basic*. A basis that specifies the active and inactive constraints in an optimal solution is called an *optimal basis*.

Active-set algorithms may start by guessing an initial basis and then iteratively updating this basis until an optimal basis is found. Solution time may be reduced substantially if a good approximation of the optimal basis can be identified and passed on to the solver. Such a user-provided initial basis is called an *advanced basis*. However, provision of an advanced basis does not always help. For example, the presolving algorithms used in some solvers may cause the advanced basis to be ignored. Further, solvers may perform poorly when the advanced basis provided is worse than what the solver would otherwise have constructed with its own heuristics. Finally, not all algorithms benefit from the provision of an advanced basis: interior-point algorithms are a notable example.

GAMS provides a hint to the solver to suggest whether a basis can or should be extracted from the initial point. The hint is based on the number of rows (aka single equations) with nonzero marginals and the value of the bratio option and is computed internally as:

```
hint := (0 == bRatio) or (rowsWithNonzeroMarg > nRows * bratio)
```

While the setting of the variable level and marginal values is important in specifying an advanced basis, only the *marginal values for the rows* are relevant in computing the basis hint and in choosing a value for bratio. For example, in a problem with 1000 rows and with the default value of bratio (0.25), GAMS would suggest a basis only if at least 250 nonbasic rows exist.

Note that the default starting point - all marginals zero - is usually not sufficient for the construction of an initial basis. If the automatic transfer of a basis from one solve statement to the next leads to poor behavior in the solver, setting the option bratio to 1 will cause the basis hint to be false and suppress the use of an initial basis. Setting the model attribute defpoint to 1 or 2 will achieve the same result. Conversely, setting bratio to 0 will force the basis hint to be true.

Active-set solvers in GAMS typically use the basis hint value to decide whether to extract (and use) the advanced basis available in the starting point. Details can vary by solver, but typically it works as follows:

- Variables with a zero level value are suggested to be nonbasic if and only if they have a non-zero marginal value.
- Variables with a non-zero level value are suggested to be nonbasic if the level value is equal one of the variable bounds. The marginal values may also be used here: nonbasic variables correspond with non-zero marginal values.
- Equations with non-zero marginal are suggested to be nonbasic. Otherwise, they are suggested to be basic.

Thus, a user can attempt to explicitly provide an initial basis by setting a corresponding starting point. That is, one can set a guess for an initial basis by specifying

- non-zero marginals for equations that are predicted to be active in a solution
- non-zero marginals for variables that are predicted to be at their bound in a solution
- non-zero levels for the variables that are predicted to be non-zero in a solution

4.43.4 Trace Features

Sometimes it might be useful to get certain information about a particular solve statement (or GAMS job) in a compact and customizable format.

- The Trace file facility allows to create files containing information about the "end data" of a GAMS job and the contained solve statements.
- The Solve trace facility can provide more detailed information about the solution progress of a particular solve statement (e.g. objective value of the incumbent solution or the best dual bound every five seconds)

4.43.4.1 Trace File

The **trace file** contains information about the "end data" of a GAMS job and the contained solve statements. Creation of a trace file can be activated via command line parameter trace (see example below). The trace feature supports several predefined formatting options, that is differently formatted trace files can be created, depending on what output information is desired. The trace file format option can be set via command line parameter traceOpt. The trace file header defines the contained trace records and the associated trace record fields.

Note

- Trace information is appended to existing trace files **in the format of the existing trace file**. That is, if a previous trace file of the same name already exists, then all new trace data will be appended in the initial format, no matter if the current `traceopt` value actually implies a different format.
- Trace file headers can be modified, i.e. the predefined formats can be customized and trace record fields can be added or removed as needed.

Trace Records

A trace file can contain different types of trace records:

Trace Record Type	Meaning
`GamsStep`	A `GamsStep` refers to a GAMS execution phase (there are potentially multiple phases of GAMS execution in a single GAMS Job). For example, a trace file created with `traceopt=0` for the [**TRNSPORT**] model, which contains a single solve statement, will contain two GamsSteps, one referring top the GAMS execution phase before the solve statement, the other one referring to the execution phase after the solve statement. Note that this may change depending on the setting of solveLink.
`GamsExit`	`GamsExit` describes the final `GamsStep`.
`GamsSolve`	`GamsSolve` describes a trace record referring to a solve statement. If a file has multiple solve statements, the corresponding trace file can have multiple `GamsSolve` trace records.

Trace Record Fields

For every trace record, the trace file can contain multiple fields from the following list.

Name of Field	Meaning
`CNS`	CNS solver.
`ComputerName`	Computer name.
`Direction`	Direction of optimization: `0=min`, `1=max`.
`DNLP`	DNLP solver.
`EMP`	EMP solver.
`ETAlg`	Elapsed time it took to execute the solve algorithm (see corresponding model attribute etAlg).
`ETSolve`	Elapsed time it took to execute a solve statement in total (see corresponding model attribute etSolve).

Name of Field	Meaning
ETSolver	Elapsed time taken by the solver only (see corresponding model attribute etSolver).
GamsCloseDownTime	The time it takes to close down GAMS (which partly depends on command line parameters) including the time to write the file summary, a GDX file, a (obfuscated) save file, a parameter file.
GamsCompTime	Time of the GAMS compilation phase.
GamsElapsedTime	Elapsed time since the start of a GAMS run in seconds.
GamsElements	Number of labels.
GamsErrorCount	Number of compilation and execution errors.
GamsExecTime	GAMS execution time.
GamsLineNumber	Number of lines. This corresponds to the number of lines in the echo print of the input file.
GamsReturnCode	GAMS return code.
GamsStartupTime	The time it takes to start up GAMS including the time to check and dump command line parameters, initialize certain internal data structures, loading a work file, reading a license file.
GamsSymbols	Number of symbols (also called identifiers) including names of intrinsic functions and predefined symbols.
GamsTotalTime	Total time of the corresponding trace record. Does only apply for trace records of type GamsStep and GamsExit.
GamsVersionID	String that contains a GAMS version ID, e.g. WEX282-282 for the Windows 64bit version of GAMS 28.2.
InputFileName	GAMS model filename.
JobDate	Day when the job started.
JobTime	Time when the Job started.
JulianDate	Julian date number with start day/time of job.
LP	LP solver.
Marginals	Indicates availability of marginals: 0=no, 1=yes.
MCP	MCP solver.
MINLP	MINLP solver.
MIP	MIP solver.
MIQCP	MIQCP solver.
ModelGenerationTime	Model generation time. Refers to the last model in the part of the program that corresponds to the trace record.
ModelName	Model name. Refers to the last model in the part of the program that corresponds to the trace record.
ModelStatus	Model status. Refers to the last model in the part of the program that corresponds to the trace record.
ModelType	Model type. Refers to the last model in the part of the program that corresponds to the trace record.
MPEC	MPEC solver.
NLP	NLP solve.
NumberOfDiscreteVariables	Number of discrete variables. Refers to the last model in the part of the program that corresponds to the trace record.
NumberOfDomainViolations	Number of domain violations (see also model attribute domUsd).
NumberOfEquations	Number of equations. Refers to the last model in the part of the program that corresponds to the trace record.
NumberOfInstructions	Length of non-linear code.
NumberOfIterations	Number of iterations. Refers to the solve of the last model in the part of the program that corresponds to the trace record.

Name of Field	Meaning
NumberOfNodes	Number of nodes. Refers to the solve of the last model in the part of the program that corresponds to the trace record. Only relevant if some tree search algorithm (e.g. branch-and-bound was used).
NumberOfNonlinearNonZeros	Number of non-zeroes. Refers to the last model in the part of the program that corresponds to the trace record.
NumberOfNonZeros	Number of variables. Refers to the last model in the part of the program that corresponds to the trace record.
NumberOfVariables	Number of variables. Refers to the last model in the part of the program that corresponds to the trace record.
ObjectiveValue	Objective value. Refers to the solve of the last model in the part of the program that corresponds to the trace record.
ObjectiveValueEstimate	Estimate of the best possible solution for a mixed-integer model (aka best bound). Refers to the solve of the last model in the part of the program that corresponds to the trace record.
OptionFile	Solver option file number. Refers to the solve of the last model in the part of the program that corresponds to the trace record.
Platform	Platform ID, e.g. `WEX` for the Windows 64bit.
QCP	QCP solver.
RMINLP	RMINLP solver.
RMIP	RMIP solver.
RMIQCP	RMIQCP solver.
RMPEC	RMPEC solver.
SolveLine	Line number of the last solve statement in the part of the program that corresponds to the trace record. This corresponds to the line number in the echo print of the input file.
SolveNumber	Number of solve statement for a particular model. For example, `solve myModelA...; solve myModelB...; solve myModelA...;` results in solve numbers `1, 1, 2`.
SolverCalcTime	Time spent in function and derivative calculations (deprecated, see also model attribute resCalc).
SolverElapsedTime	Elapsed time taken by the solver when solveLink=0 is used (almost identical to model attribute etSolver).
SolverID	Solver ID number.
SolverName	Name of the solver. Refers to the solve of the last model in the part of the program that corresponds to the trace record.
SolverReadTime	Time to import model (deprecated, see also model attribute resIn).
SolverRealTime	Elapsed time taken by the solver when solveLink=0 is used (almost identical to model attribute etSolver).
SolverSignature	String containing a solver signature, e.g. `IBM ILOG CPLEX 28.2.0 r750fa45 Released Aug 19, 2019 WEI x86 64bit/MS Window`. Refers to the solver used to solve the last model in the part of the program that corresponds to the trace record.
SolverStatus	Solver Status. Refers to the solve of the last model in the part of the program that corresponds to the trace record.
SolverTime	Time used to solve the model in seconds reported by the solver (see also model attribute resUsd).
SolverVersion	Solver Version (see also model attribute sysVer).
SolverWriteTime	Time to export solution (deprecated, see also model attribute resOut).
User1	Content of user1 string.
User2	Content of user2 string.
User3	Content of user3 string.

Name of Field	Meaning
User4	Content of user4 string.
User5	Content of user5 string.
UserName	User name.

Trace Report

GAMS can create **trace reports** from trace files, if started with command line parameter action=gt. The following sequence of commands loads model [**TRNSPORT**] from the GAMS model library, runs trnsport.gms and creates a tracefile trc.txt, and produces a trace report plus a trace summary from the trace file.

```
$ gamslib trnsport
$ gams trnsport trace=trc.txt
$ gams trc.txt a=gt
```

The **trace file** trc.txt contains the default trace record fields for the GamsStep and GamsSolve trace records:

```
* Trace Record Definition
* GamsStep
* JobDate JobTime InputFileName GAMS GamsVersionID GamsReturnCode GamsErrorCount GamsStartupTime Gams
* GamsSolve
* JobDate JobTime InputFileName ModelType SolverName SolverStatus ModelStatus SolveNumber SolveLine S
*  NumberOfIterations ModelGenerationTime SolverReadTime SolverCalcTime SolverWriteTime ObjectiveValu
*
09/25/19 07:04:00 trnsport.gms GAMS WEX282-282 1 0 0.015 0 0 0.016 0.031 ""
09/25/19 07:04:00 trnsport.gms LP CPLEX 1 1 1 64 23 NA 6 7 19 0 0 4 0 NA NA NA 153.675 ""
09/25/19 07:04:00 trnsport.gms GAMS WEX282-282 0 0 0 0 0 0 0 ""
```

The **trace summary** trc.sum contains condensed information about the trace records:

```
Trace Summary with TL=0 for [...]\trc.txt
Trace Records = 3, Error Records = 0, First Date = 09/25/19 07:04:00
```

The **trace report** is created in trc.lst and contains detailed information about the trace records from the underlying trace file including a ModelSolveStat matrix that assigns a score from 0 to 9 to all potential combinations of model status and solver status:

```
[...]
Trace Report for File: [...]\trc.txt


Using ModelSolveStat for TL=0

     1  2  3  4  5  6  7  8  9 10 11 12 13
  1  9  .  .  .  .  .  .  .  .  .  .  .  .
  2  9  9  9  9  9  .  .  9  .  .  .  .  .
  3  9  .  .  .  .  .  .  .  .  .  .  .  .
  4  9  .  .  .  .  .  .  .  .  .  .  .  .
  5  9  .  .  .  .  .  .  .  .  .  .  .  .
```

```
   6  .  5  5  5  5  .  .  5  .  .  .  .  .
   7  9  9  9  9  9  .  .  9  .  .  .  .  .
   8  9  9  9  9  9  .  .  9  .  .  .  .  .
   9  .  5  5  5  5  .  .  5  .  .  .  .  .
  10  9  .  .  .  .  .  .  .  .  .  .  .  .
  11  .  .  .  .  .  .  5  .  .  .  .  .  .
  12  .  .  .  .  .  .  .  .  .  .  3  .  3
  13  .  .  .  .  .  .  .  .  3  3  .  3  .
  14  4  4  4  4  4  6  .  4  .  .  .  3  .
  15  9  .  .  .  .  .  .  .  .  .  .  .  .
  16  9  .  .  .  .  .  .  .  .  .  .  .  .
  17  9  .  .  .  .  .  .  .  .  .  .  .  .
  18  9  .  .  .  .  .  .  .  .  .  .  .  .
  19  9  .  .  .  .  .  .  .  .  .  .  .  .


Marked trace records are listed below:

Record  Date     Time     Filename        Type    Solver       Message
        * Trace Record Definition
        * GamsStep
        * JobDate JobTime InputFileName GAMS GamsVersionID GamsReturnCode GamsErrorCount GamsStartup
        * GamsSolve
        * JobDate JobTime InputFileName ModelType SolverName SolverStatus ModelStatus SolveNumber Sol
        *  NumberOfIterations ModelGenerationTime SolverReadTime SolverCalcTime SolverWriteTime Objec
        *

First time stamp = 09/25/19 07:04:00
Last  time stamp = 09/25/19 07:04:00

Total trace records   =     3
Total comment records =     7



Total GAMS records  =     2  all return codes

                          1  RC= 0  Normal completion
                          1  RC= 1  Executing subsystem

Time (sec)             total        mean
   start                0.01        0.01
   compile              0.00        0.00
   execute              0.00        0.00
   closedown            0.02        0.01
   total                0.03        0.02



Total SOLVE records =     1  all return codes

      solvestatus         1  RC= 1  1 Normal Completion

      modelstatus         1  RC= 1  1 Optimal

Numbers                total        mean
   equations              6        6.00
   variables              7        7.00
```

```
   nonzeros               19       19.00
     nonlinear             0        0.00
   instruction             0        0.00
   equations               6        6.00
Time (sec)              total        mean
   generation            0.00        0.00
   input                 0.00        0.00
   calculation           0.00        0.00
   output                0.00        0.00
   total solve           0.00        0.00


SOLVESTAT Cross Tabulation

             1    2    3    4    5    6    7    8    9   10   11   12   13    tot
     LP      1    .    .    .    .    .    .    .    .    .    .    .    .      1
  total      1    0    0    0    0    0    0    0    0    0    0    0    0      1


MODELSTAT Cross Tabulation

             1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18
     LP      1    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .
  total      1    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0


GTRACE      TIME     =        0.000 SECONDS        VERID WEX282-282
[...]
```

The trace report may be particularly interesting, if the underlying trace file contains multiple trace records from different solve statements. In combination with command line parameter traceLevel, it is easy to filter for `GamsSolve` trace records with a low `ModelSolveStat` score, i.e. undesirable combinations of model status and solver status. If we run [**TRNSPORT**] again with an iteration limit of 0 (remember that the trace information will be appended to the trace file from the example above) and recreate the trace report as follows

```
$ gams trnsport trace=trc.txt iterlim=0
$ gams trc.txt a=gt
```

the trace report shows two solve records and the obtained statuses:

```
Total SOLVE records =     2  all return codes

      solvestatus         1  RC= 1  1 Normal Completion
      solvestatus         1  RC= 2  2 Iteration Interrupt

      modelstatus         1  RC= 1  1 Optimal
      modelstatus         1  RC= 6  6 Intermediate Infeasible
```

A look at the `ModelSolveStat` matrix from the initial example, tells us that a combination of model status 6 (Intermediate Infeasible) in combination with solve status 2 (Iteration Interrupt) results in a score of 5. If we increase the traceLevel to a value greater than or equal to 5 and recreate the trace report again, e.g. via

```
$ gams trc.txt a=gt tracelevel=5
```

the ModelSolveStat matrix at the beginning of the trace report changes such that only the scores greater than 5 for accepted combinations of model and solver status are shown

```
Using ModelSolveStat for TL=5

     1  2  3  4  5  6  7  8  9 10 11 12 13
  1  9  .  .  .  .  .  .  .  .  .  .  .  .
  2  9  9  9  9  9  .  .  9  .  .  .  .  .
  3  9  .  .  .  .  .  .  .  .  .  .  .  .
  4  9  .  .  .  .  .  .  .  .  .  .  .  .
  5  9  .  .  .  .  .  .  .  .  .  .  .  .
  6  .  .  .  .  .  .  .  .  .  .  .  .  .
  7  9  9  9  9  9  .  .  9  .  .  .  .  .
  8  9  9  9  9  9  .  .  9  .  .  .  .  .
  9  .  .  .  .  .  .  .  .  .  .  .  .  .
 10  9  .  .  .  .  .  .  .  .  .  .  .  .
 11  .  .  .  .  .  .  .  .  .  .  .  .  .
 12  .  .  .  .  .  .  .  .  .  .  .  .  .
 13  .  .  .  .  .  .  .  .  .  .  .  .  .
 14  .  .  .  .  .  6  .  .  .  .  .  .  .
 15  9  .  .  .  .  .  .  .  .  .  .  .  .
 16  9  .  .  .  .  .  .  .  .  .  .  .  .
 17  9  .  .  .  .  .  .  .  .  .  .  .  .
 18  9  .  .  .  .  .  .  .  .  .  .  .  .
 19  9  .  .  .  .  .  .  .  .  .  .  .  .
```

and GAMS throws an execution error

```
--- Starting trace report generation
*** Status: Execution error(s)
```

The trace file facility in combination with the trace report can be particularly useful for automated quality assurance tests where certain expected outcomes should be audited.

4.43.4.2 Solve trace

In order to do accurate performance evaluations it may be useful to obtain more detailed information about a solve than the "end data" that the trace file provides. E.g., for a branch-and-bound based solver, one may want to have intermediate information about the values of primal and dual bounds at the root node and subsequent nodes within the search.

The **solve trace** option that is implemented in some of the GAMS solver interfaces allows users to output solve information, e.g., primal and dual bounds, for every n nodes or at every time step. For example, the user may be interested in the objective value of the incumbent solution or the best dual bound on the optimal value every 50 nodes and every five seconds of the solve.

Note

The solve trace file format and options may change in a future GAMS release.

The solve trace option is invoked via a GAMS solver options file. Usually, options to specify a filename of the trace file to be created and options to specify time and node intervals are available. Please refer to the GAMS solver manuals for the exact names of these options (search for solvetrace or miptrace).

The solve trace file is written in comma-separated-value (CSV) format, where the entries in each line have the following meaning:

Column Name	Meaning
lineNum	a line index
seriesID	indicator why the line was written: S = start of search, N = node frequency, T = time frequency, E = end of search
node	number of enumerated branch-and-bound nodes
seconds	time since the solving started
bestFound	primal bound, i.e., objective value of incumbent solution
bestBound	dual bound, i.e., bound on optimal value

A sample solve trace file including statistics of a GAMS run using the MIP model [**DICE**] and the solver XPRESS looks as follows:

```
* miptrace file miptrace.mtr: ID = XPRESS.1 Instance = dice
* fields are lineNum, seriesID, node, seconds, bestFound, bestBound
1, S, 1, 0.078, na, na
2, N, 101, 0.781, 21, 23.9369
3, N, 201, 0.875, 21, 23.5564
4, N, 304, 1.031, 21, 23.5564
5, E, 399, 1.094, 21, 21
* miptrace file miptrace.mtr closed
```

See also the slides for the presentation Advanced Use of GAMS Solver Links (2013) and the accompanying scripts for some ideas on what to do with the solve trace functionality.

4.43.5 Branch-and-Cut-and-Heuristic Facility (BCH)

Global search algorithms can sometimes significantly benefit from user supplied routines that support the solution process of an hard optimization problem. For example, branch-and-cut solvers (e.g., CPLEX, Gurobi, SCIP, Xpress) can profit from user-supplied *cutting planes* or *good feasible solutions*. GAMS users could supply these as part of the model given to the solver, by adding a set of constraints representing likely to be violated cuts and an initial solution (possibly in combination with GAMS parameters like tryint and solver-specific options like mipstart in CPLEX). However, this does not allow a dynamic interaction between a running solver and user supplied routines that, for example, use a current relaxation solution to construct cutting planes or feasible solutions. The GAMS Branch-and-Cut-and-Heuristic (BCH) facility attempts to automate all major steps necessary to make callbacks that certain solvers provide for such usage available to the GAMS user. This allows GAMS users to apply complex solution strategies without having to have intimate knowledge about the inner workings of a specific solver.

Currently, only two solvers support the BCH facility: CPLEX and SBB. With GAMS/CPLEX, user supplied GAMS programs that implement primal heuristics and cut generation can be used. With SBB, only primal heuristics are possible.

As the name indicates, the BCH facility has been designed with the solving process of a branch-and-cut solver (e.g., CPLEX, Gurobi, SCIP, Xpress) in mind. Such solvers often allow to call a user supplied routine after a node in the branch-and-bound (B&B) tree has been processed. Within that routine, available information like the solution of a relaxation (often an LP or NLP) at that node and the current incumbent, if any, is exported by the BCH facility into a GDX file using the original GAMS namespace. Next, different user supplied GAMS programs can be called, e.g., for finding cuts which are violated by the relaxation solution (cut generator) or to find new incumbents (primal heuristic). These GAMS programs should import the information from the GDX file and do their computations. After termination, the BCH facility resumes control, reads the findings from the GAMS program and passes them to the solver.

A relaxation solution may be exported into a file `bchout.gdx` by the BCH facility. This GDX file does not only contain the variable values as level values (`.l`), but also variable bounds (`.lo` and `.up`). For a B&B solver, these are the local bounds at this node. Hence, they reflect branching decisions made in the B&B tree and bound tightenings that were deduced by the solver. In a similar way, the BCH facility may export an incumbent solution to the GDX file `bchout_i.gdx`. The bounds for the incumbent solution reflect global bounds, i.e., the original bounds, possibly tightened by the solver. GDX files can be imported by the GAMS program using the compile time $load or run time execute_load.

The BCH facility is activated and controlled by setting certain options in the solvers options file. The precise names and meanings of the options may vary from one solver to another. Therefore, also the corresponding GAMS solver manual should be checked. The options that come with the BCH facility can be used to define the calls of the users GAMS programs, to determine when they should be called, and to overwrite the filenames for the GDX files (to avoid name clashes). General BCH related options are the following:

Name	Description	Default
`UserGDXIn`	The name of the GDX file read back into the solver.	`bchin.gdx`
`UserGDXName`	The name of the GDX file exported from the solver with the solution at the node.	`bchout.gdx`
`UserGDXNameInc`	The name of the GDX file exported from the solver with the incumbent solution.	`bchout_i.gdx`
`UserGDXPrefix`	Prefix to add to `UserGDXIn`, `UserGDXName`, and `UserGDXNameInc`	*empty*
`UserJobID`	Postfix to add to listing and log filenames and to `UserGDXIn`, `UserGDXName`, and `UserGDXNameInc`. Further, `--UserJobID` is added to calls to users GAMS programs.	*empty*
`UserKeep`	Calls users GAMS programs with `gamskeep` instead of `gams`. The use of `gamskeep` will preserve the scratch directory similar to the command line option keep	0

In the following, the interface for the available callbacks are explained in more detail and corresponding options are listed.

4.43.5.1 Primal Heuristics

In the primal heuristic callback, the user can provide a GAMS program which tries to construct a feasible solution based on the information provided by the solver, e.g., a current relaxation solution and the current incumbent. Thus, the GAMS program could attempt to repair infeasibilities in the relaxation solution or try to improve the incumbent from the solver.

If the GAMS program finds a new solution, it should store it in the level values of variables that correspond to the original variables. For example, if the original model uses binary variable `open(i,t)`, then at the end of the GAMS program `open.l(i,t)` should contain a 0 (zero) or a 1 (one). The BCH facility calls the GAMS program and instructs GAMS to store the results in a GDX file at termination. This GDX file is then read in again by the BCH facility and the solution is passed back to the solver. The solver checks this solution for infeasibilities and in case this check is passed and the solution is better than the best known solution, the solver updates it's incumbent.

If the GAMS program cannot find a feasible solution, it can terminate with an execution error triggered by an abort statement to prevent the BCH facility from reading the results from the heuristic run.

BCH parameters to control the primal heuristic call are typically the following:

Name	Description	Default
`UserHeurFreq`	Determines the frequency of the heuristic call.	10

Name	Description	Default
UserHeurMult	Determines the multiplier for the frequency of the heuristic call.	2
UserHeurInterval	Determines the interval when to apply the multiplier for the frequency of the heuristic call. For example, for the first 100 (UserHeurInterval) nodes, the solver calls the heuristic every 10th (UserHeurFreq) node. After 100 nodes, the frequency gets multiplied by 10 (UserHeurMult), so that for the next 100 nodes the solver calls the heuristic every 20th node. For nodes 200-300, the heuristic get called every 40th node, for nodes 300-400 every 80th node and after node 400 every 100th node.	100
UserHeurFirst	For how many of the first nodes the heuristic should be called.	10
UserHeurObjFirst	Similar to UserHeurFirst, but specifices for how many of the first nodes the heuristic should be called if the optimal value of the current nodes relaxation promises a significant improvement of the current incumbent, i.e., the optimal value of the relaxation at the node has to be closer to the current dual bound than the current primal bound.	*solver dependent*
UserHeurNewInt	Whether to calls the heuristic when the solver found a new feasible solution.	no
UserHeurCall	Arguments to the GAMS call to invoke the heuristic GAMS program.	*empty*

As an example, for the Oil Pipeline Network Design problem, the BCH options to invoke the primal heuristic in the GAMS program bchoil_h.inc when using GAMS/CPLEX could be

```
userheurcall     bchoil_h.inc mip cplex optcr 0 reslim 10
userheurfirst    5
userheurfreq     20
userheurinterval 1000
```

4.43.5.2 Cutting Planes

In the cut generator callback, the user can provide a GAMS program which tries to find a linear cut (that is, a linear inequality) that is violated by the relaxation solution. The solver would then add these cuts to it's *cut pool*. Typically, it then resolves the relaxation at the node and calls the cut generator again. If no cutting planes are found, the solver will continue, e.g., by processing the next node. Please note that the solver cannot perform validity checks on the provided cuts. Hence, it is possible to cut off areas of the feasible region, including optimal solutions.

Exporting cuts is a little more complicated than a solution because next to the cut coefficients, also the sense and the right-hand-side of the cut inequality needs to be exported. Further, exporting several cuts with one call should be possible. For this purpose, the GAMS program has to define and fill a set cc and parameters numcuts, rhs_c(cc), and sense_c(cc) appropriately. The set cc is used as a cut index. It can be larger than the number of actually generated cuts.

Note

The elements of the cut index set must form a series of integers starting at 1 (1, 2, 3,...).

Parameter numcuts should specify the number of added cuts. rhs_c(cc) should store the right-hand-side of each cut. Finally, sense_c(cc) should store the sense of each cut, which must be 1 for lower-equal ($\leq$), 2 for equal ($=$, rather unusual for cuts), and 3 for greater-equal ($\geq$). The corresponding declaration in GAMS code may be

```
$set MaxCuts 100
Set        cc          'cuts'  / 1 * %MaxCuts% /;
Parameters numcuts     'number of cuts to be added' / 0 /
           rhs_c(cc)   'cut rhs'
           sense_c(cc) 'the sense of the cuts';
```

The only thing missing are the cut coefficients. As it should be possible to return more than one cut, using variable attributes like level values is not sufficient. Therefore, for each variable that is part of a cut, a new parameter must be added in the GAMS program. The name of the parameter must be the name of the corresponding variable with an additional `_c` at the end. Further, the parameter must be indexed like the variable, but with the cut index set cc added at the beginning. For example, assume variable `open(i,t)` should be part of a cut. Then the cut coefficients should be stored in a parameter `open_c(cc,i,t)`, e.g.,

```
Parameter  open_c(cc,i,t) 'coefficients of variable open(i,t) in cut cc';
```

The BCH facility reads all parameters that end in `_c`, takes the base name and looks for a variable with that name and indices and builds up the cut matrix. A cut cannot introduce a new variable into the model. All cuts added to the model are assumed to be *global cuts*, that is, they need to be valid for the entire problem, not just for the current node.

BCH parameters to control the cut generation call are typically the following:

Name	Description	Default
`UserCutFreq`	Determines the frequency of the cut generator call.	10
`UserCutMult`	Determines the multiplier for the frequency of the cut generator call.	2
`UserCutInterval`	Determines the interval when to apply the multiplier for the frequency of the cut generator call. See UserHeurInterval for details.	100
`UserCutFirst`	Calls the cut generator for the first n nodes.	10
`UserCutNewInt`	Whether to call the cut generator if the solver found a new integer feasible solution.	no
`UserCutCall`	Arguments to the GAMS call to invoke the cut generator GAMS program.	*empty*

As an example, for the Oil Pipeline Network Design problem, the BCH options to invoke the cut generator in the GAMS program `bchoil_c.inc` when using GAMS/CPLEX could be

```
usercutcall   bchoil_c.inc mip cplex
usercutfirst  0
usercutfreq   0
usercutnewint yes
```

4.43.5.3 Incumbent Callbacks

The incumbent callbacks can be used to execute a GAMS program when the solver found a new feasible solution that improves the incumbent. Additionally, the incumbent check callback `UserIncbCall` can be used to notify the solver whether the given feasible solution should be accepted by the solver. This allows to implement a filtering mechanism that forces a solver to search for additional solutions even though an optimal solution might have been found already. Furthermore, the callback `UserLazyConCall` allows to add lazy constraints after an incumbent has been found.

The following parameters control the incumbent callbacks:

Name	Description	Default
UserIncbCall	Arguments to the GAMS call to invoke the incumbent checking GAMS program. The incumbent is rejected if the GAMS program terminates normally. In case of a compilation or execution error, the incumbent is accepted.	*empty*
UserIncbICall	Arguments to the GAMS call to invoke the incumbent reporting GAMS program.	*empty*
UserLazyConCall	Arguments to the GAMS call to invoke the lazy constraint adding GAMS program. Lazy constraints that cut off the incumbent are expected (in a format similar to UserCutCall) if the GAMS program terminates normally. In case of a compilation or execution error, the incumbent is accepted.	*empty*

4.43.5.4 Examples

The GAMS model library contains a few examples to show to use the BCH facility:

- **bchtlbas.gms : Trim Loss Minimization with Heuristic using BCH Facility** This model implements a very simple LP/MIP based primal heuristic for the trimloss minimization problem.
- **bchfcnet.gms : Fixed Cost Network Flow Problem with Cuts using BCH Facility** This model implements simple but difficult to separate cuts for a network design problem. The global solver BARON is used to find violated cuts by solving a non-convex MINLP.
- **bchmknap.gms : Multi knapsack problem using BCH Facility** This model implements simple cover inequalities for the multi-knapsack problem.
- **bchoil.gms : Oil Pipeline Design Problem using BCH Facility** This is the most complex example. It implements three different primal heuristics: an initial heuristic based on a simplified cost structure, a rounding heuristic, and a local branching heuristic. In addition, complex cuts are generated by solving regionalized versions of the original problem.
- **dicegrid.gms : MIP Decomposition and Parallel Grid Submission - DICE Example** This example uses many of the UserJobID option to rename files, since running multiple jobs in parallel requires the use of different filenames. This example also uses the incumbent reporting call UserIncbICall.
- **solnpool.gms : Cplex Solution Pool for a Simple Facility Location Problem** This example uses the incumbent checking call UserIncbCall as an advanced filter for accepting or rejecting solutions found by CPLEX.

4.43.6 Choosing an appropriate Solver

For any of the GAMS problem classes (LP, MCP, MINLP, ...), there is no solver that is best on every problem instance. Below, we provide some links to rules of thumb on choosing a solver or solver comparisons.

- GAMS Blog: An Overview of Math Programming Solvers
- T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. Bixby, E. Danna, G. Gamrath, A. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. Steffy, K. Wolter, MIPLIB 2010, Mathematical Programming Computations, 3:2 (2011) 103-163. preprint
- M. Bussieck and S. Vigerske, MINLP Solver Software (2014).
- SNOPT vs. IPOPT: P. Gill, M. Saunders, E. Wong, On the Performance of SQP Methods for Nonlinear Optimization, 2015
- H. Mittelmann: Decision Tree for Optimization Software and Benchmarks for Optimization Software

4.43.6.1 Relative Merits of MINOS and CONOPT

How to choose between MINOS and CONOPT

It is almost impossible to predict how difficult it is to solve a particular model. The best and most reliable way to find out which solver to use is to try out both. However, there are a few rules of thumb:

CONOPT 3 is well suited for models with very non-linear constraints. If you experience that MINOS has problems achieving feasibility during the optimization, you should try CONOPT. On the other hand, if your model has few nonlinearities outside the objective function, MINOS and QUADMINOS is probably the best solver.

CONOPT is has a fast method for finding a first feasible solution that is particularly well suited for models with few degrees of freedom (this means: the number of variables is approximately the same as the number of constraints - in other words, models that are almost square). In these cases CONOPT is likely to outperform MINOS while for models with many more variables than equations MINOS is probably more suited.

CONOPT has a preprocessing step in which recursive equations and variables are solved and removed from the model. If you have a model where many equations can be solved one by one, CONOPT will take advantage of this property. Similarly, intermediate variables only used to define objective function terms are eliminated from the model and the constraints are moved into the objective function.

CONOPT has many built-in tests (e.g. tests for detecting poor scaling). Many models that can be improved by the modeler are rejected with a constructive message. CONOPT is therefore a useful diagnostic tool during model development even if another solver is used for the production runs.

Why serious NLP modelers should have both MINOS and CONOPT

It is almost impossible to predict how difficult it is to solve a particular model. However, if you have two solvers, you can try both. The overall reliability is increased and the expected solution time will be reduced.

On a test set of 196 large and difficult models, many poorly scaled or without initial values, both MINOS and CONOPT failed on 14 models. However only 4 failed on both MINOS and CONOPT. So the reliability of the combined set of solvers is much better than any individual solver.

Many examples of poorly formulated models were observed on which MINOS failed. CONOPT rejected many of the models, but with diagnostic messages pinpointing the cause of the problem. After incorporating the changes suggested by CONOPT, both MINOS and CONOPT could solve the model. Switching between the two solvers during the initial model building and debugging phase can often provide useful information for improving the model formulation.

Special Offer for two NLP Solvers

In order to encourage modelers to have two NLP solvers, GAMS offers a 50% discount on the second solver when both MINOS and CONOPT are purchased together.

4.43.6.2 PATH versus MILES

This document describes some of the differences between the MCP solvers PATH and MILES. MILES is a free solver, that comes with the GAMS/BASE module, while PATH is an optional solver, that is charged for separately.

PATH and MILES are two GAMS solvers capable of solving mixed nonlinear complementarity problems (MCP). Both solvers are based on the sequential linear complementarity algorithm, i.e., they both solve a sequence of linear mixed complementarity problems whose solutions typically converge to the solution of the MCP. To solve each of the linear subproblems (major iterations), both codes use a generalization of an algorithm due to Lemke that is based on a sequence of pivots (minor iterations) similar to those generated by the simplex method for linear programming. To do these pivots efficiently, both codes use the same sparse linear algebra package.

As a result of the above similarities, the performance of the two codes is comparable for many "easy" models. Viewed over a broad range of problems, however, PATH is typically faster and more robust than MILES. While both codes solve all the MCP and MPSGE models in GAMSLIB, PATH significantly outperforms MILES on the MCPLIB test collection found at CPNET.

Most sophisticated MCP and MPSGE modelers prefer to use PATH over MILES. PATH has a crashing scheme that allows it to quickly improve the user given starting point before starting to solve the linear subproblems. This frequently speeds up solution time. PATH automatically attempts to fix "singular" models using a technique based on proximal perturbations. In many cases, this enables the linear subproblems to be solved, leading to a model solution. This typically helps modelers at model development time.

PATH has many more solution options to enable it solve difficult models. The code automatically tries useful options on difficult problems using a restart procedure. PATH has a much more sophisticated "globalization" procedure that typically improves speed and robustness. PATH implements a nonmonotone watchdog technique. Stalling is frequently circumvented by allowing larger steps to be taken toward solutions.

PATH has many more diagnostic features that help uncover problems in a model. In particular, singularities in the model, zero rows and columns and several measures of optimality are returned to the user. Theoretically, PATH has better convergence properties than MILES. In particular, new merit functions are known to allow more reliable and faster convergence.

4.44 The Save and Restart Feature

GAMS saves the information provided in input files in intermediate - mostly binary - files, called *work files* or *scratch files*. Some of these files are used to exchange information between GAMS and the solvers. They are usually deleted just before a GAMS run is complete. However, these intermediate files may be used to process an input file, save the result and later reload this file and continue with processing another input file. Thus, input files may be processed sequentially through the use of the intermediate files. This is a useful feature that can help to reduce the required time when, e.g., several runs of similar models are being made, all of them sharing an equal large initialization part.

The same process may be described in a different way. Assume a large GAMS program is run and an output file is generated, as usual. Suppose the large program is split in two parts. The first part is run and the resulting work file is saved along with the resulting listing file. Then the second part is run after reading in the data from the work file saved previously and a new listing file is generated for the second part. The content of the two listing files will be the same as the content of the output from the very first run when the large program was processed without interruption. Only the arrangement of the content will slightly differ. Splitting the files makes it possible to interrupt a GAMS task and restart it later without loss of information. Furthermore, changes could be made or errors corrected in the later parts.

Note

- The work file preserves all information (including declarations, values, option settings and dollar control options) known to GAMS at the end of the run.
- The work file is not machine specific and thus is portable between platforms. However, a work file that has been generated with one version of GAMS cannot be used for a restart with an older GAMS version (unless command line parameter forceWork is set).
- Absolute or relative path names (e.g. for put files) that do not exist or have the wrong directory separator may cause execution errors. Moreover, if the code in the work file executes a program that might not exist on another platform (e.g. gdxxrw on Linux) the GAMS execution might also result in execution errors.

This chapter illustrates the basics of the save and restart feature in Section Basic Usage and presents some use cases in Section Use Cases. Preventing unauthorized access to and obfuscating the content of work files is discussed in Sections Secure Work Files and Obfuscated Work Files, respectively. An overview of all command line parameters for saving and restarting in GAMS is given in Section Save and Restart Options (chapter The GAMS Call and Command Line Parameters).

4.44.1 Basic Usage

The mechanism to break up the compilation of a large input file into many components and stages is provided by the command line parameters save and restart.

The following demonstrates saves and restarts with the well known transportation model [**TRNSPORT**]. First, the code is split into three parts, resulting in the files `tranmodel.gms`, `transolve.gms`, and `tranreport.gms`. The file `tranmodel.gms` contains the first part of the model up to and including the model statement:

```
Sets
   i   "canning plants"   / seattle, san-diego /
   j   "markets"          / new-york, chicago, topeka / ;

Parameters
   a(i)  "capacity of plant i in cases"
      /    seattle     350
           san-diego   600  /

   b(j)  "demand at market j in cases"
      /    new-york    325
           chicago     300
           topeka      275  / ;

Table d(i,j)  "distance in 1000 miles"
                  new-york       chicago      topeka
   seattle          2.5           1.7          1.8
   san-diego        2.5           1.8          1.4  ;

Scalar f  "freight in dollars/case per 1000 miles" /90/ ;

Parameter c(i,j)  "transport cost in $1000/case" ;
c(i,j) = f * d(i,j) / 1000 ;

Variables
    x(i,j)  "shipment quantities in cases"
    z       "total transportation costs in 1000$" ;
```

```
Positive Variable x ;

Equations
    cost        "define objective function"
    supply(i)   "observe supply limit at plant i"
    demand(j)   "satisfy demand at market j" ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;
```

The file `transolve.gms` contains the solve statement:

```
solve transport using lp minimizing z;
```

And the third file, `tranreport.gms`, contains the display statement:

```
display x.l, x.m ;
```

Observe that concatenating the three files (in the right order) results in the original model [**TRNSPORT**].

4.44.1.1 Saving The Work File

The information in `tranmodel.gms` may be stored by using the following call to GAMS:

```
> gams tranmodel save=s1
```

This command line statement creates the output file `tranmodel.lst` and the work file `s1.g00` in the current working directory.

Note that the command line parameter `s` is a synonym to save.

4.44.1.2 Restarting from the Work File

Consider the following call:

```
> gams transolve restart=s1
```

GAMS reads the work file named `s1.g00` and regenerates the information stored in `tranmodel.gms`. Then `transolve.gms` is run and the result is as if the a concatenation of `tranmodel.gms` and `transolve.gms` had been executed. Note, that the output file `tranmodel.lst` will contain the echo print of the model and `transolve.lst` will contain the echo print of the solve statement and all the output generated by the solve.

Note that the command line parameter `r` is a synonym of restart.

Observe that a restarted run always requires a GAMS input file to continue with. The restart does not alter work files. They may be used repeatedly to continue a particular run many times, possibly with many different continuation input files.

4.44.1.3 A Sequence of Saves and Restarts

In case there are more than two files that should be run sequentially, the second and any other non-final run will have to generate input files for a following restart. Therefore, their workfiles need to be saved.

Following the splitup of model [**TRNSPORT**] into the three files `tranmodel.gms`, `transolve.gms`, and `tranreport.gms`, a sequence of GAMS calls that would run the whole `trnsport` model could be as follows:

```
> gams tranmodel       s=s1
> gams transolve  r=s1 s=s2
> gams tranreport r=s2
```

The listing file of `tranreport.gms` will contain the compilation output with the echo print of the display statement, the final execution summary with the output generated from the display statement and the file summary. The listing files of all three input files together will have the same content as `trnsport.lst` (generated by a run of `trnsport.gms`).

Observe that the three files could also have been processed with a sequence of $include file statements. The advantages of using the `save` and `restart` feature instead are given in section Use Cases below.

4.44.1.4 Avoiding Common Mistakes

A common mistake that occurs when using the save and restart feature is running GAMS on the same file twice, e.g.,

```
> gams trnsport s=trans
> gams trnsport r=trans
```

In this case all the data and equation definitions are repeated, which will cause compilation errors for the second run as in GAMS each data item may be defined only once.

Further, it is the responsibility of the modeler to ensure that the contents of the input file matches that of the work file, although the compiler will issue errors if it detects any inconsistencies, such as references to symbols not previously declared.

4.44.1.5 Prefixing Symbols in the Work File

Assume writing some reporting code that works off a restart file and new symbol names that have not been used in the previous program are required. GAMS offers a convenient and systematic way to achieve this by prefixing all symbols in the work file with a specified string. Consider the following example, again using the transportation model [**TRNSPORT**]:

```
> gams trnsport s=prefixed symPrefix=aa_
```

This solves the model and saves the work file `prefixed.g00`. The command line parameter symPrefix has the effect that all symbols in the work file are prefixed with `aa_`. For the next step, consider the following simple reporting code, saved in a file called `report.gms`:

```
Scalar i / 0 /;
loop(aa_i, i = i+1);
display 'number of canning plants', i;
```

Note that this code will be run with a restart from the work file `prefixed.g00`. Therefore, the looping set `aa_i` is identical to the set `i` in the model [**TRANSPORT**]. Since all symbols from the original model have been prefixed, convenient symbol names can be used for reporting purposes. The run of `report.gms` is achieved by the following call:

```
> gams report r=prefixed
```

The resulting listing file contains the following output:

```
----     76 number of canning plants
           PARAMETER i                   =        2.000
```

4.44.2 Use Cases

The basic function of a work file is to preserve information that has been expensive to produce. The following discusses several use cases for work files.

4.44.2.1 Separation of Model and Data

The separation of model and data is one of the core principles of the GAMS modeling paradigm. Using the save and restart feature helps to exploit this separation.

Separation of model and data will be illustrated on the transportation model [**TRNSPORT**]. First, consider a GAMS file `transportmodel.gms` which contains only the algebraic representation of the transportation problem, obtained by removing all data and execution statements from [**TRNSPORT**]:

```
Sets i   canning plants
     j   markets ;

Parameters  a(i)    "capacity of plant i in cases"
            b(j)    "demand at market j in cases"
            c(i,j)  "transport cost in 1000$/case"
            d(i,j)  "distance in 1000 miles"  ;

Scalar f  "freight in $/case per 1000 miles"  ;

Variables   x(i,j)  "shipment quantities in cases"
            z       "total transportation costs in 1000$"  ;
Positive Variable x ;

Equations  cost        "define objective function"
           supply(i)   "observe supply limit at plant i"
           demand(j)   "satisfy demand at market j" ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;
```

Second, consider a GAMS file `transportdata.gms` that contains the data of the model as well as the solve and display statements:

```
Sets   i   / seattle, san-diego /
       j   / new-york, chicago, topeka / ;

Parameters  a(i)  / seattle     350
                    san-diego   600  /
            b(j)  / new-york    325
                    chicago     300
                    topeka      275  / ;

Table d(i,j)
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4  ;

Scalar f / 90 / ;

c(i,j) = f * d(i,j) / 1000 ;

Solve transport using lp minimizing z ;
Display x.l, x.m ;
```

The second file (`transportdata.gms`) cannot be run alone as the definition of model `transport` is missing. However, one may first run the first file (`transportmodel.gms`) and save its work file. Then one can restart from this work file to run the second file:

```
> gams transportmodel.gms s=transmod
> gams transportdata.gms  r=transmod
```

4.44.2.2 Advanced Separation of Model and Data

In the previous example some execution time statements namely the assignment of `c`, the `solve`, and the `display` where performed in the data file. If the model execution logic is more complex we do not want to add this to the data file. Hence we create in this example a restart file from this first file but will only compile, but not execute (see difference between compile and execute phases in section GAMS Compile Time and Execution Time Phase):

```
$onEmpty
Sets i(*)   canning plants / /
     j(*)   markets        / / ;

Parameters  a(i)    "capacity of plant i in cases" / /
            b(j)    "demand at market j in cases"  / /
            c(i,j)  "transport cost in 1000$/case"
            d(i,j)  "distance in 1000 miles"       / /;

Scalar f  "freight in $/case per 1000 miles" / 0 /;

c(i,j) = f * d(i,j) / 1000 ;

Variables   x(i,j)  "shipment quantities in cases"
            z       "total transportation costs in 1000$"  ;
```

```
Positive Variable x ;

Equations  cost        "define objective function"
           supply(i)   "observe supply limit at plant i"
           demand(j)   "satisfy demand at market j" ;

cost ..        z  =e=  sum((i,j), c(i,j)*x(i,j)) ;
supply(i) ..   sum(j, x(i,j))  =l=  a(i) ;
demand(j) ..   sum(i, x(i,j))  =g=  b(j) ;

Model transport /all/ ;

Solve transport using lp minimizing z ;
Display x.l, x.m ;
```

Second, we consider a file that contains the data of the model but no other execution time statements. Because we already have empty data statements (`/ /`) for the data items in the first file, we need to instruct the compiler to allow a second data statement with the real data using $onMulti:

```
$onMulti
Sets   i   / seattle, san-diego /
       j   / new-york, chicago, topeka / ;

Parameters  a(i)  / seattle     350
                    san-diego   600  /
            b(j)  / new-york    325
                    chicago     300
                    topeka      275  / ;

Table d(i,j)
                  new-york       chicago      topeka
    seattle          2.5           1.7          1.8
    san-diego        2.5           1.8          1.4  ;

Scalar f / 90 / ;
$offMulti
```

In order to run the model we first compile, but not execute the first model (see command line parameter action=c) and create a save file. Next, we run the second model. This does a continued compilation, it compiles the data statements from the second files and then executes the execution time statements from the first (`c(i,j)=...`, `solve ...`, and `display ...`) and second (here there are none).

```
> gams transportmodel.gms action=c s=transmod
> gams transportdata.gms  r=transmod
```

4.44.2.3 Generating Concise Listing Files

By default a GAMS listing file has many components, see chapter GAMS Output for details. In case a more concise listing file is needed, the save and restart feature may be used to generate such a file.

Recall the reorganization of the transportation model [**TRNSPORT**] into a model file `trnsportmodel.gms` and a data-and-solve file `trnsportdata.gms` from the previous section. Now consider the addition of two further files. The first one, called `trnsportreport.gms`, contains post-solution computations for reporting:

```
Parameter m(*,*) 'movement of commodities in cases';
m(i,j)             = x.l(i,j);
m('total',j)       = sum(i, x.l(i,j));
m(i,'total')       = sum(j, x.l(i,j));
m('total','total') = sum(j, m('total',j));
```

The second one, called `trnsportdisplay.gms`, contains only a display statement:

```
Option decimals = 0;
Display m;
```

Using these four files, the following save and restart sequence can be called:

```
> gams trnsportmodel            s=trans1
> gams trnsportdata    r=trans1 s=trans2
> gams trnsportreport  r=trans2 s=trans3
> gams trnsportdisplay r=trans3
```

The output file `trnsportdisplay.lst` is brief. Apart from the echo print, the execution time and the file summary, it contains only the output generated by the display statement:

```
----     52 PARAMETER m  movement of commodities in cases

             new-york     chicago      topeka       total

seattle            50         300                     350
san-diego         275                     275         550
total             325         300         275         900
```

In this way it is possible to create output files that are concise and contain only the information needed, while at the same time the more detailed output is stored in other listing files and may be inspected if needed.

4.44.2.4 Incremental Program Development

GAMS programs are often developed in stages. A typical approach is to start with set statements, tables and data manipulations, then equations are declared and defined, followed by model and solve statements and finally assignments for generating reports. As each piece of the model is built, it should be run and checked for errors by inserting diagnostic display and abort statements. As confidence grows that the parts of the model are correct, it is useful to save the completed parts in a work file. Subsequently, it is possible to work only on the piece under active development by restarting from the saved work file and thus reducing running time and the amount of output produced in each of the development runs.

This approach is especially useful when working on the results report part, since the setup and solution of a model instance is typically dominating the computing time, while the report part has to be run often to get all details of setting up content and layout into a satisfying form. Thus, the model may be generated and solved and the result saved in a work file. One may then restart from the work file while developing the report.

4.44.2.5 Tracking a Sequence of Difficult Solve Statements

In many cases where solves are known to be difficult and expensive, it may be too risky to let GAMS process a job containing many solve statements. The risk is that if one solve does not proceed to normal completion, then the following solve will not be possible or will start from a bad initial point and much time and effort will be wasted.

An alternative is to request one solve at a time and save the work file. By doing so, the output of each solve can be carefully inspected before proceeding. If everything is as expected, the job can be restarted and the next solve be executed. If an error has occurred, the previous solve may be repeated, maybe with a different initial point or modified working limits such as iteration or time limits.

4.44.2.6 What-If Analysis

Many modeling exercises involve a '*what if* ' analysis, in which a *base case* is defined and the point of the study is to see how the system changes when circumstances change, either naturally or by design. Often, the effect of many different changes to the base case are considered separately.

The save and restart feature facilitates such analyses. The base case may be saved using a work file and then all desired scenarios may be run separately by restarting from the same work file. Each scenario probably involves only doing some changes to the data (e.g., coefficients in equations or variable bounds), solving the changed model (the solution of the base case will then automatically be used as a starting point), and reporting.

4.44.3 Secure Work Files

When models are distributed to users other than the original developers or embedded in applications to be deployed by other developers, issues of privacy, security, data integrity and ownership arise. One may have to hide, protect or purge some parts of the model before it can be released. The information to be protected can be of numeric or symbolic nature. Examples include the following:

Privacy

A Social Accounting Matrix supplied by a statistical office is required in a general equilibrium model to be used by the Ministry of Finance. The data from the statistical office needs to be protected for obvious privacy reasons and the model experiments are used to evaluate policy options that are highly confidential. Most of the model structure is public, most of the data however is private and model results need to be transformed in such a way as to prohibit the discovery of the original data.

Security

Components of a model contain proprietary information that describes mathematically a chemical reaction. The associated algebra and some of the data are considered of strategic importance and need to be hidden completely. However, the final model will be used at different locations around the world.

Integrity

Data integrity safeguards are needed to assure the proper functioning of a model. Certain data and symbolic information need to be protected from accidental changes that would compromise the operation of the model.

To address these issues, so called *secure* work files have been introduced to GAMS. Such a work file behaves like any other work file but it is locked to a specific user's license file. The content of a secure work file protected against unauthorized access via the GAMS license mechanism.

Attention

A special GAMS license is required to create secure work file, see also Usage.

The security features of secure work files are not extended to the solver level. As a consequence, full information about a model instance can be extracted on the GAMS solver level by a user that is authorized to solve the model. See Limitations for more information.

4.44.3.1 An Introductory Example

The well-known transportation model [**TRNSPORT**] will be used again to illustrate the creation and deployment of a secure work file. Assume one wants to distribute this model but there are concerns about proprietary formulations and data. In addition, one would like to prevent that the user does unintentional modifications to the model. It is assumed that the objective function and the supply constraints are to be hidden from other users and only the demand figures should be allowed to be changed. Data that is not needed any more will be purged as well.

First, a copy of the model from the model library is created, the model is run and a normal work file `t1` is created:

```
> gamslib trnsport
> gams trnsport s=t1
```

Next, a file `t2.gms` with access control commands is created,

```
$eolcom //
$protect all             // make all symbols read only
$purge   d f             // remove items d and f
$hide    cost supply a   // make objective invisible
$expose  transport b     // allow changes to b
```

and a *secure* work file `t2.g00` is created by executing `t2.gms` with a restart from `t1.g00`:

```
> gams t2 r=t1 s=t2 plicense=target
```

The newly created work file is secure since the access control commands were activated with the privacy GAMS license option PLicense. This command line parameter specifies the name of the target user's license file. The effect is that the work file `t2.g00` can now only be read with the target license file.

The log output will contain the following lines:

```
GAMS Rev 124  Copyright (C) 1987-2001 GAMS Development...
Licensee: Source User Name
          Source Company Name
*** Creating a Secure Restart File for:
***       Target User Name
***       Target Company Name
--- Starting continued compilation
--- T2.GMS(6) 1 Mb
--- Starting execution
*** Status: Normal completion
```

The three lines starting with *** are a recap of the content of the target license file. From now on, the source and the target licenses are 'burned into' this file and all its descendants. One can now send the restart file to the target user or system.

The target user may run the model with new data, add new GAMS statements and create new work files. However, there are two restrictions: some of the symbols are hidden and this model can only be executed using the target license file.

For example, the target user may want to half the demand and compare the original solution with the new one. The following file, called `t3.gms`, will accomplish this:

```
Parameter rep   'summary report';
rep(i,j,'base') = x.l(i,j);
b(j) = b(j)*0.5;

solve transport minimizing z using lp;

rep(i,j,'half') = x.l(i,j);
display rep;
```

This new file may be executed on the target system, restarting from the work file `t2.g00`:

```
> gams t3 r=t2
```

The resulting log file will contain the following lines:

```
GAMS Rev 124 Copyright (C) 1987-2001 GAMS Development...
Licensee: Target User Name
               Target User Company
*** Restarting from a Secure Restart File created by:
***       Source User Name
***       Source Company Name
--- Starting continued compilation
--- T3.GMS(5) 1 Mb
...
```

Note that the originator/owner of the secure work file is mentioned by name. A similar message will be contained in the listing file:

```
...
EXECUTION TIME     =    0.000 SECONDS   1.1 Mb   WIN201-124

**** Secure Save/Restart File Source:
        Source User Name
        Source Company Name
**** Secure Save/Restart File Target:
        Target User Name
        Target User Company
...
```

A more detailed inspection of the listing file shows that the hidden variables and equations do not appear in the usual equation and variable listings and the solution print. The hidden items can only be accessed via a public (exposed) model and a solve statement. However, note that the full model instance may still be accessed by the target user, see Limitations.

In the following, secure work files and the access control commands are described in more detail.

4.44.3.2 Usage

Secure work files control access to symbolic and numeric information and can only be read by a specific GAMS user. The initial creation or additions to access control requires a special GAMS license. Saving secure work files without new access controls does not require a special GAMS license. The creation or addition of access control is signaled by the use of the GAMS command line parameter PLicense, which gives the name of a privacy license file. The shortcut `PLICENSE=LICENSE` sets the privacy license to the current license file. This is convenient when experimenting with access controls.

When a secure work file is written for the first time, the first and second lines of the current license file and the privacy license file are inserted into the work file. This information cannot be changed any more and the original source and the intended target users are locked into the work file.

A secure work file may be used just like any other work file and new work files may be derived from secure files. However, their use is restricted to the `target` user specified with the command line parameter PLicense. The target user can, if licensed, add access controls to an existing secure file by using the parameter `PLICENSE=LICENSE` but cannot change the original information about source and target users.

There are four access control commands (ACCs) that are processed during the compilation phase. These commands can be inserted anywhere in the program. They are processed in chronological order and have the following syntax:

```
$acc ident1 ident2 ...
$acc all
```

Here `ident1` and `ident2` are GAMS identifiers previously defined in the program and `acc` denotes one of the four access control commands:

Dollar Control Option	Description
purge	Removes the objects and all data associated.
hide	Hides the objects but allows them to be used in model calculations.
protect	The objects cannot be modified but they may be used in model calculations.
expose	Removes all privacy restrictions, the symbols will be reverted to their original state.

The keyword `all` applies the ACCs to all identifiers defined up to this point in the GAMS source code. Note that ACCs may be changed and redefined within the same GAMS program. However, identifiers inherited from a restart file cannot be changed.

4.44.3.3 A Practical Example

This section uses the transportation model **[TRNSPORT]** to show how to hide input data and results from the target user. The target user will be allowed to view percentage changes from an unknown base case only. In addition to the original model, a data initialization and a report model will be introduced.

First, a method to calculate input data is defined. As the GAMS language does not offer the definition of methods (or functions), here a model is used to define algebraically the desired correspondence between the input and output of the method we wish to emulate. Execution of the method will then correspond to solving the model. The model is the following:

```
$include trnsport.gms

Variable newc(i,j)    'new tansport data';
Equation defnewc(i,j) 'definition of new transport data';
defnewc(i,j).. newc(i,j) =e= f*d(i,j)/1000;

Model getc            'compute new transport data' / defnewc /;
Solve getc using cns;
```

By solving model `getc` (see Constrained Nonlinear System (CNS) for details on problem type CNS), the variable `newc(i,j)` will obtain the value of `f*d(i,j)/1000` in the variable level attributes. Thus, `newc.l` corresponds to parameter `c` in the original model.

Next, the objective function of the original model is changed to reflect `economies of scale`. Furthermore, a base case value `basex` is computed for later use in the reporting model:

```
Scalar   beta        'scaling exponent' / 1.1 /;
Equation newcost     'economies of scale objective function';
newcost.. z =e= sum((i,j), newc.l(i,j) * x(i,j)**beta);

Model estrans / newcost, supply, demand /;
solve estrans using nlp minimizing z;

Parameter basex(i,j)  'base values of x';
basex(i,j) = x.l(i,j);
```

Finally, a method to transform the results of model `estrans` to the relative change with respect to the base case is defined. As for the computation of the input data (`newc`), a model is used to emulate this method:

```
Variable delta(i,j)      'percentage change from base values';
Equation defdelta(i,j)   'definition of delta';
defdelta(i,j)$basex(i,j).. delta(i,j) =e= 100*(x.l(i,j)-basex(i,j))/basex(i,j);

Model rep / defdelta /;
solve rep using cns;
```

Assume the GAMS code above has been saved in a file `p1.gms`. Running GAMS on this file,

```
> gams p1 s=p1
```

creates a work file with the name `p1.g00`.

In the following, some test runs similar to those that are expected to be defined by the target user are made. Three scenarios to be solved in a loop are defined in file `u1.gms`:

```
Set       s        / one, two, three /;
Parameter sbeta(s) / one 1.25, two 1.5, three 2.0 /
          sf(s)    / one 85,   two 75,  three 50 /;
Parameter report   'summary report';

loop(s,
   beta = sbeta(s);
```

```
   f    = sf(s);
   solve getc using cns;
   solve estrans using nlp minmizing z;
   solve rep using cns;
   report(i,j,s) = delta.l(i,j);
   report('','beta',s) = beta;
   report('','f',s)    = f;
   report('obj','z',s) = z.l
);

display report;
```

File `u1.gms` can be executed with a restart from the work file `p1.g00`:

```
> gams u1 r=p1
```

The display statement generates the following output:

```
----    109 PARAMETER report summary report

                            one         two       three

seattle  .new-york       -4.050      -6.967      -8.083
seattle  .chicago       -18.797     -27.202     -31.550
seattle  .topeka        233.958     348.468     404.187
san-diego.new-york        3.605       6.201       7.194
san-diego.chicago        28.138      40.719      47.228
san-diego.topeka        -15.512     -23.104     -26.799
         .beta            1.250       1.500       2.000
         .f              85.000      75.000      50.000
obj      .z             526.912    1652.963   13988.774
```

Note that all symbols are still completely exposed. Access controls need to be added to the model `p1.gms` before it can be made available to the target client. The information to be protected is the original distance matrix and derived information. A recommended procedure is to first hide everything and then enable access to only selected parts of the model. The access control information is collected in the file `s1.gms`:

```
$hide all
$expose getc estrans rep
$expose i j z delta
$expose f beta a b
```

Using the initial workfile p1, a secure work file `s1.g00` is created by executing

```
> gams s1 r=p1 s=s1 plicense=license
```

To test the system from the target users point of view, the license of the current GAMS system is used also for the target user.

To test the secure work file, problem `u1.gms` is run again, restarting from the work file `s1`:

```
> gams u1 r=s1
```

Inspecting the resulting listing file, one observes that equation, variable and solution listings related to the hidden variables are not shown anymore. Any attempt to reference a hidden variable will cause a compilation error.

4.44.3.4 Limitations

One of the design goals for secure work files has been to minimize the impact on other components of the GAMS system. Solvers used within a secure environment should work as if called within a normal environment. This implies that certain information about a model can easily be recovered by using solvers like CONVERT or solver options like writelp for CPLEX, writemps for CBC, or gams/interactive for SCIP.

For example, consider the secure work file `t2.g00` from Section An Introductory Example. The target user could now reproduce the complete model instance, including the hidden equation `supply` or the objective function `cost` by running the following GAMS code, restarting from `t2.g00`:

```
option mip = scip;
transport.optfile = 1;
$echo gams/interactive = "display prob quit" > scip.opt
solve transport min z using mip;
```

This produces the log output:

```
...
original problem has 6 variables (0 bin, 0 int, 6 cont) and 5 constraints

STATISTICS
  Problem name     : r
  Variables        : 6 (0 binary, 0 integer, 0 implicit integer, 6 continuous)
  Constraints      : 0 initial, 5 maximal
OBJECTIVE
  Sense            : minimize
VARIABLES
  [continuous] <x(seattle,new-york)>: obj=0.225, original bounds=[0,+inf]
  [continuous] <x(seattle,chicago)>: obj=0.153, original bounds=[0,+inf]
  [continuous] <x(seattle,topeka)>: obj=0.162, original bounds=[0,+inf]
  [continuous] <x(san-diego,new-york)>: obj=0.225, original bounds=[0,+inf]
  [continuous] <x(san-diego,chicago)>: obj=0.162, original bounds=[0,+inf]
  [continuous] <x(san-diego,topeka)>: obj=0.126, original bounds=[0,+inf]
CONSTRAINTS
  [linear] <supply(seattle)>: <x(seattle,new-york)>[C] +<x(seattle,chicago)>[C] +<x(seattle,topeka)>
  [linear] <supply(san-diego)>: <x(san-diego,new-york)>[C] +<x(san-diego,chicago)>[C] +<x(san-diego,
  [linear] <demand(new-york)>: <x(seattle,new-york)>[C] +<x(san-diego,new-york)>[C] >= 325;
  [linear] <demand(chicago)>: <x(seattle,chicago)>[C] +<x(san-diego,chicago)>[C] >= 300;
  [linear] <demand(topeka)>: <x(seattle,topeka)>[C] +<x(san-diego,topeka)>[C] >= 275;
END
...
```

Note, that this approach is only possible for the target user, since its license is required to unlock the work file. Further, the original algebra of the model is still protected by GAMS, only the generated model instance (with original variable and equation names) and variable and equation symbols can be made available this way.

A further limitation of secure work files is that the source and target license files are locked to the secure work file and cannot be changed. If the target user receives a new license file (due to a system upgrade, for example), the secure work file cannot be read any more by the target user and must be recreated.

4.44.4 Obfuscated Work Files

GAMS offers a limited set of facilities to change all the names and other documentation related to a specific model run in a work file. This can be useful if the model has to be solved in an untrusted computing environment, e.g., a public cloud facility. When the results are back in the secure environment, the changed names can be transformed back to their original.

A work file that has all strings for symbols, set elements and explanatory text replaced with obfuscated names is called an *obfuscated* work file. GAMS obfuscates by keeping the original length of the symbol name or label name, but replaces them by a sequence of strings. For example, for symbols of length 3 it creates `A00`, `A01`, `A02`, ..., `Z__`. The method used for explanatory text is similar, but here GAMS always uses the single quote character and thus may create many weird looking labels. In addition to item names, labels and explanatory text, titles and subtitles in the listing file are obfuscated. Observe that other strings that have some meaning in the execution of GAMS, e.g., file names, cannot be changed.

Despite of the changes in the string pool, a obfuscated work file has all the capabilities of a normal work file.

Recall from Section Saving The Work File that work files are created with the command line parameter save. Obfuscated work files are created with the variant saveobfuscate. The creation of an obfuscated work file is often combined with the creation of a regular work file, as the latter will be necessary to translate information from an obfuscated work file back to the name space of the original model.

In the following, the intended use of obfuscated work files is illustrated. First, a GAMS model (here [**TRNSPORT**]) is compiled into a regular and an obfuscated work file:

```
> gamslib trnsport
> gams trnsport action=compile s=0named so=0obfuscated
```

Note, that the command line parameter `action=compile` is used to only compile, but not execute, the model. Observe that the name of the regular work file is `0named.g00` and the name of obfuscated work file is `0obfuscated.g00`.

In the next step, the obfuscated work file is moved to a non-secure machine and GAMS is restarted from this work file on an empty GAMS model file. This executes the original model (here `trnsport`), but with all names obfuscated. The outcome of this execution is saved in another, still obfuscated, work file `1obfuscated.g00`.

```
> echo *Empty > empty.gms
> gams empty r=0obfuscated s=1obfuscated
```

The work file `1obfuscated.g00` is now brought back to a trusted machine. To obtain the results, say the marginal values of the `supply` equation, in a non-obfuscated form, GAMS is executed again on a model that writes out values of `supply.m` to a GDX file `supply.gdx`. This model is restarted from the obfuscated work file `1obfuscated.g00`, but additionally the originally created non-obfuscated work file `0named.g00` is passed in via the command line parameter RestartNamed (short: `rn`):

```
> echo "execute_unload 'supply', supply.m;" > unload.gms
> gams unload r=1obfuscated rn=0named
```

Recall, that the work file `0named.g00` resulted from the initial compilation and contains the names from the original namespace. It never left the trusted environment. The effect of combining options `r` and `rn` is that *all* content from the obfuscated work file is taken, except for the names of symbols and labels, the explanatory texts, and the listing file titles and subtitles, which are read from the work file specified via `rn`.

GAMS automatically performs the following three checks to *ensure* that the named and obfuscated work files are consistent:

1. The number of labels and symbols must be identical.
2. The size of the string pool must be identical.
3. The first 10 labels point to the same addresses in the string pool.

The first two checks imply that the execution in the obfuscated name space cannot introduce new symbols or labels or even new strings (e.g., from `display 'this is a new string'`). Therefore, the empty GAMS program that was used to execute from the obfuscated work file above can hardly be replaced by any useful code since the obfuscated symbols and labels are not known.

4.45 Embedded Code Facility

4.45.1 Motivation

GAMS uses relational data tables as a basic data structure. With these, GAMS code for parallel assignment and equation definition is compact, elegant, and efficient. However, traditional data structures (arrays, lists, dictionaries, trees, graphs, ...) are not natively or easily available in GAMS. Though it is possible to represent such data structures in GAMS, the GAMS code working with such structures can easily become unwieldy, obfuscating, or inefficient. Also, in the past it was not easy to connect libraries from other systems for special algorithms (e.g. graph algorithms, matrix operations, ...) to GAMS without some data programming knowledge and a deep understanding of internal representation of GAMS data (e.g. the `GDX API`).

The Embedded Code Facility addresses this need and extends the connectivity of GAMS to other programming languages. It allows the use of external code (e.g. Python) during compile and execution time. GAMS symbols are shared with the external code, so no communication via disk is necessary. It utilizes an API (this API can be found in `apifiles/C/api/emblib*.h` together with the code for the Python and Connect embedded code library `apifiles/C/api/embpyoo.c/h`) and additional source code for common tasks so that the user can concentrate on the task at hand and not the mechanics of moving data in and out of GAMS.

4.45.2 Concept

As pointed out in section Motivation, the main idea of the Embedded Code Facility is to enable the use of external code in GAMS and give this code direct in-memory access to the GAMS database (or better: to GAMS symbols, namely sets, parameters, variables, and equations). This can be done by defining sections in the GAMS code which contain code written in a defined external code. These sections can be used at both GAMS compile and execution time (compare section GAMS Compile Time and Execution Time Phase). Details about how to do this can be found in the Syntax section.

Note

> We will continuously extend this feature to different programming languages. At the moment only Python, Connect, and GAMS are supported.

Also, the system provides some help to develop and debug the external code independent of GAMS first. More about this topic can be found in section Troubleshooting Embedded Python Code.

The communication of the data between the GAMS part of the program and the embedded code part was inspired by the existing interface to GDX in many ways. For example, using dollar control options $on/offMulti, $onMultiR, and $on/offFiltered it is possible to decide if data changed in the embedded code should replace the GAMS data or get merged with it and just like loading data from GDX, one can decide if data from embedded code should change the GAMS database filtered or domain checked.

4.45.3 Simple Example

4.45.3.1 Python

In a very first example we look into some Python code that helps to split some label names that are already present in GAMS. We do this at compile time in order to read the broken up pieces as individual sets (`country` and `city`) into GAMS plus some mapping sets (`mccCountry` and `mccCity`) between the original labels and the new labels. The compile-time embedded code starts with a `$onEmbeddedCode` followed by the type of code to expect (`Python:`). The lines between `$onEmbeddedCode` and `$offEmbeddedCode` is Python code. We do not want to go into Python details, but the first few lines initialize some empty Python `list` objects (`mccCountry` and `mccCity`) as well as some empty Python `set` objects. In the Python `for` loop that follows we iterate over all individual labels of the GAMS set `cc`. Python gets access to the GAMS set `cc` via the member function `get` of the implicitly defined Python object `gams`. `get` returns an object that is *iterable* and can be used in a Python `for` loop. The type of the records one gets depend on the dimensionality and type of the GAMS symbol. In the loop body we use the Python `split` function to extract the first (`r[0]` is country) and second (`r[1]` is city) part of the label. The three strings `cc`, `r[0]`, `r[1]` are used to build up the Python `list` objects that store the information for the maps `mccCountry` and `mccCity` and the Python `set` objects that store the labels for the new sets `country` and `city`. The Python `set` has the advantage to store a label just once even if we add it multiple times. Python prepares to send items back to GAMS via the `gams` member function `set` that can deal with both Python `list` and Python `set` objects. The command `$offEmbeddedCode` is followed by a list (without separating commas) of GAMS symbols that instructs the GAMS compiler to read these symbols back. This list is optional and can be left out. GAMS will import all symbols set by the Python code. The GAMS compiler (without executing execution-time embedded code) does not know which symbols will be implicitly loaded, hence it cannot *define* symbols. If such a symbol is referenced further down in the compilation process, the compiler will complain with error `$141: Symbol declared but no values have been assigned.` One can avoid this situation by explicitly list the symbols on the `end/pauseEmbeddedCode` line.

Another caveat with implicit loading is related to aliases. GAMS does not allow to load data into an alias, so the following code will fail with a compilation error `493 Alias cannot receive data`:

```
Set i(*); Alias (i,j);
$onEmbeddedCode Python:
gams.set('j',['i1','i2','i3'])
$offEmbeddedCode j
```

By leaving off the symbol `j` from the `$offEmbeddedCode` line, the code works fine and actually imports the set `i`. The `gams.set` and other write methods in embedded code to an alias symbol, here `j`, actually fills the aliased set, here `i`, and hence the implicit load will load the modified symbol `i` into GAMS.

Implicit loading also needs special consideration in the context of external input. Since embedded code inside an `$on/offExternalInput` section is not executed if the command line parameter IDCGDXInput has been set, there will be no symbols to be loaded implicitly. So make sure the symbols to load are given explicitly in such a situation.

Note that GAMS syntax is case insensitive while Python is case sensitive. Hence, the strings that represent the GAMS symbol names in the Python code can have any casing (e.g. `gams.get("cc")` or `gams.get("CC")`), while the corresponding Python objects need to have consistent casing throughout the Python code.

The following code presents the entire **embeddedSplit** example from the GAMS Data Utilities Library:

```
Set cc      / "France - Paris", "France - Lille", "France - Toulouse"
              "Spain - Madrid", "Spain - Cordoba", "Spain - Seville", "Spain - Bilbao"
              "USA - Washington DC", "USA - Houston", "USA - New York",
              "Germany - Berlin", "Germany - Munich", "Germany - Bonn" /
    country
    city
    mccCountry(cc,country<)  Mapping between country and related elements in set cc
    mccCity(cc,city<)        Mapping between city and related elements in set cc;

$onEmbeddedCode Python:
mccCountry = []
mccCity = []
for cc in gams.get("cc"):
  r = str.split(cc, " - ", 1)
  mccCountry.append((cc,r[0]))
  mccCity.append((cc,r[1]))
gams.set("mccCountry",mccCountry)
gams.set("mccCity",mccCity)
$offEmbeddedCode mccCountry mccCity

Option mccCountry:0:0:1, mccCity:0:0:1;
Display country, city, mccCountry, mccCity;
```

The display in the listing file looks as follows:

```
----     25 SET country
Spain  ,    USA    ,    Germany,    France

----     25 SET city
Berlin       ,    Bilbao       ,    Cordoba      ,    Madrid
New York     ,    Washington DC,    Paris        ,    Houston
Munich       ,    Lille        ,    Seville      ,    Bonn
Toulouse

----     25 SET mccCountry
France - Paris     .France
France - Lille     .France
France - Toulouse  .France
Spain - Madrid     .Spain
Spain - Cordoba    .Spain
Spain - Seville    .Spain
Spain - Bilbao     .Spain
USA - Washington DC.USA
USA - Houston      .USA
USA - New York     .USA
Germany - Berlin   .Germany
Germany - Munich   .Germany
Germany - Bonn     .Germany

----     25 SET mccCity
France - Paris     .Paris
France - Lille     .Lille
France - Toulouse  .Toulouse
Spain - Madrid     .Madrid
Spain - Cordoba    .Cordoba
Spain - Seville    .Seville
Spain - Bilbao     .Bilbao
```

```
USA - Washington DC.Washington DC
USA - Houston       .Houston
USA - New York      .New York
Germany - Berlin    .Berlin
Germany - Munich    .Munich
Germany - Bonn      .Bonn
```

The second example demonstrates the use of embedded code at execution time. The syntax for the execution time embedded code in this example is identical to the compile time variant with the exception of the keywords that start and end the embedded code section: `embeddedCode` and `endEmbeddedCode`. It is important to understand that the execution of the code happens at GAMS execution time, so e.g. no new labels can be produced and send back to GAMS. In this example we use some Python code to generate a random permutation of set elements of set `i` and store this in a two dimensional set `p`. In this example we do not use a loop to iterate through the elements of a GAMS system but make use of the fact that the Python object returned by `gams.get("i")` is iterable and can in its entirety be stored in the Python `list` with name `i` with the short and powerful command `i = list(gams.get("i"))`. The permutation of elements in list `p` which is a copy of list `i` is created by the Python statement `random.shuffle(p)`. The following code presents the entire example:

```
Set i /i1*i10/
    p(i,i) "permutation";

embeddedCode Python:
import random
i = list(gams.get("i"))
p = list(i)
random.shuffle(p)
for idx in range(len(i)):
    p[idx] = (i[idx], p[idx])
gams.set("p", p)
endEmbeddedCode p

option p:0:0:1;
display p;
```

The display in the listing file looks as follows:

```
----     11 SET p  permutation
i1 .i1
i2 .i7
i3 .i5
i4 .i2
i5 .i10
i6 .i6
i7 .i9
i8 .i4
i9 .i8
i10.i3
```

4.45.3.2 Connect

In the following example, we need to process a CSV file that cannot be read directly by GAMS under $onDelim. So we instruct Connect to read the CSV file into the Connect database and then write to GAMS:

```
Set dates, stocks;
Parameter stockprice(dates<,stocks<);

$onText
Read a CSV file that looks as follows:

date;symbol;price
2016/01/04;AAPL;105,35
2016/01/04;AXP;67,59
2016/01/04;BA;140,50
...

and can't be read directly by GAMS under $onDelim because some labels
include '/' and are not quoted, use ';' as field separator, and ',' as
decimal character.
$offText

$onEmbeddedCode Connect:
- CSVReader:
    file: stockprice.csv
    name: stockprice
    indexColumns: [1, 2]
    valueColumns: [3]
    fieldSeparator: ';'
    decimalSeparator: ','
- GAMSWriter:
    symbols:
      - name: stockprice
$offEmbeddedCode

display stockprice;
```

We leave the CSV file processing to the CSVReader and then export the `stockprice` symbol to GAMS. Please note that the symbols written to GAMS by the GAMSWriter do not have to be explicitly listed on the `$offEmbeddedCode` line, this is optional. Moreover, the Connect agents GAMSReader and GAMSWriter are only available under embedded code.

We might use the data to calculate an optimal portfolio. The share of each stock in this optimal portfolio is represented by the parameter `share(stocks)` and can be exported to Excel using the ExcelWriter. Before exporting to Excel, the data first needs to be transferred from GAMS to Connect using the GAMSReader:

```
* Continues the code from the previous example

Parameter share(stocks);
* For demonstration we fill share with random numbers
share(stocks) = uniform(0,1);

embeddedCode Connect:
- GAMSReader:
    symbols:
      - name: share
- ExcelWriter:
    file: share.xlsx
    symbols:
      - name: share
endEmbeddedCode
```

4.45.3.3 GAMS

In the following example, we need some execution time GAMS power to create data needed at compilation. We want to solve the bin packing problem. In order to solve this we need the set of bins which is not part of the input data. There are different ways of dealing with this not so uncommon issue and they are all given in the GAMS Model Library model **binpacking**, but we want to concenrate here on a solution that utilizes embedded code GAMS:

```
Set i 'items';
Parameter s(i<) 'item sizes';
Scalar B 'bin capacity';
$gdxLoad data.gdx s B

* Randomize the item sizes:
s(i) = uniform(0.9,1.1)*s(i);

Scalar nj 'number of bins required to pack all items';
$save.keepCode bp

$onEmbeddedCode GAMS: restart=bp
scalar size /0/;
nj = 1; loop(i, size = size+s(i); if (size>B, nj = nj+1; size = s(i)));
$offEmbeddedCode nj

$eval NJ nj
Set j 'bins' / b1*b%NJ% /;

* Simple optimization model to minimize number of bins:
Binary variable y(i,j) 'assignment of item to bin', z(j) 'bin open';
Variable open_bins 'number of open bins';

Equations
    defopen(j) 'allow item in open bins only'
    defone(i)  'assign each item to one bin'
    defobj     'count number of open bins';

defopen(j)..   sum(i, s(i)*y(i,j)) =l= z(j)*B;
defone(i)..    sum(j, y(i,j)) =e= 1;
defobj..       open_bins =e= sum(j, z(j));

model bp 'bin packing' /all/;
solve bp min open_bins using mip;
```

In this example we read the item sizes and the bin capacity from a GDX file and need to create the set of bins `j`. We even randomize the item sizes (at GAMS execution time). In the embedded code GAMS section we utilize a simple way to estimate the number of bins: we pack the items (in data entry order) in a bin and as soon as the bin capacity is exceeded, we open the next bin. Obviously, this is far from being an optimal strategy, but it allows based on the actual sizes to quickly overestimate the number of bins required for the optimization model. The scalar of the number of bin (`nj`) is communicated back to the outer GAMS process and we use this to set a compile time variable `NJ` that is used to build the set of bins `j`. The embedded code GAMS that calculates `nj` can access the items and its sizes without declaration or data statements. This is possible because the embedded code *continues* via embedded code arguments `restart=bp` from a save file created one line above with $save.keepCode bp. Here the save file is created at compile time via `$save` and the execution code compiled so far is kept in the save file due to suffix `.keepCode`. So when the embedded code GAMS section runs, it first executes the code from the save file (here `s(i) = uniform(0.9,1.1)*s(i);`) before it executes its own code (`nj=1; loop(i, ...);`). Without the suffix `.keepCode` the embedded code GAMS would have used the item sizes available at compile time, i.e. the sizes as we find them in the GDX file. The remaining part of the model is a straight forward binary linear model for the bin packing problem.

4.45.4 Syntax

This section explains the GAMS functions/keywords which were introduced to enable the Embedded Code Facility. The first subsection deals with the syntax for compile time, the second with the syntax for execution time (compare section GAMS Compile Time and Execution Time Phase).

4.45.4.1 Compile Time

There are three dollar control options to start an embedded code section at compile time:

```
$onEmbeddedCode[.tag]  [Connect|Python]: [arguments]
$onEmbeddedCodeS[.tag] [Connect|Python]: [arguments]
$onEmbeddedCodeV[.tag] [Connect|Python]: [arguments]
```

Lines following one of the above statements are passed on to the embedded code engine (e.g. Connect or the Python interpreter) until this dollar control option, which ends the embedded code section at compile time, is hit:

```
$offEmbeddedCode[.tag] {symbol[<[=]embSymbol[.dimX]]}
```

These dollar control options are explained here in more detail. The usage of the optional `arguments` by the embedded code is explained in the specific embedded code engine section below.

Note

- The optional `arguments` from the `$onEmbeddedCode[S|V][.tag]` statement can be accessed as `gams.arguments` in the Python code.
- The optional `output symbols` from the `$offEmbeddedCode[.tag]` statement need to be written to by the embedded code.
- More about the specific GAMS Python syntax can be found below.

4.45.4.2 Execution Time

At execution time an embedded code section is started with one of these statements:

```
embeddedCode[.tag]  [Connect|Python]: [arguments]
embeddedCodeS[.tag] [Connect|Python]: [arguments]
embeddedCodeV[.tag] [Connect|Python]: [arguments]
```

Similar to the compile time alternatives $onEmbeddedCode[S|V][.tag], the first two variants are synonyms which allow parameter substitution in the code that follows, while the last variant does not allow this but passes the code verbatim to the embedded code engine. The usage of the optional `arguments` by the embedded code is explained in the specific embedded code engine section below. The arguments passed to the embedded code engine at execution time can be extended by the put_utility command `ECArguments`. For example,

```
put_utility 'ECArguments' / ' world...';
embeddedCode  Python: Hello
gams.printLog(gams.arguments)
endEmbeddedCode
```

prints `Hello world...`.

Lines following one of the statements `embeddedCode[S|V]` are passed on to the embedded code engine (e.g. the Python interpreter) until one of the following two statements, which end the embedded code section at execution time, is hit:

```
endEmbeddedCode[.tag]   {output symbols}
pauseEmbeddedCode[.tag] {output symbols}
```

When the optional `tag` suffix (which can be any string) is used to start an embedded code section, the same one has to be used to end/pause that section. Both statements end the embedded code section and switch back to GAMS syntax in the following lines. Also, both statements can be followed by a GAMS symbol or a list of GAMS symbols which would get updated in the GAMS database after the embedded code got executed. If `output symbols` are specified, they need to be written to (e.g. by `gams.set` for embedded Python code) by the embedded code before.

To continue a previously paused embedded code section one of the following statements is used:

```
continueEmbeddedCode[.tag]  [handle]: [arguments]
continueEmbeddedCodeS[.tag] [handle]: [arguments]
continueEmbeddedCodeV[.tag] [handle]: [arguments]
```

As seen before, the first two variants are synonyms which allow parameter substitution in the embedded code that follows, while the last variant does not allow this but passes the code verbatim to the interpreter. Again, the usage of the optional `arguments` by the embedded code is explained in the specific embedded code engine section below.

New in these statements is the optional `handle`. If omitted, the last code section that was paused will be continued. However, sometimes one might need to maintain different embedded code sections active in parallel and independent of each other. There is a new function to store a handle of the last embedded code section that was executed which could then later be used to continue a specific paused code section:

```
handle = embeddedHandle;
```

What it exactly means to end or pause/continue an embedded code section depends on the embedded code engine and is explained in more detail with the code engine below. Independent of the embedded code engine it is important to note that if GAMS stops and restarts while executing a `solve` statement (with solveLink=0) continuing any embedded code section causes the code to fail with the following error message:

```
    Error executing "continueEmbeddedCode" section
--- (Hint: "Solve" with SolveLink=0 frees previously initialized embedded libraries):
    Error at line 71: No embedded library initialized
```

This happens because under solvelink=0 the GAMS process terminates when the solver starts up. This causes the loss of the state of the embedded code environment that is linked into the GAMS process as a shared library. Under all other settings of solveLink the GAMS process is not stopped and restarted and hence a paused embedded code environment can be continued after the solve statement has carried out.

4.45.5 Python

GAMS and Python complement each other in many different ways. GAMS has a compact but readable syntax for data assignment and equation definition statements. Python is a great scripting language to do more traditional type programming especially string manipulation which is completely missing in GAMS. Moreover, the vast number of packages help solving scientific problems outside the scope of GAMS. In order to open this world to our customers in an easy way, GAMS comes with a Python 3 installation. By default, this installation is used in the Embedded Code Facility for Python and is ready to be used with the GAMS Python API. The Python installation is located in `[GAMS directory]/GMSPython`. In contrast to earlier versions this installation comes without the Python package manager `pip` and therefore is not easily extendable. `GMSPython` comes with a selection of third party packages and modules that are either required by the APIs of the GAMS Python API collection or that are particularly helpful from within embedded Python code. Those packages are located in the `site-packages` directory of `GMSPython`. Running the `GMSPython` interpreter like follows will display all available modules:

```
[GAMS directory]/GMSPython/python -c"help('modules')"
```

The list of redistributed third party packages is subject to constant changes and it is not recommended to rely on their availability. If you need packages that do not come with `GMSPython`, we recommend that you install your own version of Python and follow a few steps to connect GAMS with your Python installation. It is also possible to install `pip` for `GMSPython` using get-pip. While any Python installation should work in combination with GAMS we can recommend the `Anaconda` or `Miniconda` Python distributions. These Python distributions work with environments which allow you to have many different Python installations at the same time that don't get in each other's way.

During the lifetime of the GAMS process a single Python interpreter is used and the code *"lives"* in Python's global scope. This means that objects defined in one embedded Python code section are still there in another embedded Python code section:

```
$onEmbeddedCode Python:
s = 'Hello world'
$offEmbeddedCode
$onEmbeddedCode Python:
gams.printLog(s)
$offEmbeddedCode
```

will produce the following output in the GAMS log:

```
--- Starting compilation
--- main.gms(3) 2 Mb
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
--- main.gms(6) 2 Mb
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
Hello world
--- main.gms(7) 2 Mb
```

If the GAMS process stops and restarts during execution, e.g. because of a solve with solveLink=0, the Python interpreter will be started new. This can cause complications if GAMS is in a sequence of `pause/continueEmbeddedCode` as discussed above.

Note

> Sometimes it is desirable to move Python code to separate files which get imported from within an embedded Python code section. By default the current working directory is not considered by Pythons `import` statement. In order to achieve this, one can add `sys.path.append(".")` before the actual import.

4.45.5.1 Interface between GAMS and Python

The Python class `ECGamsDatabase` is the interface between GAMS and Python. An instance of this class is automatically created when an embedded code section is entered and can be accessed using the identifier `gams`. The following methods can be used in order to interact with GAMS:

`gams.get(symbolName, keyType=KeyType.STRING, keyFormat=KeyFormat.AUTO, valueFormat=ValueFormat.AUTO,`

This method retrieves an iterable object representing the symbol identified with `symbolName`. Typically there are two possibilities to access the records. Iterating using e.g. a `for` loop provides access to the individual records. By calling `list()` on the iterable object, a list containing all the data is created. Several optional parameters can be used in order to modify the format of the retrieved data:

- `keyType`: Determines the data type of the keys. It can be either `KeyType.STRING` (labels) or `KeyType.INT` (label indexes). The default setting is `KeyType.STRING`.
- `keyFormat`: Specifies the representation of the keys. Possible values are as follows:
  - `KeyFormat.TUPLE`: Encapsulate keys in a tuple
  - `KeyFormat.FLAT`: No encapsulation
  - `KeyFormat.SKIP`: Keys are skipped and do not appear in the retrieved data
  - `KeyFormat.AUTO` (default): Depending on the dimension of the GAMS symbol, a default format is applied:
    - Zero dimensional/scalar: `KeyFormat.SKIP`
      - One dimensional: `KeyFormat.FLAT`
      - Multi dimensional: `KeyFormat.TUPLE`
- `valueFormat`: Specifies the representation of the values. Possible values are as follows:
  - `ValueFormat.TUPLE`: Encapsulate values in a tuple
  - `ValueFormat.FLAT`: No encapsulation
  - `ValueFormat.SKIP`: Values are skipped and do not appear in the retrieved data
  - `ValueFormat.AUTO` (default): Depending on the type of the GAMS symbol, a default formats is applied:
    - Set: `ValueFormat.SKIP`
      - Parameter: `ValueFormat.FLAT`
      - Variable/Equation: `ValueFormat.TUPLE`
- `recordFormat` Specifies the encapsulation of records into tuples. Possible values are as follows:
  - `RecordFormat.TUPLE`: Encapsulates every record in a tuple
  - `RecordFormat.FLAT`: No encapsulation. Throws an exception if it can not be applied. It is guaranteed that the length of a retrieved Python list is equal to the number of records of the corresponding GAMS symbol. This principle leads to an incompatibility of `RecordFormat.FLAT` whenever a record consists of more than one item (e.g. multi dimensional symbols, variables and equations which have five numeric values).
  - `RecordFormat.AUTO` (default): Depending on the number of items that represent a record, a default format is applied. If possible this is always `RecordFormat.FLAT`.

GAMS special values `NA`, `INF`, and `-INF` will be mapped to IEEE special values `float('nan')`, `float('inf')`, and `float('-inf')`. GAMS special value `EPS` will be either mapped to `0` or to the small numeric value `4.94066E-324` depending on the setting of flag gams.epsAsZero.

The following Python code shows some examples of `gams.get` and illustrates the use of different formats:

```
Set i / i1 text 1, i2 text 2 /
    j / j1*j2 /

Scalar p0 /3.14/;
Parameter p1(i)   / #i 3.14 /
          p2(i,j) / i1.#j 3.14 /
Variable v0       / fx 3.14 /;
equation e1(i)    / #i.fx 3.14 /
         e2(i,j)  / i1.#j.fx 3.14 /;

$onEmbeddedCode Python:
# scalar parameter
gams.printLog(f"{list(gams.get('p0'))}")                                    # prints [3.14]
gams.printLog(f"{list(gams.get('p0', recordFormat=RecordFormat.TUPLE))}") # prints [(3.14,)]

# one dimensional parameters:
gams.printLog(f"{list(gams.get('p1'))}")                                    # prints [('i1', 3.14)
gams.printLog(f"{list(gams.get('p1', keyFormat=KeyFormat.TUPLE))}")       # prints [(('i1',), 3.
gams.printLog(f"{list(gams.get('p1', valueFormat=ValueFormat.TUPLE))}")   # prints [('i1', (3.14
gams.printLog(f"{list(gams.get('p1', keyFormat=KeyFormat.TUPLE, valueFormat=ValueFormat.TUPLE))}

# two dimensional parameter:
gams.printLog(f"{list(gams.get('p2'))}")                                    # prints [(('i1', 'j1'
gams.printLog(f"{list(gams.get('p2', keyFormat=KeyFormat.FLAT))}")        # prints [('i1', 'j1',

# one dimensional sets:
gams.printLog(f"{list(gams.get('i'))}")                                     # prints ['i1', 'i2']
gams.printLog(f"{list(gams.get('i',valueFormat=ValueFormat.FLAT))}")      # prints [('i1', 'text

# scalar variables/equations
gams.printLog(f"{list(gams.get('v0'))}")                                    # prints [(3.14, 0.0,

# one dimensional variables/equations:
gams.printLog(f"{list(gams.get('e1'))}")                                    # prints [('i1', (3.14
gams.printLog(f"{list(gams.get('e1', valueFormat=ValueFormat.FLAT))}")    # prints [('i1', 3.14,
gams.printLog(f"{list(gams.get('e1', keyFormat=KeyFormat.TUPLE))}")       # prints [(('i1',), (3

# two dimensional variables/equations:
gams.printLog(f"{list(gams.get('e2'))}")                                    # prints [(('i1', 'j1'
gams.printLog(f"{list(gams.get('e2', keyFormat=KeyFormat.FLAT, valueFormat=ValueFormat.FLAT))}")

# using label indexes instead of labels
gams.printLog(f"{list(gams.get('p1', keyType=KeyType.INT))}")             # prints [(1, 3.14), (
gams.printLog(f"{list(gams.get('i', keyFormat=KeyFormat.TUPLE, valueFormat=ValueFormat.TUPLE, ke
gams.printLog(f"{list(gams.get('e2', keyType=KeyType.INT))}")             # prints [((1, 3), (3.
$offEmbeddedCode

gams.set(symbolName, data, mergeType=MergeType.DEFAULT, domCheck=DomainCheckType.DEFAULT, mapKey
```

This method sets the data for the GAMS symbol identified with `symbolName`. The parameter `data` takes a Python list or set containing items that represent the records of the symbol. It is also possible to pass an instance of a subclass of **_GamsSymbol** (e.g. **GamsParameter** or **GamsSet**) when using the GAMS control API in an embedded code section. In case of a Python list or set, depending on the type and the dimension of the symbol, different formats can be used in order to specify the data. Different formats can not be mixed within one list. In general each record needs to be represented as a tuple containing the keys and the value field(s). Keys and/or values can also be enclosed in a tuple. Keys can be entered as labels (string) or label indexes (int). The argument `mapKeys` allows to pass a callable to remap the

elements of the key (e.g. turn them explicitly into strings via `mapKeys=str`). Value fields depend on the type of the symbol:

- Parameters: One numerical value
- Sets: explanatory text (optional)
- Variable/Equations: Five numerical values: level, marginal, lower bound, upper bound, scale/prior/stage

IEEE special values `float('nan')`, `float('inf')`, and `float('-inf')` will be remapped to GAMS special values `NA`, `INF`, and `-INF`. The small numeric value `4.94066E-324` will be mapped into GAMS special value `EPS`.

The following Python code gives some examples on different valid formats for different symbol types and dimensions:

```
Set i / i1 text 1, i2 text 2 /
    j / j1*j2 /
    ii(i);
Scalar p0;
Parameter p1(i)
          p2(i,j);
equation e1(i)
         e2(i,j);
$onEmbeddedCode Python:
# scalar parameter
gams.set('p0', [3.14])
gams.set('p0', [(3.14,)])

# one dimensional parameters:
gams.set('p1', [("i1", 3.14), ("i2", 3.14)])
gams.set('p1', [(("i1",), 3.14), (("i2",), 3.14)])
gams.set('p1', [("i1", (3.14,)), ("i2", (3.14,))])
gams.set('p1', [(("i1",), (3.14,)), (("i2",), (3.14,))])

# two dimensional parameter:
gams.set('p2', [('i1', 'j1', 3.14), ('i1', 'j2', 3.14)])
gams.set('p2', [(('i1', 'j1'), (3.14,)), (('i1', 'j2'), (3.14,))])

# one dimensional sets:
gams.set('ii', ['i1', 'i2'])
gams.set('ii', [('i1',), ('i2',)])

# one dimensional sets with explanatory text
gams.set('ii', [('i1', "text 1"), ('i2', "text 2")])
gams.set('ii', [(('i1',), ("text 1",)), (('i2',), ("text 2",))])

# one dimensional variables/equations:
gams.set('e1', [("i1", 3.14, 0, 0, 10, 1), ("i2", 3.14, 0, 0, 10, 1)])
gams.set('e1', [("i1", (3.14, 0, 0, 10, 1)), ("i2", (3.14, 0, 0, 10, 1))])
gams.set('e1', [(("i1",), (3.14, 0, 0, 10, 1)), (("i2",), (3.14, 0, 0, 10, 1))])

# two dimensional variables/equations:
gams.set('e2', [("i1", "j1", 3.14, 0, 0, 10, 1), ("i1", "j2", 3.14, 0, 0, 10, 1)])
gams.set('e2', [(("i1", "j1"), (3.14, 0, 0, 10, 1)), (("i1", "j2"), (3.14, 0, 0, 10, 1))])

# using label indexes instead of labels
gams.set('p1', [((1,), (3.14,)), ((2,), (3.14,))]) # one dimensional parameter
gams.set('ii', [((1,), ("text 1",)), ((2,), ("text 2",))]) # one dimensional set
gams.set('e2', [((1, 3), (3.14, 0, 0, 10, 1)), ((1, 4), (3.14, 0, 0, 10, 1))]) # two dimens
$offEmbeddedCode p0 p1 p2 ii e1 e2
```

The optional parameter `mergeType` specifies if data in a GAMS symbol is merged (`MergeType.MERGE`) or replaced (`MergeType.REPLACE`). If left at `MergeType.DEFAULT` it depends on the setting of $on/offMulti[R] if GAMS does a merge, replace, or trigger an error during compile time. During execution time `MergeType.DEFAULT` is the same as `MergeType.MERGE`. The optional parameter `domCheck` specifies if Domain Checking is applied (`DomainCheckType.CHECKED`) or if records that would cause a domain violation are filtered (`DomainCheckType.FILTERED`). If left at `DomainCheckType.DEFAULT` it depends on the setting of $on/offFiltered if GAMS does a filtered load or checks the domains during compile time. During execution time `DomainCheckType.DEFAULT` is the same as `DomainCheckType.FILTERED`.

Note

When calling `gams.set()` in an embedded code section during execution time, new labels that are not known to the current GAMS program can not be added. The attempt will result in an execution error.

`gams.getUel(idx)`

Returns the label corresponding to the label index `idx`

`gams.mergeUel(label)`

Adds `label` to the GAMS universe if it was unknown and returns the corresponding label index.

Note

When calling `gams.mergeUel()` in an embedded code section during execution time, new labels that are not known to the current GAMS program can not be added. The attempt will result in an execution error.

`gams.getUelCount()`

Returns the number of labels.

`gams.printLog(msg)`

Print `msg` to log.

`gams.arguments`

Contains the command line that was passed to the Python interpreter of the embedded code section. The syntax for passing arguments to the Python interpreter can be seen above.

`gams.epsAsZero`

Flag to read GAMS `EPS` as 0 (`True`) or as a small number, `4.94066E-324`, when set to `False`. Default is `True`.

`gams.ws`

Property to retrieve an instance of **GamsWorkspace** that allows to use the GAMS control API. The instance is created when the property is read for the first time using a temporary working directory. A different working directory can be specified using gams.wsWorkingDir. For debug output, the property gams.debug can be set to a value from **DebugLevel**

`gams.wsWorkingDir`

Property that can be specified before accessing gams.ws for the first time in order to set the working directory. Setting the property after the first call to gams.ws will have no effect. of the created GamsWorkspace.

`gams.db`

Property to retrieve an instance of **GamsDatabase**. The instance is created when the property is read for the first time and allows to access the GAMS symbols using the methods of the GAMS control API.

`gams.debug`

Property that can be set to a value from **DebugLevel** for debug output. Default is `DebugLevel.Off` (`0`). The property needs to be changed before accessing gams.ws for the first time in order to take effect in the GAMS control API. Setting the property after the first call to gams.ws will have no effect on the GamsWorkspace.

4.45.5.2 Using the Control API

The ECGamsDatabase class provides mechanisms for using the GAMS control API in an embedded code section. The property `gams.ws` can be used to get an instance of GamsWorkspace. The property `gams.db` allows to access an instance of GamsDatabase that can be used to read and write data from the internal GAMS database like it can be done using `gams.get` and `gams.set` but using the access mechanisms of the GamsDatabase class. In addition to the different reading and writing mechanisms offered by the GAMS control API other classes like *GamsModelInstance* can be used inside embedded Python code. This combination provides programmatic access to a generated model instance and exchange of update parameters and results between Python and GAMS. The GAMS Library model **embmiex1** gives an example of how to combine embedded Python code and a `GamsModelInstance` which utilized the library include `$libInclude pyEmbMI`.

Moreover, the GAMS Transfer Python API can be combine with the GAMS control API in the embedded code context as the following example demonstrates:

```
Set i / i1*i1000 /; alias(i,j);
Parameter a(i,j), aInv(i,j);
a(i,j) = 1 / (ord(i)+ord(j) - 1);
a(i,i) = 1;

embeddedCode Python:
import gams.transfer as gt
import numpy as np

m = gt.Container(gams.db)
A = m.data["a"].toDense()
m.data['aInv'].setRecords(np.linalg.inv(A))
m.write(gams.db, ["aInv"])
endEmbeddedCode aInv
```

4.45.5.3 Exchange via Files and Environment Variables

The Python class ECGamsDatabase provides read and write access to GAMS symbols. There are two other communication methods that can be used at GAMS compile time: *files* and *environment variables*. At compile time the Python code can produce a text file that can be included into GAMS via `$include` as in the following example:

```
$onEmbeddedCode Python: 10
with open('i.txt', 'w') as f:
  for i in range(int(gams.arguments)):
    f.write(f'i{i+1}\n')
$offEmbeddedCode

Set i /
$include i.txt
/;
display i;
```

Here the Python code received the number of elements to write to a text file via the argument after `Python:`. This text file is then included in the data statement of the GAMS set `i`. The display in the listing file looks as follows:

```
----     20 SET i
i1 ,    i2 ,    i3 ,    i4 ,    i5 ,    i6 ,    i7 ,    i8 ,    i9 ,    i10
```

Python provides many packages to read input files for many different formats and hence can be used to transform such formats to a GAMS compatible input format, as an alternative to providing the data via `list` objects and the gams.set functionality.

The second alternative to exchange information at compile time are environment variables. GAMS and Python allow to get and set environment variables and hence can be conveniently used to exchange small pieces of information. The following code provides an example where the maximum value of a parameter `b` is needed to build a set `k`:

```
Set       i    / i1*i5 /;
Parameter b(i) / i1 2, i2 7, i3 59, i4 2, i5 47 /;
$onEmbeddedCode Python:
import os
os.environ["MAXB"] = str(int(max([b for i,b in gams.get("b")])))
gams.printLog(f'max value in b is {os.environ["MAXB"]}')
$offEmbeddedCode

$if x%sysEnv.MAXB%==x $abort MAXB is not set
Set k "from 0 to max(b)" / 0*%sysEnv.MAXB% /;
$eval CARDK card(k)
$log card(k)=%CARDK%
```

It is important to understand that `os.environ` is initialized when Python imports the `os` module. Hence, if we later set an environment variable in GAMS (via `$setEnv ABC abc` or `put_utility "setEnv" / "ABC" / "abc"`) and access this in some embedded code via `os.environ["ABC"]`, this will not provide the expected value "abc". In such a case the environment variable needs to be accessed via `gams.get_env("ABC")` (and can be stored in `os.environ["ABC"]`) as demonstrated by the following code:

```
$onEmbeddedCode Python:
import os
os.environ["TEST"] = 'Hello world'
$offEmbeddedCode
$log TEST=%sysEnv.TEST%
$setEnv ABC abc
$onEmbeddedCode Python:
gams.printLog(f'ABC={os.environ.get("ABC", "ABC not set")}')
gams.printLog(f'ABC={gams.get_env("ABC")}')
$offEmbeddedCode
```

which produces the following output:

```
--- Starting compilation
--- main.gms(4) 2 Mb
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
TEST=Hello world
--- main.gms(10) 2 Mb
--- Initialize embedded library embpycclib64.dll
--- Execute embedded library embpycclib64.dll
ABC=ABC not set
ABC=abc
```

Obviously, the set `k` can also be filled directly with data from within Python:

```
Set       i    / i1*i5 /;
Parameter b(i) / i1 2, i2 7, i3 59, i4 2, i5 47 /;
Set       k(*) "from 0 to max(b)";

$onEmbeddedCode Python:
kmax = max([b for i,b in gams.get("b")])
gams.set('k',[str(k) for k in range(int(kmax+1))])
$offEmbeddedCode k
```

In both cases the resulting GAMS symbol `k` is a set with elements `0` to `59`.

Note that the `(*)` in the declaration of `k` is necessary to inform the compiler of dimensionality of the symbol `k`. GAMS symbol declarations can be done without domain list, so a `set k` does not necessarily mean that this is a one dimensional set. The compiler will *learn* the dimensionality from the use of `k`. For example, if `k` shows up as a domain set in a declaration (`parameter p(k)`) it is clear that `k` is a one dimensional set. Without knowing the dimensionality `k` will not be available with embedded code or exported (at compile-time) to GDX and the following code

```
Set k;
$onEmbeddedCode Python:
gams.set('k',['0','1','2'])
$offEmbeddedCode k
```

will fail with compilation error `Cannot load/unload symbol with unknown dimension`.

4.45.5.4 Encodings

Python 3 expects source code to be encoded in `UTF-8` per default. Therefore all embedded Python code has to be encoded in `UTF-8` as well. However, this can be customized by adding a comment in the format `# coding=<encoding name>` or `# -*- coding: <encoding name> -*-` as first line of an Embedded Python Code section. The encoding can be changed from any Embedded Python Code section and will affect all subsequent sections. So in case of a GAMS source file which is encoded using a non-`UTF-8` encoding like `cp1252` or others, it is sufficient to change the encoding only once in the very first Embedded Code section. Note that `UTF-16` encoding is not supported.

Note

Changing the source code encoding has no effect on the default encoding that is used by the `open()` function since it uses whatever `locale.getpreferredencoding(False)` returns. Furthermore the encoding used for `stdin`/`stdout`/`stderr` has to be controlled separately. This can be achieved by setting the environment variable `PYTHONIOENCODING`.

4.45.5.5 Multiple Independent Python Sessions

At execution time the user has the ability to *pause* and *continue* an embedded code segment. As explained at the beginning, the user Python code *lives* in Python's global scope.

4.45.5.6 Troubleshooting Embedded Python Code

The GAMS compiler ensures that the number of errors during execution time is minimized. While the logic of the GAMS program might be flawed there is nothing (with a few exceptions) that the GAMS system cannot execute. This is different if we embed foreign code in a GAMS program. The GAMS compiler does not understand the foreign code syntax and just skips over it. Only when the code is executed we will find out if everything works as expected. If the embedded code contains some (Python) syntax errors the Python parser will inform us about this and the message will appear in the GAMS log. For example, the following Python code using the gams.printLog function two times will generate a syntax error:

```
$onEmbeddedCode Python:
gams.printLog('hello')
 gams.printLog('world...')
$offEmbeddedCode
```

The GAMS log will provide some guidance:

```
--- Execute embedded library libembpycclib64.so
  File "<string>", line 7
    gams.printLog('world...')
    ^
IndentationError: unexpected indent
Python error! Return code from Python execution: -1
*** Error executing embedded code section:
*** Check log above
```

Moreover, if the Python code raises an exception which is not handled within the code this will also lead to a compilation or execution error in GAMS depending at what phase the embedded code is executed.

```
embeddedCode Python:
raise Exception('something is wrong')
endEmbeddedCode
```

will produce the following GAMS log and an execution time error:

```
--- Initialize embedded library libembpycclib64.so
--- Execute embedded library libembpycclib64.so
Exception from Python (line 0): something is wrong
*** Error at line 1: Error executing "embeddedCode" section: Check log above
```

Note

It is good practice to raise a Python exception if an error occurs. In any case using `exit()` needs to be avoided since it terminates the executable in an uncontrolled way.

The Python code is executed as part of the GAMS process and GAMS gives control to the Python interpreter when executing the embedded code. So in the worst case if the Python interpreter crashes, the entire GAMS process will crash. Therefore, it is important to be able to test and debug the embedded Python code independent of GAMS. In the following examples we call the Python interpreter executable as part of a GAMS job. In principle this can be tested and debugged completely independent of GAMS where a GDX file represents the content of the GAMS database.

In the first example we mimic the embedded code facility at compile time by exporting the entire GAMS database to a GDX file `debug.gdx`. With `$on/offEcho` we write the embedded code with a few extra lines at the top and bottom surrounded by a try/except block and execute the Python interpreter via `$call`. One of the extra lines at the end of the embedded code triggers the creation of a GDX result file `debugOut.gdx` which can be imported in subsequent `$gdxin/$load` commands.

```
Set i /i1*i10/
    p(i,i) permutation;

$gdxOut debug.gdx
$unload
$gdxOut

$onEcho > debug.py
from gams.core.embedded import *
gams = ECGAMSDatabase(r'%gams.sysdir% '[:-2], 'debug.gdx')
try:
   import random
   i = list(gams.get("i"))
   p = list(i)
   random.shuffle(p)
   for idx in range(len(i)):
       p[idx] = (i[idx], p[idx])
   gams.set("p", p)
   gmdWriteGDX(gams._gmd,'debugOut.gdx',1);
except Exception as e:
   print(str(e))
$offEcho

$call.checkErrorLevel ="%gams.sysdir%GMSPython/python" debug.py

$gdxIn debugOut.gdx
$loadDC p

Option p:0:0:1;
Display p;
```

In the second example we mimic the embedded code facility at execution time by exporting the entire GAMS database to a GDX file `debug.gdx` via `execute_unload 'debug.gdx';`. We write the embedded code with a few extra lines at the top and bottom surrounded by a try/except block via the `put` facility and execute the Python interpreter via `execute`. Identical to the compile time example, we export the result to the GDX file `debugOut.gdx` which can be imported via the `execute_load` statement.

```
Set i /i1*i10/
    p(i,i) permutation;

execute_unload 'debug.gdx';
file fpy / 'debug.py' /; put fpy;

$onPutS
from gams.core.embedded import *
gams = ECGAMSDatabase(r'%gams.sysdir% '[:-2],'debug.gdx')
try:
   import random
   i = list(gams.get("i"))
   p = list(i)
   random.shuffle(p)
   for idx in range(len(i)):
       p[idx] = (i[idx], p[idx])
   gams.set("p", p)
   gmdWriteGDX(gams._gmd,'debugOut.gdx',1);
except Exception as e:
   print(str(e))
$offPut

putClose fpy;
execute.checkErrorLevel '="%gams.sysdir%GMSPython/python" debug.py';

execute_load 'debugOut.gdx', p;
Option p:0:0:1;
Display p;
```

Automatic Indentation

All embedded Python code is automatically wrapped in a `try-except` block that requires an indentation (two spaces). While this additional indentation makes no difference for most Python code, it does make a difference when it comes to multi-line strings. Therefore, multi-line strings created with triple quotes (either single or double quotes) are excluded from this rule to avoid unintentional spaces, e.g. when writing to a file:

```
$onEmbeddedCode Python:
f = open("file.txt", "w")
f.write('''some
multi
line
string
''')
f.close()
$offEmbeddedCode
```

Still, when using the line continuation character (\) in combination with strings, one has to be aware of the extra spaces that are added automatically:

```
$onEmbeddedCode Python:
s = "some\
multi\
line\
string"
gams.printLog(s)
$offEmbeddedCode
```

This will print `some multi line string` instead of `somemultilinestring` due to the automatically applied indentation.

SSL Certificate Error

When doing HTTPS requests from within Embedded Python code using packages like `urllib` or `urllib3`, one might get an error indicating that no certificate was found:

```
urllib.error.URLError: <urlopen error [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: unable to get local
      issuer certificate (_ssl.c:1131)>
```

`GMSPython` comes with the `certifi` package that provides certificates to be used for validating the trustworthiness of SSL certificates. In order to make use of those, one can set the environment variable `SSL_CERT_FILE` at the beginning of the Embedded Code section, before making the actual request:

```
$onEmbeddedCode Python:
import certifi
import os
import urllib.request
os.environ['SSL_CERT_FILE'] = certifi.where()
urllib.request.urlretrieve('<url>', '<file>')
$offEmbeddedCode
```

4.45.5.7 Performance Considerations of Embedded Python Code

If the same embedded code section (e.g. in a loop) is executed many times there are a few considerations to be taken into account in order to get the best performance. For this we will experiment with the example from the introduction. We look for a random permutation of a set `i`. In addition we have a cost matrix `c(i,ii)` and we are looking for the least cost permutation. We should just formulate this as matching in a bi-partite graph but in order to demonstrate some performance considerations we will repeatedly call the Python code that provides a random permutation and we will evaluate the cost of the permutation in GAMS and store the value of the cheapest one. Here is the naive implementation using embedded code:

```
Set i / i1*i50 /
    p(i,i) permutation;
Alias (i,ii);
Parameter c(i,i) cost of permutation;
c(i,ii) = uniform(-50,50);

Set    iter / 1*100 /;
Scalar tcost
       minTCost / +inf /;
loop(iter,
  embeddedCode Python:
  import random
  i = list(gams.get("i"))
```

```
  p = list(i)
  random.shuffle(p)
  gams.set("p", [ (i[idx], p[idx]) for idx in range(len(i)) ])
  endEmbeddedCode p
  tcost = sum(p, c(p));
  if (tcost < minTCost, minTCost = tcost);
);
Display minTCost;
```

Even though the Python interpreter stays alive the fresh start and end of an embedded code section goes together with the setup and initialization of multiple data structure, including the `gams` object of type `ECGAMSDatabase`. We can avoid the repeated setup and initialization by using `pause` and `continue`:

```
Set i / i1*i50 /
    p(i,i) permutation;
Alias (i,ii);
Parameter c(i,i) cost of permutation;
c(i,ii) = uniform(-50,50);

embeddedCode Python:
import random
pauseEmbeddedCode

Set    iter / 1*100 /;
Scalar tcost
       minTCost / +inf /;
loop(iter,
  continueEmbeddedCode:
  i = list(gams.get("i"))
  p = list(i)
  random.shuffle(p)
  gams.set("p", [ (i[idx], p[idx]) for idx in range(len(i)) ])
  pauseEmbeddedCode p
  tcost = sum(p, c(p));
  if (tcost < minTCost, minTCost = tcost);
);
continueEmbeddedCode:
pass
endEmbeddedCode
Display minTCost;
```

The last embedded code execution of the Python `pass` statement is to clean up and terminate the Python session. Since the amount of data is relatively small, there is little difference in the running time between these variants.

There are other performance considerations. We can actually extract the set `i` once before the GAMS loop starts. Moreover, we can work with label indexes rather than the labels itself. Indexes are integers and are often faster than labels that are stored as strings. The only difference to the code above is the extraction method of the Python list `i` by `i = list(gams.get("i",keyType=KeyType.INT))`.

4.45.5.8 Extending GMSPython

While we recommend to use your own Python installation if you need additional packages, there are ways to extend the Python system that comes with GAMS in `GMSPython`. Here are the steps:

1. Get pip via get-pip (https://pip.pypa.io/en/stable/installation/)
2. Update pip
3. Install additional packages

We highly recommend installing pip and additional packages in the user site (use --user when running get-pip and pip). This is especially important for macOS users. Installing files in the GAMS system directory may interfere with the file notarization and may prevent GAMS from starting properly. Here is a typical dialog for Linux/macOS using curl to download get-pip.py. If you don't have curl or wget, use your web browser for downloading.

```
curl https://bootstrap.pypa.io/get-pip.py -o get-pip.py
/path/to/gams/GMSPython/python get-pip.py --user
~/.local/bin/pip install -U pip
~/.local/bin/pip install geocoder --user
```

4.45.5.9 Using an External Python 3 Installation

Let's assume you don't want to work with the GAMS distributed GMSPython but you want to connect GAMS with a Python 3 installation of your liking. In order for the Embedded Code Facility to work, you need to make the alternative Python interpreter aware of the required modules provided by GAMS by following the Getting Started steps in the Python API documentation.

In addition to this you need to do one more step to allow GAMS to find your Python installation when executing Embedded Python code: you need to point to the Python dynamic load library or shared object. On Windows this file is called python3X.dll, on Linux libpython3.X.so, and on macOS libpython3.X.dylib. The X stands for the minor version number of Python 3. While embedded code has been built and tested for Python 3.12, chances are that you can also point to other Python 3 libraries. Since the expert level API is required, the set of versions is limited to what GAMS supports (currently GAMS ships these for Python 3.8 to 3.12). Limited experiments with the embedded code library for Python 3.12 with Python 3.8, 3.9, 3.10, and 3.11 libraries were successful. You need to set the environment variable GMSPYTHONLIB to point GAMS to the Python library:

- Windows:
  ```
  set GMSPYTHONLIB=c:\path\to\python\python312.dll
  ```
- Linux:
  ```
  export GMSPYTHONLIB=/path/to/python/lib/libpython3.12.so
  ```
- macOS:
  ```
  export GMSPYTHONLIB=/path/to/python/lib/libpython3.12.dylib
  ```

GAMS will extract the Python home from the location of the Python library. In case this extraction does not work (because of an unconventional Python installation), you can instruct GAMS to set the Python home via the additional environment variable GMSPYTHONHOME. Please note, that this environment variable only needs to be set for GAMS Embedded Python code, so it is sufficient to add these variables to the GAMS configuration YAML file.

Note

> When using environment-based Python distributions like Anaconda or Miniconda, it is not sufficient to set GMSPYTHONLIB (and GMSPYTHONHOME). The environment to be used needs to be activated properly before (e.g. conda activate myEnv). Packages like numpy which have been installed using conda (conda install numpy) won't work otherwise. In addition to setting GMSPYTHONLIB we therefore recommend to start GAMS itself and also tool like GAMS Studio from an activated conda environment in order to make sure that the environment and all installed packages work as expected.

Although it is recommended to use a self-contained Python installation, it is possible to use a virtual environment instead (e.g. via venv). In this case, GMSPYTHONLIB has to be set to the Python library of the installation from which the environment was derived from. In addition, the PYTHONPATH environment variable needs to be set to the site-packages directory of the virtual environment in order to make its packages available. Note that with this setup the site-packages directory of the original Python installation will also be found by GAMS Embedded Python code.

4.45.5.10 Building your own Embedded Python Code Library

Although the Embedded Code Facility and its binary components are part of the GAMS distribution, it is possible to build it manually from source using the following commands. The exact command line might change depending on the compiler and the operating system in use:

- Windows:

```
cd [GAMS directory]/apifiles/C/api
icl.exe -Feembpycclib64.dll -IC:\path\to\python\include embpyoo.c gmdcc.c gclgms.c -LD -link -no
```

- Linux:

```
cd [GAMS directory]/apifiles/C/api
gcc -o libembpycclib64.so -nostartfiles -shared -Wl,-Bsymbolic -pthread -fPIC -Ipath/to/your/pyt
```

- macOS on x86_64 CPUs:

```
cd [GAMS directory]/apifiles/C/api
gcc -o libembpycclib64.dylib -dynamiclib -shared -install_name @rpath/libembpycclib64.dylib -Ipa
```

- macOS on arm64 CPUs:

```
cd [GAMS directory]/apifiles/C/api
clang -o libembpycclib64.dylib -dynamiclib -shared -install_name @rpath/libembpycclib64.dylib -I
```

4.45.6 Connect

The Connect framework gives unified and platform independent access to data exchange with different formats (e.g. CSV and Excel). The instructions how to access the various data sources are given in YAML syntax. These YAML instructions are the *code* for the embedded *Connect* code. The embedded Connect code engine will parse the YAML instructions and use Connect *agents* to act upon the instructions. The agents GAMSReader and GAMSWriter send data back and forth to GAMS. A simple example is given above.

Embedded code Connect is build on top of embedded code Python. Similar considerations (e.g. Python interpreter is loaded once) carry over from Python to Connect. Moreover, the agent PythonCode has access to the object `gams` of type `ECGamsDatabase` which is used for communication of data with GAMS. In contrast to embedded Python code, the code of the agent `PythonCode` does not execute in the global scope but in a separate block. This means that variables set in the code are not available in another `PythonCode` section. The following code ends up in the except clause and prints `Cannot access my_i`.

```
embeddedCode Connect:
- PythonCode:
    code: |
        my_i = 1
- PythonCode:
    code: |
        try:
          gams.printLog(f'{my_i}')
        except:
          gams.printLog('Cannot access my_i')
endEmbeddedCode
```

Since Python allows to add attributes to an object at runtime, the `my_i` variable can be added to the `gams` object and hence becomes available in different `PythonCode` sections and even in different embedded Connect code parts (when pause/continue is used):

```
embeddedCode Connect:
- PythonCode:
    code: |
        gams.my_i = 1
- PythonCode:
    code: |
        gams.printLog(f'{gams.my_i}')
pauseEmbeddedCode
continueEmbeddedCode:
- PythonCode:
    code: |
        gams.printLog(f'{gams.my_i}')
endEmbeddedCode
```

Even though embedded code Python and Connect share the same basis, they run in different instantiations of the Python interpreter and hence don't share e.g. the same `gams` object.

4.45.6.1 Substitutions in Embedded Connect Code

The GAMS compiler has mechanisms to substitute compile-time variables to parameterize GAMS code including embedded code (unless the `V`, e.g. `$onEmbeddedCodeV`, of embedded code is used). Here is an example how to use double-dash parameters to allow the user to set the CSV file name via the GAMS command line (`gams mymodel.gms --csvfilename=mystockprice.csv`):

```
$if not set CSVFILENAME $set CSVFILENAME stockprice.csv
$onEmbeddedCode Connect:
- CSVReader:
    file: %CSVFILENAME%
    name: stockprice
    indexColumns: [1, 2]
    valueColumns: [3]
- GAMSWriter:
    symbols:
      - name: stockprice
$offEmbeddedCode
```

Not only user defined compile-time variables will be substituted, but also all other compile time variables, e.g. `%gams.scrDir%` or `%system.dirSep%`. Another way to pass substitution instructions down to embedded Connect code are the embedded code `arguments`:

```
$if not set MYCSVFILENAME $set MYCSVFILENAME stockprice.csv
$onEmbeddedCodeV Connect: --CSVFILENAME=%MYCSVFILENAME%
- CSVReader:
    file: %CSVFILENAME%
    name: stockprice
    indexColumns: [1, 2]
    valueColumns: [3]
- GAMSWriter:
    symbols:
      - name: stockprice
$offEmbeddedCode
```

Together with the put_utility command ECArguments that appends some arbitrary text to the embedded code arguments this can be used to parameterize the embedded Connect code at execution time, e.g. to create files based on label names:

```
Set i /1*5/; Parameter p(i);
loop(i,
   p(i) = uniform(0,1);
   put_utility 'ECArguments' / '--CSVFILENAME=' i.tl:0 '.csv';
   EmbeddedCodeV Connect:
   - GAMSReader:
       symbols:
         - name: p
   - CSVWriter:
       name: p
       file: %CSVFILENAME%
   endEmbeddedCode
);
```

Please note that while the GAMS command line parameters allow various formats to specify double-dash parameters, the embedded code Connect substitution arguments must be in the form `--key=value`, so a `=` character to separate key and value with no spaces. The `value` might be a (double) quoted string though. Please also note that the embedded code arguments set with put_utility command `ECArguments` will be applied to any subsequent embedded code section. So one might want to clear the argument by adding a `put_utility "ECArguments" / "";` after the relevant embedded code section.

4.45.7 GAMS

The embedded code GAMS differs in various ways from the other embedded code *languages*. It's definitely not a *new* language one gets access to or does things not easily possible in GAMS. Moreover, it does the data communication *not* in-memory but via disk files. Mostly it adds some syntactical sugar to methods to call GAMS from within GAMS that are available for a long time. So a compile time embedded code GAMS section

```
$onEmbeddedCode GAMS: args
* ...
$offEmbeddedCode list_of_symbols
```

essentially can be done via some file creation via `$on/offEcho`, spawning of a GAMS process via `$call`, and loading of symbols via `$gdxLoad`:

```
$onEcho > sub.gms
* ...
$offEcho
$call.checkErrorLevel gams sub.gms args gdx=sub.gdx
$gdxLoad sub.gdx list_of_symbols
```

Similar at execution time an embedded code GAMS section

```
embeddedCode GAMS: args
* First part ...
pauseEmbeddedCode list_of_symbols
continueEmbeddedCode GAMS:
* Second part ...
endEmbeddedCode list_of_symbols
```

essentially can be done via some file creation via a put file, spawning of a GAMS process via `execute`, and loading of symbols via `execute_loadpoint`:

```
file fgms / 'sub.gms' /; put fgms;
$onPut
* First part ...
$offPut
putclose;
execute.checkErrorLevel 'gams sub.gms args gdx=sub.gdx save=sub'
execute_loadpoint 'sub.gdx', list_of_symbols;

put fgms;
$onPut
* Second part ...
$offPut
putclose;
execute.checkErrorLevel 'gams sub.gms args gdx=sub.gdx restart=sub'
execute_loadpoint 'sub.gdx', list_of_symbols;
```

In addition to the more readable syntax provided by the embedded code syntax, embedded code GAMS takes care of files (source, log, listing, GDX etc) and options related to these files, e.g. logFile. In case of execution-time `pauseEmbeddedCode`/`continueEmbeddedCode` embedded code handles the management of the save and restart operations which is especially convenient when working with multiple independent embedded code sections in combination with function embeddedHandle. Moreover, the log and listing streams of the embedded code GAMS section become part of the outer GAMS log and listing stream in case the GAMS subprocess returns with an error. For example, the following code with a syntax error (symbol names can't start with _) in the embedded code section:

```
embeddedCode GAMS:
* This produces an error
set _i /1*10/;
endEmbeddedCode
```

results in the following relevant part of the log:

```
...
--- Initialize embedded library embgamscclib64.dll
--- Execute embedded library embgamscclib64.dll
*** Error running gams (return code 2)
--- Start of log file ecgamslog.dat:
    --- Job myEmb.dat Start 01/18/23 12:24:00 42.0.0 3dcbfd09 WEX-WEI x86 64bit/MS Windows
    ...
    --- Starting compilation
    --- myEmb.dat(2) 3 Mb 1 Error
    *** Error   2 in C:\Users\mbuss\Downloads\225e\myEmb.dat
        Identifier expected
    --- GDX File C:\Users\mbuss\Downloads\225e\ecgamsgdx.dat
    --- myEmb.dat(2) 3 Mb 1 Error
    *** Status: Compilation error(s)
    --- Job myEmb.dat Stop 01/18/23 12:24:00 elapsed 0:00:00.007
--- End of log file ecgamslog.dat
*** Error at line 1: Error executing "embeddedCode" section: Inspect listing file for details
...
```

and following relevant part of the listing file:

```
...
E x e c u t i o n

--- Start of include of file ecgamslst.dat
GAMS 42.1.0  3dcbfd09 Jan 30, 2023   WEX-WEI x86 64bit/MS Windows - 01/31/23 12:24:00 Page 1
G e n e r a l   A l g e b r a i c   M o d e l i n g   S y s t e m
C o m p i l a t i o n


   1  * This produces an error
   2  set _i /1*10/;
****      $2
****   2  Identifier expected

**** 1 ERROR(S)   0 WARNING(S)

...
**** USER ERROR(S) ENCOUNTERED
--- End of include of file ecgamslst.dat
**** Exec Error at line 1: Error executing "embeddedCode" section: Inspect listing file for details
...
```

4.45.7.1 Best Practices

Embedded code GAMS at compile time can be used to build up data, e.g. domain sets, required at compile time inside an embedded code GAMS at execution time. Data from the embedded code GAMS section can be moved via GDX (see example above) or file ASCII input files into the outer GAMS program. The following two examples complements the bin packing example by demonstrating two ways to communicate the data via an ASCII input file.

The following solution creates a global compile time variable for the size of set `j`:

```
...
$save.keepCode bp

$onEmbeddedCode GAMS: restart=bp
scalar size /0/;
nj = 1; loop(i, size = size+s(i); if (size>B, nj = nj+1; size = s(i)));
file fnj / 'nj.gms' /; put fnj '$setGlobal NJ ' nj:0:0;
$offEmbeddedCode
$include nj
Set j 'bins' / b1*b%NJ% /;
...
```

In the next solution the embedded code GAMS writes already the data statement for set `j`:

```
...
$save.keepCode bp

$onEmbeddedCode GAMS: restart=bp
scalar jcnt, size /0/;
nj = 1; loop(i, size = size+s(i); if (size>B, nj = nj+1; size = s(i)));
```

```
file fnj / 'nj.gms' /; put fnj '* Set j:';
* Either as short hand b1*bN or all individual elements
if (1=0,
  put / 'b1*b' nj:0:0;
else
  for (jcnt=1 to nj, put / 'b' jcnt:0:0);
);
$offEmbeddedCode
Set j 'bins' /
$include nj
/;
...
```

Embedded code GAMS works best in combination with restarting from a save file. This makes the embedded code short because it can utilize the symbol declarations and definitions from the restart file. Save files can be created at compile time via $save[.keepCode] fileName and at execution time via put_utility 'save' / 'fileName';.

Execution time embedded code GAMS can be useful if at execution time GAMS needs to deal with new labels as in the following sophisticated example. Here we build on top of the GAMS Model Library **trnsport** model and represent the Jacobian matrix of the LP model utilizing the Convert option dumpGDX in the original name space inside GAMS:

```
...
Model transport / all /;

* Generate the Jacobian (t.gdx) with e1,e2,e3,... and x1,x2,x3,... and a mapping (tdm.gdx) to map the
* scalar elements e1,e2,e3,... and x1,x2,x3,... to the original name space:
$onEcho > convert.opt
dumpGDX t.gdx
dictMap tdm.gdx
$offEcho
transport.optfile = 1; option lp=convert;
solve transport using lp minimizing z;

Sets
    v 'variables block names' / x, z /,
    e 'equation block names' / cost, supply, demand /
    empty 'the empty string' / '' /;
Parameter jac(e,*,*,v,*,*) 'Jacobian of the generated LP model';

embeddedCode GAMS:
$call.checkErrorLevel gdxdump tdm.gdx noData > tdm.gms
$include tdm
Parameter A(i,j) 'Jacobian in e1,e2,.. (set i) and x1,x2,... (set j) space', jac;
$gdxLoad t.gdx A
Alias (*,ti,tii,tj,tjj);
loop((cost_EM(i)      ,x_VM(j,ti,tj)), jac('cost'  ,'' ,'','x',ti,tj) = A(i,j));
loop((cost_EM(i)      ,z_VM(j)      ), jac('cost'  ,'' ,'','z','','') = A(i,j));
loop((supply_EM(i,tii),x_VM(j,ti,tj)), jac('supply',tii,'','x',ti,tj) = A(i,j));
loop((supply_EM(i,tii),z_VM(j)      ), jac('supply',tii,'','z','','') = A(i,j));
loop((supply_EM(i,tjj),x_VM(j,ti,tj)), jac('demand','',tjj,'x',ti,tj) = A(i,j));
loop((supply_EM(i,tjj),z_VM(j)      ), jac('demand','',tjj,'z','','') = A(i,j));
endEmbeddedCode jac

option jac:3:3:3; display jac;
```

The display of the last line in the code provides the following output in the listing file:

```
----     87 PARAMETER jac

                                    x           x           x           x           x           x
                              seattle     seattle     seattle   san-diego   san-diego   san-diego
                             new-york     chicago      topeka    new-york     chicago      topeka

cost   .          .            -0.225      -0.153      -0.162      -0.225      -0.162      -0.126
supply.seattle    .             1.000       1.000       1.000
supply.san-diego.                                                   1.000       1.000       1.000
demand.           .seattle      1.000       1.000       1.000
demand.           .san-diego                                        1.000       1.000       1.000
```

Other embedded code languages allow inside the language to set instructions to either filter/domain check or replace/merge data that comes back into GAMS. At compile time the dollar control options $on/offFiltered and $on/offMulti[R] impact the load of symbols from embedded code sections, including GAMS. At execution time embedded code GAMS *filters* and *merges* the data. If a different behavior is required, the symbols to be loaded should not be listed on the `end/pauseEmbeddedCode` line but should be loaded with the appropriate execute_load*** call like `execute_load`, `execute_loaddc`, or `execute_loadpoint`. The GDX file to load can be set via `put_utility "gdxIn" / gams.scrDir "ecgamsgdx." gams.scrExt;`:

```
Set ii / i0*i10 /, i / i1*i5 /; Parameter p(i) / i1 1, i3 1, i5 1 /;
embeddedCode GAMS:
Parameter p / i2 1, i4 1, i6 1 /;
endEmbeddedCode p
display p;
embeddedCode GAMS:
Parameter p / i2 1, i4 1, i6 1 /;
endEmbeddedCode
put_utility 'gdxIn' / gams.scrDir 'ecgamsgdx.' gams.scrExt;
execute_load p;
display p;
```

GAMS' first load of `p` filters and merges the symbol into the existing `p` resulting in `i1 1, i2 1, i3 1, i4 1, i5 1` while the second load replaces the symbol resulting in `i2 1, i4 1`. An `execute_loaddc p;` would have resulted in a domain violation (`i6`) and would have triggered an execution error.

4.46 GAMS Connect

4.46.1 Concept

GAMS Connect is a framework inspired by the concept of a so-called ETL (extract, transform, load) procedure that allows to integrate data from various data sources. The GAMS Connect framework consists of the Connect database and the Connect agents that operate on the Connect database. Via the available Connect interfaces the user passes instructions to call Connect agents for reading data from various file types into the Connect database, transforming data in the Connect database, and writing data from the Connect database to various file types. Instructions are passed in YAML syntax. Note that in contrast to a typical ETL procedure, read, transform and write operations do not need to be strictly separated.

4.46.2 Usage

GAMS Connect is available via the GAMS command line parameters ConnectIn and ConnectOut, via embedded code Connect, and as a standalone command line utility gamsconnect.

Instructions processed by the GAMS Connect interfaces need to be passed in YAML syntax as follows:

```
- <agent name1>:
    <root option1>: <value>
    <root option2>: <value>
    .
    .
    <root option3>:
      - <option1>: <value>
        <option2>: <value>
        .
        .
      - <option1>: <value>
        <option2>: <value>
        .
        .
      .
      .
- <agent name2>:
    .
    .
.
.
```

The user lists the tasks to be performed successively. All tasks begin at the same indentation level starting with a `-` (a dash and a space) followed by the Connect agent name and a `:` (a colon). Connect agent options are represented in a simple `<option>: <value>` form. Please check the documentation of Connect Agents for available options. Options at the first indentation level are called `root` options and typically define general settings, e.g. the file name. While some agents only have `root` options, others have a more complex options structure, where a root option may be a list of dictionaries containing other options. A common example is the root option `symbols` (see e.g. GDXReader). Via `symbols` many agents allow to define symbol specific options, e.g. the name of the symbol. The option tables of agents with a more complex options structure provide a *Scope* to reflect this structure - options may be allowed at the first indentation level (`root`) and/or are assigned to other root options (e.g. `symbols`).

Note that YAML syntax also supports an abbreviated form for lists and dictionary, e.g. `<root option3>: [ {<option1>: <value>, <option2>: <value>}, {<option1>: <value>, <option2>: <value>} ]`.

Here is an example that uses embedded Connect code to process instructions:

```
$onecho > distance.csv
i;j;distance in miles
seattle;new-york;2,5
seattle;chicago;1,7
seattle;topeka;1,8
san-diego;new-york;2,5
san-diego;chicago;1,8
san-diego;topeka;1,4
$offecho
```

```
$onecho > capacity.csv
i,capacity in cases
seattle,350.0
san-diego,600.0
$offecho

Set i 'Suppliers', j 'Markets';
Parameter d(i<,j<) 'Distance', a(i) 'Capacity';
$onEmbeddedCode Connect:
- CSVReader:
    file: distance.csv
    name: distance
    indexColumns: [1, 2]
    valueColumns: [3]
    fieldSeparator: ';'
    decimalSeparator: ','
- CSVReader:
    file: capacity.csv
    name: capacity
    indexColumns: [1]
    valueColumns: [2]
- GAMSWriter:
    symbols:
      - name: distance
        newName: d
      - name: capacity
        newName: a
$offEmbeddedCode

display i, j, d, a;
```

In this example, we are reading two CSV files `distance.csv` and `capacity.csv` using the CSVReader. Then we directly write to symbols in GAMS using the GAMSWriter.

Note that even though GAMS is case insensitive, GAMS Connect is **case sensitive**, i.e., YAML instructions are treated case sensitive. This also includes, e.g., indices in CSV files. Consider the following example where the index `j2` should be substituted by `ABC` when reading the CSV file `y.csv`:

```
$onecho > y.csv
i1,j1,2.5
i1,J2,1.7
i2,j1,1.8
i2,j2,1.4
$offecho

set i,j;
parameter p(i<,j<);

$onEmbeddedCode Connect:
- CSVReader:
    file: y.csv
    name: p
    indexColumns: [1,2]
    valueColumns: [3]
    header: false
    indexSubstitutions: { j2: ABC }
- GAMSWriter:
```

```
    writeAll: true
$offEmbeddedCode

display i,j,p;
```

Since the YAML instructions are treated case sensitive, the index `J2` will not be substituted.

```
            j1          J2          ABC

i1       2.500       1.700
i2       1.800                   1.400
```

An exception are symbol names in the Connect database. Creating or accessing symbols, e.g. via the `name` option of many agents, is case insensitive.

All instructions provided to the Connect framework are read using `UTF-8` encoding (`utf-8-sig`). This can be customized by adding a comment in the format `# coding=<encoding name>` or `# -*- coding: <encoding name> -*-` as first line in the YAML code. Note that `UTF-16` encoding is not supported.

A note on the **sorting behavior** of Connect agents. All reader and transformer agents do not guarantee a specific order of the created symbol records. However, as the symbol records in the Connect database are saved in *categorical data structures*, the order in the data source is preserved in *ordered categories*. All writer agents guarantee that symbol records are written in the order of the categories. Here is an example:

```
$onecho > ijk_in.csv
i,j,k1,k2
i1,j2,,3
i2,j1,4,
i1,j1,1,2
i2,j2,5,6
$offecho

EmbeddedCode Connect:
- CSVReader:
    file: ijk_in.csv
    name: ijk
    valueColumns: "3:lastCol"
    indexColumns: [1,2]
- PythonCode:
    code: |
        sym = connect.container.data["ijk"]
        print("ijk records in the Connect database after reading:\n", sym.records)

        data_sorted = sym.records.sort_values(sym.records.columns[:-1].tolist())
        print("ijk records sorted according to categories:\n", data_sorted)

        sym.reorderUELs(uels=['j1','j2'], dimensions=1)
        data_sorted = sym.records.sort_values(sym.records.columns[:-1].tolist())
        print("ijk records sorted according to reordered categories:\n", data_sorted)
- CSVWriter:
    file: ijk_out.csv
    name: ijk
    unstack: True
endEmbeddedCode
```

From the data source `ijk_in.csv`, the CSVReader creates the symbol records as shown with the first print:

```
ijk records in the Connect database after reading:
     i   j level_2  value
0  i1  j2      k2    3.0
1  i2  j1      k1    4.0
2  i1  j1      k1    1.0
3  i1  j1      k2    2.0
4  i2  j2      k1    5.0
5  i2  j2      k2    6.0
```

The ordered categories are inferred from the data source, therefore the order for column `i` is `['i1','i2']`, for column `j` `['j2','j1']` and for column `k` the order of the header `['k1','k2']` is preserved. The categories define the order of the symbol records if sorted:

```
ijk records sorted according to categories:
     i   j level_2  value
0  i1  j2      k2    3.0
2  i1  j1      k1    1.0
3  i1  j1      k2    2.0
4  i2  j2      k1    5.0
5  i2  j2      k2    6.0
1  i2  j1      k1    4.0
```

If the order is not as desired, `.reorderUELs()` can be used to change the order of the categories, e.g. to `['j1','j2']` for column `j`:

```
ijk records sorted according to reordered categories:
     i   j level_2  value
2  i1  j1      k1    1.0
3  i1  j1      k2    2.0
0  i1  j2      k2    3.0
1  i2  j1      k1    4.0
4  i2  j2      k1    5.0
5  i2  j2      k2    6.0
```

The CSVWriter automatically sorts the symbol records according to the categories (note that the order of column `j` was changed with the PythonCode agent) and therefore, the content of file `ijk_out.csv` looks as follows:

```
i,j,k1,k2
i1,j1,1.0,2.0
i1,j2,,3.0
i2,j1,4.0,
i2,j2,5.0,6.0
```

4.46.3 Connect Agents Summary

Current Connect agents support the following data source formats: CSV, Excel, GDX and SQL. The following Connect agents are available:

Connect agent	Description	Supported symbol types
Concatenate	Allows concatenating multiple symbols in the Connect database.	Sets and parameters
CSVReader	Allows reading a symbol from a specified CSV file into the Connect database.	Sets and parameters
CSVWriter	Allows writing a symbol in the Connect database to a specified CSV file.	Sets and parameters
DomainWriter	Allows rewriting the domain information of an existing Connect symbol.	Sets, parameters, variables, and equations
ExcelReader	Allows reading symbols from a specified Excel file into the Connect database.	Sets and parameters
ExcelWriter	Allows writing symbols in the Connect database to a specified Excel file.	Sets and parameters
Filter	Allows to reduce symbol data by applying filters on labels and numerical values.	Sets, parameters, variables, and equations
GAMSReader	Allows reading symbols from the GAMS database into the Connect database.	Sets, parameters, variables, and equations
GAMSWriter	Allows writing symbols in the Connect database to the GAMS database.	Sets, parameters, variables, and equations
GDXReader	Allows reading symbols from a specified GDX file into the Connect database.	Sets, parameters, variables, and equations
GDXWriter	Allows writing symbols in the Connect database to a specified GDX file.	Sets, parameters, variables, and equations
LabelManipulator	Allows to modify labels of symbols in the Connect database.	Sets, parameters, variables, and equations
Options	Allows to set more general options that can affect the Connect database and other Connect agents.	-
Projection	Allows index reordering and projection onto a reduced index space of a GAMS symbol.	Sets, parameters, variables, and equations
PythonCode	Allows executing arbitrary Python code.	-
RawCSVReader	Allows reading unstructured data from a specified CSV file into the Connect database.	-
RawExcelReader	Allows reading unstructured data from a specified Excel file into the Connect database.	-
SQLReader	Allows reading symbols from a specified SQL database into the Connect database.	Sets and parameters
SQLWriter	Allows writing symbols in the Connect database to a specified SQL database.	Sets and parameters

4.46.4 Getting Started

We introduce the basic functionalities of GAMS Connect agents on some simple examples. For more examples see section Examples.

4.46.4.1 Simple Connect Example for CSV

The following example (a modified version of the **trnsport** model) shows how to read and write CSV files. The full example is part of DataLib as model **connect03**. Here is a code snippet of the first lines:

```
$onEcho > distance.csv
i,new-york,chicago,topeka
seattle,2.5,1.7,1.8
san-diego,2.5,1.8,1.4
$offEcho

$onEcho > capacity.csv
i,capacity
seattle,350
san-diego,600
$offEcho

$onEcho > demand.csv
j,demand
new-york,325
chicago,300
topeka,275
$offEcho

Set i 'canning plants', j 'markets';

Parameter d(i<,j<) 'Distance', a(i) 'Capacity', b(j) 'Demand';

$onEmbeddedCode Connect:
- CSVReader:
    file: distance.csv
    name: d
    indexColumns: 1
    valueColumns: "2:lastCol"
- CSVReader:
    file: capacity.csv
    name: a
    indexColumns: 1
    valueColumns: 2
- CSVReader:
    file: demand.csv
    name: b
    indexColumns: 1
    valueColumns: 2
- GAMSWriter:
    writeAll: True
$offEmbeddedCode

[...]
```

It starts out with the declaration of sets and parameters. With compile-time embedded Connect code, data for the parameters is read from CSV files using the Connect agent CSVReader. The CSVReader agent, for example, reads the CSV file `distance.csv` and creates the parameter `d` in the Connect database. The name of the parameter must be given by the option name. Column number 1 is specified as the first domain set using option indexColumns. The valueColumns option is used to specify the column numbers 2, 3 and 4 containing the values. Per default, the first row of the columns specified via `valueColumns` will be used as the second domain set. The symbolic constant `lastCol` can be used if the number of index or value columns is unknown. As a last step, all symbols from the Connect database are written to the GAMS database using the Connect agent GAMSWriter. The GAMSWriter agent makes the parameters `d`, `a` and `b` available outside the embedded Connect code. Note that the sets `i` and `j` are defined implicitly through parameter `d`.

Finally, after solving the `transport` model, Connect can be used to export results to a CSV file:

```
[...]

Model transport / all /;
solve transport using lp minimizing z;

EmbeddedCode Connect:
- GAMSReader:
    symbols:
        - name: x
- Projection:
    name: x.l(i,j)
    newName: x_level(i,j)
- CSVWriter:
    file: shipment_quantities.csv
    name: x_level
    unstack: True
endEmbeddedCode
```

This time, we need to use execution-time embedded Connect code. The Connect agent GAMSReader imports variable `x` into the Connect database. With the Connect agent CSVWriter we write the variable level to the CSV file `shipment_quantities.csv`:

```
i_0,new-york,chicago,topeka
seattle,50.0,300.0,0.0
san-diego,275.0,0.0,275.0
```

Setting the option unstack to `True` allows to use the last dimension as the header row.

4.46.4.2 Simple Connect Example for Excel

The following example is part of GAMS Model Library as model **cta** and shows how to read and write Excel spreadsheets. Here is a code snippet of the first lines:

```
Set
   i 'rows'
   j 'columns'
   k 'planes';

Parameter
   dat(k<,i<,j<) 'unprotected data table'
   pro(k,i,j)    'information sensitive cells';

* extract data from Excel workbook
$onEmbeddedCode Connect:
- ExcelReader:
    file: cox3.xlsx
    symbols:
      - name: dat
        range: Sheet1!A1
        rowDimension: 2
        columnDimension: 1
      - name: pro
        range: Sheet2!A1
        rowDimension: 2
```

```
        columnDimension: 1
- GAMSWriter:
    writeAll: True
$offEmbeddedCode

[...]
```

It starts out with the declaration of sets and parameters. With compile-time embedded Connect code, data for the parameters is read from the Excel file `cox3.xlsx` using the Connect agent ExcelReader. The ExcelReader agent allows reading data for multiple symbols that are listed under the keyword `symbols`, here, parameter `dat` and `pro`. For each symbol, the symbol name is given by option name and the Excel range by option range. The option rowDimension defines that the first two columns of the data range will be used for the labels. In addition, the option columnDimension defines that the first row of the data range will be used for the labels. As a last step, all symbols from the Connect database are written to the GAMS database using the Connect agent GAMSWriter. The GAMSWriter agent makes the parameters `dat` and `pro` available outside the embedded Connect code. Note that the sets `i`, `j` and `k` are defined implicitly through parameter `dat`.

Finally, after solving the `cox3c` model with alternative solutions, Connect can be used to export results to Excel:

```
[...]

loop(l$((obj.l - best)/best <= 0.01),
   ll(l) = yes;
   binrep(s,l)             = round(b.l(s));
   binrep('','','Obj',l)   = obj.l;
   binrep('','','mSec',l)  = cox3c.resUsd*1000;
   binrep('','','nodes',l) = cox3c.nodUsd;
   binrep('Comp','Cells','Adjusted',l) = sum((i,j,k)$(not s(i,j,k)), 1$round(adjn.l(i,j,k) + adjp.l(
   solve cox3c min obj using mip;
);

embeddedCode Connect:
- GAMSReader:
    symbols:
      - name: binrep
- ExcelWriter:
    file: results.xlsx
    clearSheet: True
    symbols:
      - name: binrep
endEmbeddedCode
```

This time, we need to use execution-time embedded Connect code. The Connect agent GAMSReader imports the reporting parameter `binrep` into the Connect database. With the Connect agent ExcelWriter we write the parameter into the `binrep` sheet of the Excel file `results.xlsx`.

4.46.4.3 Simple Connect Example for SQL

The following example (a modified version of the **whouse** model) shows how to read from and write to a SQL database (sqlite). The full example is part of DataLib as model **connect04**. Here is a code snippet of the first lines:

```
[...]

Set t 'time in quarters';

Parameter
   price(t)  'selling price ($ per unit)'
   istock(t) 'initial stock      (units)';

Scalar
   storecost 'storage cost  ($ per quarter per unit)'
   storecap  'stocking capacity of warehouse (units)';

$onEmbeddedCode Connect:
- SQLReader:
    connection: {"database": "whouse.db"}
    symbols:
      - name: t
        query: "SELECT * FROM timeTable;"
        type: set
      - name: price
        query: "SELECT * FROM priceTable;"
      - name: istock
        query: "SELECT * FROM iniStockTable;"
      - name: storecost
        query: "SELECT * FROM storeCostTable;"
      - name: storecap
        query: "SELECT * FROM storeCapTable;"
- GAMSWriter:
    writeAll: True
$offEmbeddedCode

[...]
```

It starts out with the declaration of sets and parameters. With compile-time embedded Connect code, data for all the symbols are read from the sqlite database `whouse.db` using the Connect agent SQLReader by passing the connection url through the option connection. The SQLReader agent, for example, queries the table `priceTable` for data and creates the parameter `price` in the Connect database. The SQLReader allows reading data for multiple symbols that are listed under the keyword `symbols` and are fetched through the same connection. For each symbol the name must be given by the option name. The SQL query statement is passed through the option query. The symbol type can be specified using the option type. By default, every symbol is treated as a GAMS parameter. As a last step, all symbols from the Connect database are written to the GAMS database using the Connect agent GAMSWriter. The GAMSWriter agent makes all read in symbols available outside the embedded Connect code.

Further, after solving the warehouse model, Connect can be used to export the results to tables in the SQL database.

```
[...]

Model swp 'simple warehouse problem' / all /;

solve swp minimizing cost using lp;


EmbeddedCode Connect:
- GAMSReader:
    readAll: True
```

```
- Projection:
    name: stock.l(t)
    newName: stock_level(t)
- Projection:
    name: sell.l(t)
    newName: sell_level(t)
- Projection:
    name: buy.l(t)
    newName: buy_level(t)
- SQLWriter:
    connection: {"database": "whouse.db"}
    ifExists: replace
    symbols:
      - name: stock_level
        tableName: stock_level
      - name: sell_level
        tableName: sell_level
      - name: buy_level
        tableName: buy_level
endEmbeddedCode
```

Here, we need to use execution-time embedded Connect code. The Connect agent GAMSReader imports all the variables into the Connect database. The SQLWriter agent then writes each symbol to respective tables in the SQL database `whouse.db`. For example the stock level:

```
|t_0     |level     |
|:-------|:---------|
|q-1     |100.0     |
|q-2     |0.0       |
|q-3     |0.0       |
|q-4     |0.0       |
```

The ifExists option allows to either append to an extending table or replace it with new data. By default, the value for `ifExists` is set to `fails`.

4.46.5 Connect Agents

4.46.5.1 Concatenate

The Concatenate agent allows concatenating multiple symbols (sets or parameters) in the Connect database into a single symbol of the same type. It takes the union of domain sets of all concatenated symbols and uses that as the domain for the output symbol. There are several options to guide this domain finding process which are explained below. The general idea is best explained with an example. Consider three parameters `p1(i,j)`, `p2(k,i)`, and `p3(k,l)`. The union of all domain sets is `i`, `j`, `k`, and `l` and, hence, the output symbol will be `parameterOutput(symbols,i,j,k,l)`. The very first index of `parameterOutput` contains the name of the concatenated symbol followed by the domain sets. If a domain set is not used by a concatenated symbol the corresponding records in `parameterOutput` will feature the emptyUel, a `-` (dash) by default, as the following figures show:

The Concatenate agent is especially useful in combination with UI components that provide a pivot table, like GAMS MIRO, to represent many individual output symbols in a single powerful and configurable table format.

Obviously, there are more complex situations with respect to the domain of the resulting `parameterOutput`. For example, only a subset of domain sets are relevant and the remaining ones should be combined in as few index positions as possible. For this, assume only domain sets `i` and `k` from the above example are relevant and `j` and `l` can be combined in a single index position - a so-called universal domain. The resulting `parameterOutput` would look as follows:

Moreover, the Concatenate agent needs to deal with universe domain sets $*$ and domain sets that are used multiple times in a concatenated symbol. In addition to the `symbols` index (always the first index position of the output symbol), by default the union of domain sets of the concatenated symbols determine the domain of the output symbol. If a domain set (including the universe $*$) appears multiple times in a concatenated symbol domain, these duplicates will be part of the output symbol domain. For example, `q1(*,i,j,*)` and `q2(*,i,i)` will result in the output symbol `parameterOutput(symbols,*,i,j,*,i,)` by default, mapping index positions 1 to 4 of `q1` to positions 2 to 5 of `parameterOutput` and index positions 1 to 3 of `q2` to 2, 3, and 6.

All the described situations can be configured with a few options of the agent. The option outputDimensions allows to control the domain of the output symbol. The default behavior (`outputDimension: all`) gets the domain sets from the concatenated symbols and builds the union with duplicates if required. Alternatively, `outputDimensions` can be a list of the relevant domain sets (including an empty list). In any case, the agent iterates through the concatenated symbols and maps the index positions of a concatenated symbol into the index positions of the output symbol using the domain set names. Names not present in `outputDimensions` will be added as universal domains. Per default, the domain set names of a concatenated symbol will be the original domain set names as stored in the Connect database. There are two ways to adjust the domain set names of concatenated symbols: dimensionMap and an explicitly given domain for a symbol in the name option. The `dimensionMap` which is given once and holds for all symbols allows to map original domain names of concatenated symbols to the desired domain names. The `name` option provides such a map by symbol and via the index position rather than the domain names of the concatenated symbol. In the above example with `p1(i,j)`, `p2(k,i)`, and `p3(k,l)`, we could put indices `i` and `l` as well as `j` and `k` together resulting in the following output symbol:

This can be accomplished in two ways: either we use `dimensionMap: {i: il, l: il, j: jk, k: jk}` or we use `name: p1(il,jk)`, `name: p2(jk,il)`, and `name: p3(jk,il)` to explicitly define the domain names for each symbol. Note that it is not required to set `outputDimensions: [il,jk]` since per default the union of domain sets is built using the mapped domain names. In case a domain set is used more than once in a domain of a concatenated symbol the mapping goes from left to right to find the corresponding output domain. If this is not desired, the Projection agent can be used to reorder index positions in symbols or explicit index naming can be used. In the example with `q1(*,i,j,*)` and `q2(*,i,i)`, the second index position of `q2` will be put together with the second index position of `q1`. If one wants to map the second `i` of `q2` (in the third index position) together with the `i` of `q1` (in second index position), one can, e.g., do with `name: q1(*,i,j,*)`, and `name: q2(*,i2,i)`.

Note

1. The Concatenate agent creates result symbols `parameterOutput` and `setOutput` for parameters and sets separately. Both have the same output domain. If you want different output domains for `parameterOutput` and `setOutput` use two instantiations of the Concatenate agent.
2. Variables and equations need to be turned into parameters with the Projection agent before they can be concatenated.
3. If option `name` is given without an explicit domain for the concatenated symbol, the domain names stored in the Connect container are used and mapped via the `dimensionMap` option, if provided.
4. A domain set of a concatenated symbol that cannot be assigned to an index in `outputDimensions` will be mapped to a so-called universal domain. The Concatenate agent automatically adds as many universal domains as required to the output symbols.

Here is a complete example that uses the Concatenate agent:

```
Sets
   i(i) / i0*i3 "i_text" /
   j(j) / j0*j3 "j_text" /
   k(k) / k0*k3 "k_text" /;

Parameters
   p1(i) / i1 1 /
   p2(k,j) / k1.j0 2, k1.j1 3, k1.j3 4 /
   p3(j,j) / j1.j2 5, j2.j0 6, j3.j1 7, j3.j2 8 /
   s / 5 /;

Positive Variable x(i,j);
x.l(i,j)$(uniform(0,1)>0.8) = uniformint(0,10);

EmbeddedCode Connect:
- GAMSReader:
    readAll: True
- Projection:
    name: x.l(i,j)
    newName: x_level(i,j)
- Concatenate:
    outputDimensions: [j,i]
- GDXWriter:
    file: concat_output.gdx
    symbols:
      - name: setOutput
      - name: parameterOutput
endEmbeddedCode
```

The resulting set and parameter outputs look as follows:

The following options are available for the Concatenate agent.

Option	Scope	Default	Description
concatenateAll	root	`auto`	Indicate if all sets and parameters in the Connect database will be concatenated.
dimensionMap	root	None	Define a mapping for the domain names of concatenated symbols as stored in the Connect database to the desired domain names.
emptyUel	root	`-`	Define a character to use for empty uels.
name	symbols	None	Specify the name of the symbol with potentially index space.
newName	symbols	None	Specify a new name for the symbol in the `symbols` column of the output symbol.
outputDimensions	root	`all`	Define the dimensions of the output symbols.
parameterName	root	`parameterOutput`	Name of the parameter output symbol.
setName	root	`setOutput`	Name of the set output symbol.
skip	root	None	Indicate if sets or parameters should be skipped.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.
universalDimension	root	`uni`	Specify the base name of universal dimensions.

Detailed description of the options:

concatenateAll = *boolean or string* (default=`auto`)

If `True` all sets and parameters in the given database are concatenated, symbols will be ignored. Default is `auto` where the Concatenate agent uses `symbols` if specified, otherwise concatenates all sets and parameters in the Connect database.

dimensionMap = *dict* (optional)

Define a mapping for domain names of concatenated symbols as stored in the Connect database to the desired domain names. For example, `dimensionsMap: {i: ij, j: ij}` will map both symbol domains `i` and `j` to `ij`.

emptyUel = *string* (default=`-`)

Define a character to use for empty uels.

name = *string* (required)

Specify the name of the symbol with potentially index space. Requires the format `symName(i1,i2,...,iN)`. The index space may be specified to establish a mapping for the domain names of the symbol as stored in the Connect database to the desired domain names. If no index space is provided, the domain names stored in the Connect data are used and mapped via the `dimensionMap` option if provided.

newName = *string* (optional)

Specify a new name for the symbol in the `symbols` column of the output symbol.

outputDimensions = *list or string* (default=`all`)

Define the dimensions of the output symbols explicitly using a list, e.g., `outputDimensions: [i,j]`. The default `all` gets the domain sets from the concatenated symbols and builds the union with duplicates if required.

parameterName = *string* (default=`parameterOutput`)

Name of the parameter output symbol.

setName = *string* (default=`setOutput`)

Name of the set output symbol.

skip = *string* (optional)

Indicate if sets or parameters should be skipped. Per default the agent takes both sets and parameters into account (if both are available via symbols or concatenateAll) and generates a set and parameter output symbol with the same domain. If `set` is specified, the sets will be skipped, i.e. sets are not taken into account for setting up the domain and no set output symbol will be generated. If `par` is specified, the parameters will be skipped.

symbols = *list* (optional)

A list containing symbol specific options. Allows to concatenate a subset of symbols.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` $>$ `1` some scalar debugging output will be written to the log. For `trace` $>$ `2` the intermediate data frames will be written abbreviated to the log. For `trace` $>$ `3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

universalDimension = *string* (default=`uni`)

Specify the base name of universal dimensions.

4.46.5.2 CSVReader

The CSVReader allows reading a symbol (set or parameter) from a specified CSV file into the Connect database. Its implementation is based on the `pandas.DataFrame` class and its I/O API method `read_csv`. See Simple Connect Example for CSV for a simple example that uses the CSVReader.

Option	Default	Description
autoColumn	None	Generate automatic column names.
autoRow	None	Generate automatic row labels.
decimalSeparator	. (period)	Specify a decimal separator.
fieldSeparator	, (comma)	Specify a field separator.
file	None	Specify a CSV file path.
header	inferred	Specify the header(s) used as the column names.
indexColumns	None	Specify columns to use as the row labels.
indexSubstitutions	None	Dictionary used for substitutions in the index columns.
name	None	Specify a symbol name for the Connect database.
names	None	List of column names to use.
quoting	0	Control field quoting behavior.
readCSVArguments	None	Dictionary containing keyword arguments for the `pandas.read_csv` method.
skipRows	None	Specify the rows to skip or the number of rows to skip.
stack	inferred	Stacks the column names to index.
textColumns	None	Specify columns to get the set element text from.
textSubstitutions	None	Dictionary used for substitutions in the text columns.
thousandsSeparator	None	Specify a thousands separator.
trace	inherited	Specify the trace level for debugging output.
valueColumns	None	Specify columns to get the values from.
valueSubstitutions	None	Dictionary used for substitutions in the value columns.

Detailed description of the options:

autoColumn = *string* (optional)

Generate automatic column names. The `autoColumn` string is used as the prefix for the column label numbers. This option overrides the use of a `header` or `names`. However, if there is a header row, one must skip the row by enabling `header` or using `skipRows`.

autoRow = *string* (optional)

Generate automatic row labels. The `autoRow` string is used as the prefix for the row label numbers. The generated unique elements will be used in the first index position shifting other elements to the right. Using `autoRow` can be helpful when there are no labels that can be used as unique elements but also to store entries that would be a duplicate entry without a unique row label.

decimalSeparator = *string* (default=`.`)

Specify a decimal separator. [`.` (period), `,` (comma)]

fieldSeparator = *string* (default=`,`)

Specify a field separator. [`,` (comma), `;` (SemiColon), `\t` (Tab)]

file = *string* (required)

Specify a CSV file path.

header = *boolean, list* (optional)

Specify the header(s) used as the column names. Default behavior is to infer the column names: if no names are passed the behavior is identical to `header=True` and column names are inferred from the first line of data, if column names are passed explicitly then the behavior is identical to `header=False`. Explicitly pass `header=True` to be able to replace existing names. Note that missing column names are filled with `Unnamed: n` (where n is the nth column (zero based) in the DataFrame). Hence, reading the CSV file:

```
,j1,
i1,1,2
i2,3,4
,5,6
```

results in the following 2-dimensional parameter:

```
            j1  Unnamed: 2

i1       1.000       2.000
i2       3.000       4.000
```

For a multi-row header, a list of integers can be passed providing the positions of the header rows in the data. Note that reading multi-row header is only supported for parameters. Moreover, the CSVReader can only read all columns and not a subset of columns, wherefore only `indexColumns` can be specified and all other columns will automatically be read as `valueColumns`. Note that `indexColumns` can not be provided as column names together with a multi-row header. `autoRow` and `autoColumn` will be ignored in case of a multi-row header. Here is an example how to read data with a multi-row header:

```
$onecho > multirow_header.csv
j,,j1,j1,j1,j2,j2,j2
k,,k1,k2,k3,k1,k2,k3
h,i,,,,,,
h1,i1,1,2,,4,5,6
h1,i2,,,3,4,5,
$offEcho

$onEmbeddedCode Connect:
- CSVReader:
    file: multirow_header.csv
    name: p
    header: [1,2]
    indexColumns: [1,2]
- PythonCode:
    code: |
        print(connect.container["p"].records)
$offEmbeddedCode
```

The same can be achieved if the data has no index column names:

```
j,,j1,j1,j1,j2,j2,j2
k,,k1,k2,k3,k1,k2,k3
h1,i1,1,2,,4,5,6
h1,i2,,,3,4,5,
```

If the first line of data after the multi-row header has no data in the `valueColumns`, the CSVReader will interpret this line as index column names.

indexColumns = *list or string* (optional)

Specify columns to use as the row labels. The columns can either be given as column positions or column names. Column positions can be represented as an integer, a list of integers or a string. For example: `indexColumns: 1`, `indexColumns: [1, 2, 3, 4, 6]` or `indexColumns: "1:4, 6"`. The symbolic constant `lastCol` can be used with the string representation: `"2:lastCol"`. If no `header` or `names` is provided, `lastCol` will be determined by the first line of data. Column names can be represented as a list of strings. For example: `indexColumns: ["i1","i2"]`. Note that `indexColumns` and `valueColumns`/`textColumns` and must either be given as positions or names not both. Further note that `indexColumns` as column names are not supported together with a multi-row header.

By default the `pandas.read_csv` method interprets the following indices as `NaN`: "", "#N/A", "#N/A N/A", "#NA", "-1.#IND", "-1.#QNAN", "-NaN", "-nan", "1.#IND", "1.#QNAN", "<NA>", "N/A", "NA", "NULL", "NaN", "n/a", "nan", "null". The default can be changed by specifying `pandas.read_csv` arguments `keep_default_na` and `na_value` via readCSVArguments. Rows with indices that are interpreted as `NaN` will be dropped automatically. The indexSubstitutions option allows to remap `NaN` entries in the index columns.

indexSubstitutions = *dictionary* (optional)

Dictionary used for substitutions in the index columns. Each key in `indexSubstitutions` is replaced by its corresponding value. This option allows arbitrary replacements in the index columns of the `DataFrame` including stacked column names. Consider the following CSV file:

```
i1,j1,2.5
i1,,1.7
i2,j1,1.8
i2,,1.4
```

Reading this data into a 2-dimensional parameter results in a parameter with `NaN` entries dropped:

```
            j1

i1       2.500
i2       1.800
```

By specifying `indexSubstitutions: { .nan: j2 }` we can substitute `NaN` entries by `j2`:

```
            j1          j2

i1       2.500       1.700
i2       1.800       1.400
```

name = *string* (required)

Specify a symbol name for the Connect database. Note that each symbol in the Connect database must have a unique name.

names = *list* (optional)

List of column names to use. If the file contains a header row, then you should explicitly pass `header=True` to override the column names. Duplicates in this list are not allowed.

quoting = *integer* (default=0)

Control field quoting behavior. Use QUOTE_MINIMAL (0), QUOTE_ALL (1), QUOTE_NONNUMERIC (2) or QUOTE_NONE (3). QUOTE_NONNUMERIC (2) instructs the reader to convert all non-quoted fields to type float. QUOTE_NONE (3) instructs reader to perform no special processing of quote characters.

readCSVArguments = *dictionary* (optional)

Dictionary containing keyword arguments for the pandas.read_csv method. Not all arguments of that method are exposed through the YAML interface of the CSVReader agent. By specifying `readCSVArguments`, it is possible to pass arguments directly to the `pandas.read_csv` method that is used by the CSVReader agent. For example, `readCSVArguments: {keep_default_na: False, skip_blank_lines: False}`.

skipRows = *list or integer* (optional)

Specify the rows to skip (list) or the number of rows to skip (integer). For example: `skipRows: [1, 3]` or `skipRows: 5`.

stack = *boolean* (optional)

Stacks the column names to index. Defaults to `True` if there is more than one value/text column, otherwise `False`. Note that missing column names are filled with `Unnamed: n` (where n is the nth column (zero based) in the DataFrame).

textColumns = *list or string* (optional)

Specify columns to get the set element text from. The columns can be given as column positions or column names. Column positions can be represented as a integer, a list of integers or a string. For example: `textColumns: 1`, `textColumns: [1, 2, 3, 4, 6]` or `textColumns: "1:4, 6"`. The symbolic constant `lastCol` can be used with the string representation: `"2:lastCol"`. If no `header` or `names` is provided, `lastCol` will be determined by the first line of data. Column names can be represented as a list of strings. For example: `textColumns: ["i1","i2"]`. Note that `textColumns` and `indexColumns` must either be given as positions or names not both.

By default the `pandas.read_csv` method interprets the following text as `NaN`: "", "#N/A", "#N/A N/A", "#NA", "-1.#IND", "-1.#QNAN", "-NaN", "-nan", "1.#IND", "1.#QNAN", "<NA>", "N/A", "NA", "NULL", "NaN", "n/a", "nan", "null". The default can be changed by specifying `pandas.read_csv` arguments `keep_default_na` and `na_value` via readCSVArguments. Rows with texts that are interpreted as `NaN` will be dropped automatically. The textSubstitutions option allows to remap `NaN` entries in the text columns.

textSubstitutions = *dictionary* (optional)

Dictionary used for substitutions in the text columns. Each key in `textSubstitutions` is replaced by its corresponding value. While it is possible to make arbitrary replacements this is especially useful for controlling sparse/dense reading. The default reading behavior is sparse since rows with text that is interpreted as `NaN` are dropped automatically. Consider the following CSV file:

```
i1,text1
i2,
i3,text3
```

thousandsSeparator = *string* (optional)

Specify a thousands separator.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace > 1` some scalar debugging output will be written to the log. For `trace > 2` the intermediate data frames will be written abbreviated to the log. For `trace > 3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

valueColumns = *list or string* (optional)

Specify columns to get the values from. The columns can be given as column positions or column names. Column positions can be represented as a integer, a list of integers or a string. For example: `valueColumns: 1`, `valueColumns: [1, 2, 3, 4, 6]` or `valueColumns: "1:4, 6"`. The symbolic constant `lastCol` can be used with the string representation: `"2:lastCol"`. If no `header` or `names` is provided, `lastCol` will be determined by the first line of data. Column names can be represented as a list of strings. For example: `valueColumns: ["i1","i2"]`. Note that `valueColumns` and `indexColumns` must either be given as positions or names not both.

By default the `pandas.read_csv` method interprets the following values as `NaN`: "", "#N/A", "#N/A N/A", "#NA", "-1.#IND", "-1.#QNAN", "-NaN", "-nan", "1.#IND", "1.#QNAN", "<NA>", "N/A", "NA", "NULL", "NaN", "n/a", "nan", "null". The default can be changed by specifying `pandas.read_csv` arguments `keep_default_na` and `na_value` via readCSVArguments. Rows with values that are interpreted as `NaN` will be dropped automatically. Changing the default of values that are interpreted as `NaN` is useful if, e.g., "NA" values should not be dropped but interpreted as GAMS special value `NA`. Moreover, the valueSubstitutions option allows to remap `NaN` entries in the value columns.

valueSubstitutions = *dictionary* (optional)

Dictionary used for substitutions in the value columns. Each key in `valueSubstitutions` is replaced by its corresponding value. While it is possible to make arbitrary replacements this is especially useful for controlling sparse/dense reading. All `NaN` entries are removed automatically by default which results in a sparse reading behavior. Consider the following CSV file:

```
i1,j1,
i1,j2,1.7
i2,j1,
i2,j2,1.4
```

Reading this data into a 2-dimensional parameter results in a sparse parameter with all `NaN` entries removed:

```
            j2

i1       1.700
i2       1.400
```

By specifying `valueSubstitutions: { .nan: EPS }` we get a dense representation where all `NaN` entries are replaced by GAMS special value `EPS`:

```
            j1          j2

i1         EPS       1.700
i2         EPS       1.400
```

Beside `EPS` there are the following other GAMS special values that can be used by specifying their string representation: `INF`, `-INF`, `EPS`, `NA`, and `UNDEF`. See the GAMS Transfer documentation for more information.

Reading this data into a 1-dimensional set results in a sparse set in which all `NaN` entries (those that do not have any set element text) are removed:

```
'i1' 'text 1',
'i3' 'text 3'
```

By specifying `textSubstitutions: { .nan: '' }` we get a dense representation:

```
'i1' 'text 1',
'i2',
'i3' 'text 3'
```

It is also possible to use `textSubstitutions` in order to interpret the set element text. Let's assume we have the following CSV file:

```
,j1,j2,j3
i1,Y,Y,Y
i2,Y,Y,N
i3,0,Y,Y
```

Reading this data into a 2-dimensional set results in a dense set:

```
'i1'.'j1' Y,
'i1'.'j2' Y,
'i1'.'j3' Y,
'i2'.'j1' Y,
'i2'.'j2' Y,
'i2'.'j3' N,
'i3'.'j1' 0,
'i3'.'j2' Y,
'i3'.'j3' Y
```

By specifying `textSubstitutions: { 'N': .nan, '0': .nan }` we replace all occurrences of `N` and `0` by `NaN` which gets dropped automatically:

```
'i1'.'j1' Y,
'i1'.'j2' Y,
'i1'.'j3' Y,
'i2'.'j1' Y,
'i2'.'j2' Y,
'i3'.'j2' Y,
'i3'.'j3' Y
```

4.46.5.3 CSVWriter

The CSVWriter allows writing a symbol (set or parameter) in the Connect database to a specified CSV file. Variables and equations need to be turned into parameters with the Projection agent before they can be written. See Simple Connect Example for CSV for a simple example that uses the CSVWriter.

Option	Default	Description
decimalSeparator	`.` (period)	Specify a decimal separator.
file	None	Specify a CSV file path.
fieldSeparator	`,` (comma)	Specify a field separator.
header	`True`	Indicate if the header will be written.
name	None	Specify the name of the symbol in the Connect database.
quoting	`0`	Control field quoting behavior.
setHeader	None	Specify a string that will be used as the header.
skipElementText	`False`	Indicate if the set element text will be skipped.
toCSVArguments	None	Dictionary containing keyword arguments for the `pandas.to_csv` method.
trace	inherited	Specify the trace level for debugging output.
unstack	`False`	Specify the dimensions to be unstacked to the header row(s).

Detailed description of the options:

decimalSeparator = *string* (default=`.`)

Specify a decimal separator. [`.` (period), `,` (comma)]

file = *string* (required)

Specify a CSV file path.

fieldSeparator = *string* (default=`,`)

Specify a field separator. [`,` (comma), `;` (SemiColon), `\t` (Tab)]

header = *boolean* (default=`True`)

Indicate if the header will be written.

name = *string* (required)

Specify the name of the symbol in the Connect database.

quoting = *integer* (default=`0`)

Control field quoting behavior. Use QUOTE_MINIMAL (`0`), QUOTE_ALL (`1`), QUOTE_NONNUMERIC (`2`) or QUOTE_NONE (`3`). QUOTE_MINIMAL (`0`) instructs the writer to only quote those fields which contain special characters such as `fieldSeparator`. QUOTE_ALL (`1`) instructs the writer to quote all fields. QUOTE_NONNUMERIC (`2`) instructs the writer to quote all non-numeric fields. QUOTE_NONE (`3`) instructs the writer to never quote fields.

setHeader = *string* (optional)

Specify a string that will be used as the header. If an empty header is desired, the string can be empty.

skipElementText = *boolean* (default=`False`)

Indicate if the set element text will be skipped. If `False`, the set element text will be written in the last column of the CSV file.

toCSVArguments = *dictionary* (optional)

Dictionary containing keyword arguments for the `pandas.to_csv` method. Not all arguments of that method are exposed through the YAML interface of the CSVWriter agent. By specifying `toCSVArguments`, it is possible to pass arguments directly to the `pandas.to_csv` method that is used by the CSVWriter agent. For example, `toExcelArguments: {index_label: ["index1", "index2", "index3"]}`.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace > 1` some scalar debugging output will be written to the log. For `trace > 2` the intermediate data frames will be written abbreviated to the log. For `trace > 3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

unstack = *boolean, list* (default=`False`)

Specify the dimensions to be unstacked to the header row(s). If `False` (default) no dimension will be unstacked to the header row. If `True` the last dimension will be unstacked to the header row. If multiple dimensions should be unstacked to header rows, a list of integers providing the dimension numbers to unstack can be specified.

4.46.5.4 DomainWriter

The DomainWriter agent allows to rewrite domain information for existing Connect symbols and helps dealing with domain violations.

Option	Scope	Default	Description
dropDomainViolations	symbols	`False`	Indicate how to deal with domain violations.
name	symbols	None	Specify the name of the symbol in the Connect database.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.
writeAll	root	`auto`	Indicate if all symbols in the Connect database will be treated with the root `dropDomainViolations` setting.

Detailed description of the options:

dropDomainViolations = *boolean (root)/boolean or string (symbols)* (default=`False`)

The Connect symbols might have some domain violations. This agent allows to drop these domain violations so a write to GAMS or GDX works properly. Setting the root option `dropDomainViolations: True` together with `writeAll: True` will drop domain violations from all symbols in the Connect database. The symbols option also allows to drop domain violations. In the `symbols` section the `dropDomainViolations` attribute can be of type boolean (`True` or `False`) or of type string (`before` and `after`). If the attribute has not been set for the symbol, the attribute is inherited from the root attribute. The value `before` means that domain violations are dropped *before* a new domain is applied, see attribute name. The value `after` means that domain violations are dropped *after* a new domain is applied. The value `True` means that domain violations are dropped before and after a new domain is applied. `False` means to not drop domain violations.

name = *string* (required)

Specify a symbol name with index space for the Connect database. `name` requires the format `symName(i1,i2,...,iN)`. The list of indices needs to coincide with the names of the actual GAMS domain sets for a *regular* domain. A *relaxed* domain is set if the index is quoted. For example `name: x(i,'j')` means that for the first index a regular domain with domain set `i` is established, while for the second index the universal domain $*$ is used and a *relaxed domain name* `j` is set.

symbols = *list* (optional)

A list containing symbol specific options. Allows to concatenate a subset of symbols.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` $>$ `1` some scalar debugging output will be written to the log. For `trace` $>$ `2` the intermediate data frames will be written abbreviated to the log. For `trace` $>$ `3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

writeAll = *boolean or* ***auto*** (default=`auto`)

Indicate if all symbols in the Connect database will be treated according to root attribute dropDomainViolations. If `True`, treat all symbols according to `dropDomainViolations` and ignore symbols. The default `auto` becomes `True` if there are no symbol options specified, otherwise `False`.

4.46.5.5 ExcelReader

The ExcelReader agent allows to read symbols (sets and parameters) from an Excel file into the Connect database. See Simple Connect Example for Excel for a simple example that uses the ExcelReader.

Note

The ExcelReader supports `.xlsx` and `.xlsm` files.

Option	Scope	Default	Description
autoMerge	root/symbols	`False`	Indicate if empty cells in the labels should be merged with previous cells.
columnDimension	root/symbols	`1`	Column dimension of the symbol.
file	root/symbols	None	Specify an Excel file path.
ignoreColumns	symbols	None	Columns to be ignored when reading.
ignoreRows	symbols	None	Rows to be ignored when reading.
ignoreText	root/symbols	`auto`	Indicate if the set element text should be ignored.
index	root	None	Specify the Excel range for reading symbols and options directly from the spreadsheet.
indexSubstitutions	root/symbols	None	Dictionary used for substitutions in the row and column index.
mergedCells	root	`False`	Control the handling of empty cells in the labels and the values that are part of a merged Excel range.
name	symbols	None	Specify the name of the symbol in the Connect database.
range	symbols	`symName!A1`	Specify the Excel range of a symbol.
rowDimension	root/symbols	`1`	Row dimension of the symbol.
skipEmpty	root/symbols	`1`	Number of empty rows or columns to skip before the next empty row or column indicates the end of the block for reading with open ranges.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.
type	root/symbols	`par`	Control the symbol type.
valueSubstitutions	root/symbols	None	Dictionary used for mapping in the values.

Detailed description of the options: **autoMerge** = *boolean* (default=`False`)

Indicate if empty cells in the labels should be merged with previous cells. If `True`, each empty cell in the labels will be filled with the value of a previous cell that is not empty. This will resolve merged cells in the labels and will also fill empty cells in the labels that are not part of a merged cell. If the user wants to keep empty cells in the labels that are not part of a merged cell and still resolve merged cells, `autoMerge` needs to be set to `False` and mergedCells needs to be set to `True`.

columnDimension = *integer* (default=`1`)

The number of rows in the data range that will be used to define the labels for columns. The first `columnDimension` rows of the data range will be used for labels.

file = *string* (required)

Specify an Excel file path.

ignoreColumns = *list, integer or string* (optional)

Columns to be ignored when reading. Specify a list of column numbers (integers) or letters (strings) to ignore multiple columns. Specify a column number (integer) ot letter (string) to ignore a single column.

ignoreRows = *list or integer* (optional)

Specify rows to be ignored when reading. Specify a list of row numbers (integers) to ignore multiple rows or specify a row number (integer) to ignore a single row.

ignoreText = *boolean or string* (default=`auto`)

Indicate if the set element text should be ignored. Can be set to `True`/`False` or `auto`. With the default `auto` the set element text will be ignored based on the specification of range and columnDimension/rowDimension. If either `columnDimension` or `rowDimension` is set to `0` and an open range is specified, the set element text will be ignored. If either `columnDimension` or `rowDimension` is set to `0` and a full range is specified, the set element text will be ignored if it is not included in the range specification. In all other cases the set element text will not be ignored.

index = *string* (optional)

Specify the Excel range for reading symbols and options directly from the spreadsheet.

indexSubstitutions = *dictionary* (optional)

Dictionary used for substitutions in the row and column index. Each key in `indexSubstitutions` is replaced by its corresponding value. This option allows arbitrary replacements in the index. Consider the following Excel spreadsheet:

Reading this data into a 2-dimensional parameter results in a parameter with `NaN` entries dropped:

```
            j1

i1       1.000
i2       3.000
```

By specifying `indexSubstitutions: { .nan: j2 }` we can substitute `NaN` entries by `j2`:

```
            j1          j2

i1       1.000       2.000
i2       3.000       4.000
```

mergedCells = *boolean* (default=`False`)

Control the handling of empty cells that are part of a merged Excel range. The option applies to merged cells in the labels and the values, i.e., the numerical values in case of a GAMS parameter and the set element text in case of a GAMS set. If `False`, the cells are left empty. If `True`, the merged label/value is used in all cells. Note that setting this option to `True` has an impact on performance since the Excel file has to be opened in a non-read-only mode that results in non-constant memory consumption (no lazy loading). From the performance perspective, autoMerge should be preferred over `mergedCells` if applicable.

name = *string* (required)

Specify a symbol name for the Connect database. Note that each symbol in the Connect database must have a unique name.

range = *string* (default=`symName!A1`)

Specify the Excel range of a symbol using the format `sheetName!cellRange`. `cellRange` can be either a single cell also known as an *open range* (north-west corner like `B2`) or a *full range* (north-west and south-east corner like `B2:D4`). Per default the ExcelReader uses the range `symName!A1`, where `symName` is the name of the symbol that is read. If only `sheetName!` is specified, the ExcelReader will use an open range starting at cell `A1`. The ExcelReader also allows for named ranges - a named range includes a sheet name and a cell range. Before interpreting the provided `range` attribute, the string will be used to search for a pre-defined Excel range with that name.

rowDimension = *integer* (default=`1`)

The number of columns in the data range that will be used to define the labels for the rows. The first `rowDimension` columns of the data range will be used for the labels.

skipEmpty = *integer* (deault=`1`)

Number of empty rows or columns to skip before the next empty row or column indicates the end of the block for reading with open ranges. If a full range is specified `skipEmpty` will be ignored.

symbols = *list* (optional)

A list containing symbol specific options.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace > 1` some scalar debugging output will be written to the log. For `trace > 2` the intermediate data frames will be written abbreviated to the log. For `trace > 3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

type = *string* (default=`par`)

Control the symbol type. Supported symbol types are `par` for parameters and `set` for sets.

valueSubstitutions = *dictionary* (optional)

Dictionary used for mapping in the values. Each key in `valueSubstitutions` is replaced by its corresponding value. The replacement is only performed on the values which is the numerical value in case of a GAMS parameter and the set element text in case of a GAMS set. While it is possible to make arbitrary replacements this is especially useful for controlling sparse/dense reading. All `NaN` entries are removed automatically by default which results in a sparse reading behavior. Let's assume we have the following spreadsheet:

Reading this data into a 2-dimensional parameter results in a sparse parameter in which all `NaN` entries are removed:

```
'i1'.'j1' 2.5,
'i1'.'j2' 1.7,
'i2'.'j2' 1.8,
'i2'.'j3' 1.4
```

By specifying `valueSubstitutions: { .nan: EPS }` we get a dense representation in which all `NaN` entries are replaced by GAMS special value `EPS`:

```
'i1'.'j1' 2.5,
'i1'.'j2' 1.7,
'i1'.'j3' Eps,
'i2'.'j1' Eps,
'i2'.'j2' 1.8,
'i2'.'j3' 1.4
```

Beside `EPS` there are the following other GAMS special values that can be used by specifying their string representation: `INF`, `-INF`, `EPS`, `NA`, and `UNDEF`. See the GAMS Transfer documentation for more information.

Let's assume we have data representing a GAMS set:

Reading this data into a 1-dimensional set results in a sparse set in which all `NaN` entries (those that do not have any set element text) are removed:

```
'i1' 'text 1',
'i3' 'text 3'
```

By specifying `valueSubstitutions: { .nan: '' }` we get a dense representation:

```
'i1' 'text 1'
'i2' ''
'i3' 'text 3'
```

It is also possible to use `valueSubstitutions` in order to interpret the set element text. Let's assume we have the following Excel data:

Reading this data into a 2-dimensional set results in a dense set:

```
'i1'.'j1' No,
'i1'.'j2' Y,
'i1'.'j3' Y,
'i2'.'j1' Y,
'i2'.'j2' Y,
'i2'.'j3' Y,
'i3'.'j1' Y,
'i3'.'j2' Y,
'i3'.'j3' N
```

By specifying `valueSubstitutions: { 'N': .nan, 'No': .nan }` we replace all occurrences of `N` and `No` by `NaN` which gets dropped automatically. Note that `No` has to be quotes in order to not be interpreted as `False` by the YAML parser:

```
'i1'.'j2' Y,
'i1'.'j3' Y,
'i2'.'j1' Y,
'i2'.'j2' Y,
'i2'.'j3' Y,
'i3'.'j1' Y,
'i3'.'j2' Y
```

4.46.5.6 ExcelWriter

The ExcelWriter agent allows to write symbols (sets and parameters) from the Connect database to an Excel file. Variables and equations need to be turned into parameters with the Projection agent before they can be written. If the Excel file exists, the ExcelWriter appends to the existing file. See Simple Connect Example for Excel for a simple example that uses the ExcelWriter.

Note

The ExcelWriter only supports `.xlsx` files.

Attention

Please be aware of the following limitation when appending to an Excel file with formulas using the ExcelWriter: Whereas Excel stores formulas and the corresponding values, the ExcelReader and the ExcelWriter read/store either formulas or values, not both. As a consequence, when appending to an Excel file with formulas, all cells with formulas within the Excel file will not have values anymore and a subsequent read by the ExcelReader results into `NaN` for these cells. To avoid this, write to a separate output Excel file. On Windows one can merge the input Excel file with the output Excel file at the end using the tool win32.ExcelMerge (see Connect Example for Excel (executeTool win32.ExcelMerge)). An alternative approach when appending to an Excel file with formulas is to open and save the Excel file before reading it to let Excel evaluate formulas and restore the corresponding values.

Option	Scope	Default	Description
clearSheet	root/symbols	`False`	Indicate if a sheet should be cleared before writing if it exists.
columnDimension	root/symbols	`auto`	Column dimension of the symbol.
file	root	None	Specify an Excel file path.
mergedCells	root/symbols	`False`	Write merged cells.
name	symbols	None	Specify the name of the symbol in the Connect database.
range	symbols	`symName!A1`	Specify the Excel range of a symbol.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.
valueSubstitutions	root/symbols	None	Dictionary used for mapping in the `value` column of the `DataFrame`.
writeAll	root	`auto`	Indicate if all set and parameter type symbols in the Connect database will be written to the specified Excel file.

Detailed description of the options:

clearSheet = *boolean* (default=`False`)

Indicate if a sheet should be cleared before writing if it exists. The default is `False` where a sheet will not be cleared if it exists, instead the ExcelWriter overwrites, i.e., writes content to the sheet without removing the existing content.

columnDimension = *integer* (default=`auto`)

The last `columnDimension` index positions of the symbol that will be written to the rows that define the labels of the columns. The first `dim-columnDimension` index positions will be written to the columns that define the labels for the rows. With the default `auto`, `columnDimension` will be set to `1` if `dim`>`0` and otherwise to `0`.

file = *string* (required)

Specify an Excel file path.

mergedCells = *boolean* (default=`False`)

Write merged cells. Please be aware that overwriting already existing merged cells in a sheet may cause problems, in that case consider setting clearSheet to `True`.

name = *string* (required)

Specify a symbol name for the Connect database.

range = *string* (default=`symName!A1`)

Specify the Excel range of a symbol using the format `sheetName!cellRange`. `cellRange` can be either a single cell (north-west corner like `B2`) or a full range (north-west and south-east corner like `B2:D4`). Per default the ExcelWriter uses the range `symName!A1`, where `symName` is the name of the symbol that is written. If only `sheetName!` is specified, the ExcelWriter will use an open range starting at cell `A1`. The ExcelWriter also allows for named ranges - a named range includes a sheet name and a cell range. Before interpreting the provided `range` attribute, the string will be used to search for a pre-defined Excel range with that name.

symbols = *list* (required)

A list containing symbol specific options.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` $>$ `1` some scalar debugging output will be written to the log. For `trace` $>$ `2` the intermediate data frames will be written abbreviated to the log. For `trace` $>$ `3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

valueSubstitutions = *dictionary* (optional)

Dictionary used for mapping in the `value` column of the `DataFrame`. Each key in `valueSubstitutions` is replaced by its corresponding value. The replacement is only performed on the `value` column of the `DataFrame` which is the numerical value in case of a GAMS parameter and the set element text in case of a GAMS set. Note that for parameters the ExcelWriter automatically converts numerical GAMS special values to their string representation, i.e., `INF`, `-INF`, `EPS`, `NA`, and `UNDEF`. See the GAMS Transfer documentation for more information on GAMS special values. If the GAMS special values should be replaced by custom values, use the string representation (upper case) in the dictionary. For example, specify {`'EPS': 0`} to replace GAMS special value `EPS` by zero.

writeAll = *boolean or* ***`auto`*** (default=`auto`)

Indicate if all set and parameter type symbols in the Connect database will be written to the specified Excel file. The default `auto` becomes `True` if there are no symbol options specified, otherwise `False`. If `True`, the ExcelWriter writes each symbol into a separate sheet using the range `symName!A1`.

4.46.5.7 Filter

The Filter agent allows to reduce symbol data by applying filters on labels and numerical values. Here is a complete example that uses the Filter agent:

```
Set i / seattle, san-diego /
    j / new-york, chicago, topeka /;

Parameter d(i,j) /
seattle.new-york   2.5
seattle.chicago    1.7
seattle.topeka     1.8
san-diego.new-york 2.5
san-diego.chicago  1.8
san-diego.topeka   1.4
/;

$onEmbeddedCode Connect:
- GAMSReader:
    symbols:
      - name: d
- Filter:
    name: d
    newName: d_new
    labelFilters:
      - column: 1
        keep: ['seattle']
      - column: 2
        reject: ['topeka']
    valueFilters:
      - column: value
        rule: x<2.5
- GDXWriter:
    file: report.gdx
    symbols:
      - name: d_new
$offEmbeddedCode
```

The records of the parameter `d` are filtered and stored in a new parameter called `d_new`. Two label filters remove all labels except `seattle` from the first dimension and remove the label `topeka` from the second one. The remaining records are filtered by value where only values less than 2.5 are kept in the data. The resulting parameter `d_new` which is exported into `report.gdx` has only one record (`seattle.chicago 1.7`) left.

The following options are available for the Filter agent:

Option	Scope	Default	Description
column	labelFilters/valueFilters	None	Specify the column to which a filter is applied.
eps	valueFilters	`True`	Used to keep or reject special value `EPS`.
infinity	valueFilters	`True`	Used to keep or reject special value `+INF`.
keep	labelFilters	None	Specify a list of labels to keep.
labelFilters	root	None	Specify filters for index columns of a symbol.
na	valueFilters	`True`	Used to keep or reject special value `NA`.

Option	Scope	Default	Description
name	root	None	Specify a symbol name for the Connect database.
negativeInfinity	valueFilters	`True`	Used to keep or reject special value `-INF`.
newName	root	None	Specify a new name for the symbol in the Connect database.
regex	labelFilters	None	Specify a regular expression to be used for filtering labels.
reject	labelFilters	None	Specify a list of labels to reject.
rule	valueFilters	None	Specify a boolean expression to be used for filtering on numerical columns.
ruleIdentifier	valueFilters	`x`	The identifier used for the value filter rule.
trace	root	inherited	Specify the trace level for debugging output.
undf	valueFilters	`True`	Used to keep or reject special value `UNDF`.
valueFilters	root	None	Specify filters for numerical columns of a symbol.

Detailed description of the options:

column = *integer or string* (optional)

Used to specify the column on which a label filter or a value filter is applied. For label filters the index position can be specified using an integer. For value filters, the following strings are allowed depending on the symbol type:

- Set: not allowed
- Parameter: `value`, `all`
- Variable and Equation: `level`, `marginal`, `upper`, `lower`, `scale`, `all`

Specifying the string `all` will apply the filter on all columns of the current filter (all index columns for labelFilters and all numerical columns for valueFilters).

eps = *boolean* (default=`True`)

Used to keep (`True`) or reject (`False`) special value `EPS`.

infinity = *boolean* (default=`True`)

Used to keep (`True`) or reject (`False`) special value `+INF`.

keep = *list* (optional)

A list of labels to be kept when applying the label filter. For each label filter it is only allowed to specify either keep, reject, or regex at a time.

labelFilters = *list* (optional)

A list containing label filters.

na = *boolean* (default=`True`)

Used to keep (`True`) or reject (`False`) special value `NA`.

name = *string* (required)

Specify the name of the symbol from the Connect database on whose data the filters will be applied.

negativeInfinity = *boolean* (default=`True`)

Used to keep (`True`) or reject (`False`) special value `-INF`.

newName = *string* (required)

Specify a new name for the symbol in the Connect database which will get the data after all filters have been applied. Each symbol in the Connect database must have a unique name.

regex = *string* (optional)

A string containing a regular expression that needs to match in order to keep the corresponding label. Uses a full match paradigm which means that the whole label needs to match the specified regular expression. For each label filter it is only allowed to specify either keep, reject, or regex at a time.

reject = *string* (optional)

A list of labels to be rejected when applying the label filter. For each label filter it is only allowed to specify either keep, reject, or regex at a time.

rule = *string* (optional)

Used to specify a boolean expression for a value filter. Each numerical value of the specified column is tested and the corresponding record is only kept if the expression evaluates to true. The string needs to contain Python syntax that is valid for `pandas.Series`. Comparison operators like >, `>=`, <, `<=`, `==`, or `!=` can be used in combination with boolean operators like `&` or |, but not `and` or `or`. Note that using `&` or | requires the operands to be enclosed in round brackets in order to form a valid expression. As an example, the expression `((x<=10) & (x>=0)) | (x>20)` would keep only those values that are between `0` and `10` (included) or greater than `20`.

ruleIdentifier = *string* (default=`x`)

Specifies the identifier that is used in the rule of a value filter.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` > `1` some scalar debugging output will be written to the log. For `trace` > `2` the intermediate data frames will be written abbreviated to the log. For `trace` > `3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

undf = *boolean* (default=`True`)

Used to keep (`True`) or reject (`False`) special value `UNDF`.

valueFilters = *list* (optional)

A list containing value filters.

4.46.5.8 GAMSReader

The GAMSReader allows reading symbols from the GAMS database into the Connect database. Without GAMS context (e.g. when running the `gamsconnect` script from the command line) this agent is not available and its execution will result in an exception.

Option	Scope	Default	Description
name	symbols	None	Specify the name of the symbol in the GAMS database.
newName	symbols	None	Specify a new name for the symbol in the Connect database.
readAll	root	`auto`	Indicate if all symbols in the GAMS database will be read into the Connect database.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.

Detailed description of the options:

name = *string* (required)

Specify the name of the symbol in the GAMS database.

newName = *string* (optional)

Specify a new name for the symbol in the Connect database. Each symbol in the Connect database must have a unique name.

readAll = *boolean or* ***`auto`*** (default=`auto`)

Indicate if all symbols in the GAMS database will be read into the Connect database. If `True`, read all symbols into the Connect database and ignore symbols. The default `auto` becomes `True` if there are no symbol options specified, otherwise `False`.

symbols = *list* (optional)

A list containing symbol specific options. Allows to read a subset of symbols.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` > `1` some scalar debugging output will be written to the log. For `trace` > `2` the intermediate data frames will be written abbreviated to the log. For `trace` > `3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

4.46.5.9 GAMSWriter

The GAMSWriter allows writing symbols in the Connect database to the GAMS database. Without GAMS context (e.g. when running the `gamsconnect` script from the command line) and as part of the connectOut command line option this agent is not available and its execution will result in an exception.

Option	Scope	Default	Description
domainCheckType	root/symbols	`default`	Specify if domain checking is applied or if records that would cause a domain violation are filtered.
duplicateRecords	root/symbols	`all`	Specify how to deal with duplicate records.
mergeType	root/symbols	`default`	Specify if data in a GAMS symbol is merged or replaced.
name	symbols	None	Specify the name of the symbol in the Connect database.
newName	symbols	None	Specify a new name for the symbol in the GAMS database.
~~symbols~~	~~root~~	~~None~~	~~Specify symbol specific options.~~
trace	root	inherited	Specify the trace level for debugging output.
writeAll	root	`auto`	Indicate if all symbols in the Connect database will be written to the GAMS database.

Detailed description of the options:

domainCheckType = *string* (default=`default`)

Specify if Domain Checking is applied (`checked`) or if records that would cause a domain violation are filtered (`filtered`). If left at `default` it depends on the setting of $on/offFiltered if GAMS does a filtered load or checks the domains during compile time. During execution time `default` is the same as `filtered`.

duplicateRecords = *string* (default=`all`)

The Connect container can hold multiple records even for the same indexes. Such duplicate records are only a problem when exchanging the data with GAMS (and GDX). The attribute determines which record(s) to keep in case of duplicate records. With the default of `all` the GAMSWriter will fail in case duplicate records exist. With `first` the first record will be written to GAMS, with `last` the last record will be written to GAMS. With `none` none of the duplicate records will be written to GAMS. Note that the agent currently deals with duplicate records in a case sensitive way.

mergeType = *string* (default=`default`)

Specify if data in a GAMS symbol is merged (`merge`) or replaced (`replace`). If left at `default` it depends on the setting of $on/offMulti[R] if GAMS does a merge, replace, or triggers an error during compile time. During execution time `default` is the same as `merge`.

name = *string* (required)

Specify the name of the symbol in the Connect database.

newName = *string* (optional)

Specify a new name for the symbol in the GAMS database. Note, each symbol in the GAMS database must have a unique name.

symbols = *list* (optional)

A list containing symbol specific options. Allows to write a subset of symbols.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` $>$ `1` some scalar debugging output will be written to the log. For `trace` $>$ `2` the intermediate data frames will be written abbreviated to the log. For `trace` $>$ `3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

writeAll = *boolean or* ***`auto`*** (default=`auto`)

Indicate if all symbols in the Connect database will be written to the GAMS database. If `True`, write all symbols to the GAMS database and ignore symbols. The default `auto` becomes `True` if there are no symbol options specified, otherwise `False`.

4.46.5.10 GDXReader

The GDXReader allows reading symbols from a specified GDX file into the Connect database.

Option	Scope	Default	Description
file	root	None	Specify a GDX file path.
name	symbols	None	Specify the name of the symbol in the GDX file.
newName	symbols	None	Specify a new name for the symbol in the Connect database.
readAll	root	`auto`	Indicate if all symbols in the GDX file will be read into the Connect database.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.

Detailed description of the options:

file = *string* (required)

Specify a GDX file path.

name = *string* (required)

Specify the name of the symbol in the GDX file.

newName = *string* (optional)

Specify a new name for the symbol in the Connect database. Each symbol in the Connect database must have a unique name.

readAll = *boolean or* `auto` (default=`auto`)

Indicate if all symbols in the GDX file will be read into the Connect database. If `True`, read all symbols into the Connect database and ignore symbols. The default `auto` becomes `True` if there are no symbol options specified, otherwise `False`.

symbols = *list* (optional)

A list containing symbol specific options. Allows to read a subset of symbols.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` > `1` some scalar debugging output will be written to the log. For `trace` > `2` the intermediate data frames will be written abbreviated to the log. For `trace` > `3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

4.46.5.11 GDXWriter

The GDXWriter allows writing symbols in the Connect database to a specified GDX file.

Option	Scope	Default	Description
duplicateRecords	root/symbols	`all`	Specify how to deal with duplicate records.
file	root	None	Specify a GDX file path.
name	symbols	None	Specify the name of the symbol in the Connect database.
newName	symbols	None	Specify a new name for the symbol in the GDX file.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.
writeAll	root	`auto`	Indicate if all symbols in the Connect database will be written to the GDX file.

Detailed description of the options:

duplicateRecords = *string* (default=`all`)

The Connect container can hold multiple records even for the same indexes. Such duplicate records are only a problem when exchanging the data with GDX (and GAMS). The attribute determines which record(s) to keep in case of duplicate records. With the default of `all` the GDXWriter will fail in case duplicate records exist. With `first` the first record will be written to GDX, with `last` the last record will be written to GDX. With `none` none of the duplicate records will be written to GDX. Note that the agent currently deals with duplicate records in a case sensitive way.

file = *string* (required)

Specify a GDX file path.

name = *string* (required)

Specify the name of the symbol in the Connect database.

newName = *string* (optional)

Specify a new name for the symbol in the GDX file. Note, each symbol in the GDX file must have a unique name.

symbols = *list* (optional)

A list containing symbol specific options. Allows to write a subset of symbols.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` > `1` some scalar debugging output will be written to the log. For `trace` > `2` the intermediate data frames will be written abbreviated to the log. For `trace` > `3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

writeAll = *boolean or* ***`auto`*** (default=`auto`)

Indicate if all symbols in the Connect database will be written to the GDX file. If `True`, write all symbols to the GDX file and ignore symbols. The default `auto` becomes `True` if there are no symbol options specified, otherwise `False`.

4.46.5.12 LabelManipulator

The LabelManipulator agent allows to modify labels of symbols in the Connect database. Four different modes are available:

- case: Applies either upper, lower, or capitalize casing to labels.
- code: Replaces labels using a Python code.
- map: Uses a 1-dimensional GAMS set to perform an explicit mapping of labels.
- regex: Performs a replacement based on a regular expression.

Here is a complete example that uses the LabelManipulator agent in all four modes:

```
Set i / seattle, san-diego /
    j / new-york, chicago, topeka /
    map / chicago 'Berlin', san-diego 'Oslo' /;

Parameter d(i,j) /
seattle.new-york   2.5
seattle.chicago    1.7
seattle.topeka     1.8
san-diego.new-york 2.5
san-diego.chicago  1.8
san-diego.topeka   1.4
/;

$onEmbeddedCode Connect:
- GAMSReader:
    readAll: true
- LabelManipulator:
    map:
        setName: map
- LabelManipulator:
    case:
        rule: upper
- LabelManipulator:
    symbols:
      - name: d
    code:
        rule: x.split('-')[-1]
- LabelManipulator:
    symbols:
      - name: d
    regex:
        pattern: '[^O]$'
        replace: '\g<0>X'
- PythonCode:
    code: |
        print("Set i:\n", connect.container["i"].records)
        print("Set j:\n",connect.container["j"].records)
        print("Parameter d:\n",connect.container["d"].records)
$offEmbeddedCode
```

The first LabelManipulator applies a mapping provided by the set `map` to the labels of all symbols in the Connect database. This maps `chicago` to `Berlin` and `san-diego` to `Oslo`. The second LabelManipulator changes the labels of all symbols to upper case. The third LabelManipulator is only applied on symbol `d` and splits labels at `-` into a list and keeps the last entry. This changes `NEW-YORK` to `YORK`. The last LabelManipulator is also only applied on symbol `d` and adds an `X` to the end of all labels that do not end with an `O`. The resulting symbols look as follows:

```
Set i:
        uni element_text
0  SEATTLE
1     OSLO

Set j:
         uni element_text
0  NEW-YORK
1    BERLIN
2    TOPEKA
```

```
Parameter d:
           i         j  value
0  SEATTLEX     YORKX    2.5
1  SEATTLEX   BERLINX    1.7
2  SEATTLEX   TOPEKAX    1.8
3      OSLO     YORKX    2.5
4      OSLO   BERLINX    1.8
5      OSLO   TOPEKAX    1.4
```

The following options are available for the LabelManipulator agent:

Option	Scope	Default	Description
case	root	None	Apply specified casing to labels.
code	root	None	Replace labels using Python code.
invert	map	`False`	Used to invert the mapping direction.
map	root	None	Replace labels using a 1-dimensional GAMS set containing an explicit key-value mapping.
name	symbols	None	Specify a symbol name for the Connect database.
newName	symbols	None	Specify a new name for the symbol in the Connect database.
outputSet	regex/case/code	None	Name of the output set that contains the applied mapping.
pattern	regex	None	The regular expression that needs to match.
regex	root	None	Replace labels using a regular expression.
replace	regex	None	The rule used for replacing labels that match the given pattern.
rule	case/code	None	case: The type of casing to be applied. code: Python function that defines the mapping behavior.
ruleIdentifier	code	`x`	The identifier used for labels in the rule.
setName	map	None	The name of the GAMS set used in map mode.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.
writeAll	root	`auto`	Indicate if all symbols in the Connect database will be affected.

Detailed description of the options:

case = *dictionary* (optional)

Used for executing the LabelManipulator in case mode to apply a specified casing to labels. It is only allowed to specify either case, code, map or regex.

code = *dictionary* (optional)

Used for executing the LabelManipulator in code mode to replace labels using a Python function. The given Python function is executed with each label and its return value is used as the replacement. It is only allowed to specify either case, code, map or regex.

invert = *boolean* (default=`False`)

Used for controlling the mapping direction in map mode. If set to `False` (default), the labels that match a label of the provided 1-dimensional GAMS set are replaced by the corresponding set element text. If set to `True` the direction is inverted, meaning all labels that match a set element text of the GAMS set are replaced by the corresponding label.

map = *dictionary* (optional)

Used for executing the LabelManipulator in map mode to replace labels using a 1-dimensional GAMS set containing an explicit key-value mapping. It is only allowed to specify either case, code, map or regex.

name = *string* (optional)

Specify the name of the symbol in the GAMS database. Data of the symbol gets replaced if no newName is specified.

newName = *string* (optional)

Specify a new name for the symbol in the Connect database. The original symbol specified under name remains unchanged. Each symbol in the Connect database must have a unique name.

outputSet = *string* (optional)

Name of the output set that contains mappings that were actually applied on the symbol labels. Per default no output set is written. Providing a name for the output set indicates that an output set should be written to the Connect database. Note that each symbol in the Connect database must have a unique name. Supported by case, code and regex mode.

pattern = *string* (optional)

The regular expression that needs to match for a label to be replaced.

regex = *dictionary* (optional)

Used for executing the LabelManipulator in regex mode to replace labels using a regular expression. It is only allowed to specify either case, code, map or regex.

replace = *string* (optional)

A string that specifies the replacement for all labels for which a given pattern in regex mode matches.

rule = *string* (optional)

Can be specified in case and code mode.

case: The type of casing to be applied. Allowed values are:

- `lower`: Change all labels to lower case.
- `upper`: Change all labels to upper case.
- `capitalize`: Change all labels to a capitalized casing - first letter becomes upper case, all others become lower case.

code: A Python function that defines the mapping behavior.

ruleIdentifier = *string* (default=`x`)

Specifies the identifier that is used in the rule of the code mode.

setName = *string*

Replace labels using a 1-dimensional GAMS set containing an explicit key-value mapping. A 1-dimensional GAMS set that contains an explicit key-value mapping to replace labels. If `invert: False` (default) all labels that match a label of the GAMS set will be replaced by the corresponding set element text.

symbols = *dictionary* (optional)

A list containing symbol specific options. Allows to execute the LabelManipulator on a subset of GAMS symbols in the Connect database.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace > 1` some scalar debugging output will be written to the log. For `trace > 2` the intermediate data frames will be written abbreviated to the log. For `trace > 3` the intermediate data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

writeAll = *boolean or* ***auto*** (default=`auto`)

Indicate if the LabelManipulator is applied to all symbols of the database or only a subset of GAMS symbols in the Connect database. The default `auto` becomes `True` if there are no symbol options specified, otherwise `False`.

4.46.5.13 Options

The Options agent allows to set more general options that can affect the Connect database and other Connect agents. More specifically, the value of an option set via the Options agent can be inherited as a default value to Connect agents that utilize the considered option.

Option	Default	Description
trace	0	Specify the trace level for debugging output.

Detailed description of the options:

trace = *integer* (default=0)

Specify the trace level for debugging output. A trace level of `0` (default) means no debugging output. For `trace > 0` the Connect database will write some scalar debugging output to the log. The debugging output of Connect agents depends on their implementation of `trace`, please refer to the corresponding documentation.

4.46.5.14 Projection

The Projection agent allows index reordering and projection onto a reduced index space of a GAMS symbol. For variables and equations a suffix (`.l`, `.m`, `.lo`, `.up`, or `.scale`) can be extracted and is written to a parameter. Otherwise, the type of the source symbol determines the type of the new symbol, unless `asSet` is set to `True`. Variables and equations can be turned into parameters with an extra index containing the labels `level`, `marginal`, `lower`, `upper`, and `scale` if the attribute `asParameter` is set to `True`. Moreover, if `name` is a *list* of scalar symbols of the same type (parameters, variables, or equations), they can be stored in a one-dimensional symbols of the same type with the index label being the *name* of the scalar symbol.

Option	Default	Description
aggregationMethod	`first`	Specify the aggregation method for the projection.
asParameter	`False`	Indicate that variable or equation symbols will be turned into a parameter with an extra index that contains the suffix.
asSet	`False`	Indicate that the new symbol is a set independent of the type of the source symbol.
name	None	Specify a symbol name with index space and potentially suffix for the Connect database.
newName	None	Specify a new name with index space for the symbol in the Connect database.
text	None	Element text for resulting sets.
trace	inherited	Specify the trace level for debugging output.

Detailed description of the options:

aggregationMethod = *string* (default=`first`)

Specify the method to aggregate when at least one index position is projected out. The default is `first`, meaning that the first record will be stored in the new symbol. For sets, variables, and equations (without a suffix specified) only `first` and `last` are meaningful. For parameters, variables, and equations with suffix many other aggregation methods are available and meaningful: `max`, `mean`, `median`, `min`, `prod`, `sem` (unbiased standard error of the mean), `sum`, `std` (standard deviation), `nunique` (number of distinct elements), `first`, `last`. The projection agent is based on pandas DataFrames and more detailed explanations of the aggregation method can be found at the `pandas website`.

asParameter = *boolean* (default=`False`)

If the symbol specified by `name` is a variable or equation and `asParameter` is set to `True`, the new symbol (after potential `aggregationMethod` is applied) will be a parameter with an additional index (at the end of the index list) that contains the suffix labels `level`, `marginal`, `lower`, `upper`, and `scale` and the corresponding suffix value.

asSet = *boolean* (default=`False`)

Usually the type of the source symbol and the use of a suffix with variables and equations determine the type of the target symbol. With `asSet` set to `True` the target symbol will be a set.

name = *string* or *list of strings*(required)

One either specifies a single symbol name with index space and potentially suffix for the Connect database or a list of symbol names of scalar symbols. In the prior case, `name` requires the format `symName[.suffix](i1,i2,...,iN)`. The suffix is only allowed on variable and equation symbols and need to be either `l`, `m`, `lo`, `up`, or `scale`. The list of indices does not need to coincide with the names of the actual GAMS domain sets. This index list together with the index list specified for `newName` is solely intended to establish the index order in the symbol specified by `newName`. In the latter case (a list of symbol names) the symbols need to be scalar symbols of the same type (parameter, variable, or equation) and a new one-dimensional symbol (of the same type) is created (using `newName`) that holds the symbol names as labels.

newName = *string* (required)

Specify a new name with index space for the projected or reordered symbol in the Connect database. Note that each symbol in the Connect database must have a unique name. `newName` is given as `symName(i1,i2,...,iN)`. The list of indices does not need to coincide with the names of the actual GAMS domain sets. This index list together with the index list specified for `name` is solely intended to establish the index order. Hence, the names in the index list need to be unique and only names that are part of the index list specified for `name` can be used. For example: `name: p(I,j,k)` and `newName: q(k,i)`.

text = *string* (default=None)

Control the handling of element text if the resulting symbol is a set. If set to `""`, the text will be dropped. When left at default (`None`) and the projected symbol is a set, the element text of the original set will be used. For other symbols types, the text will be dropped. If `text` is a string, this string will be assigned to all elements. The string can contain *place holders* {`i1`} that will be replaced with the content of the matching index position. For example, `text: "{i2} - {i1}: {element_text}"`, where {`i1`} and {`i2`} should be index space names in the symbol name with index space (attribute `name`). {`element_text`} refers to original element text (set) or string representation of a numerical value (parameter or variable/equation with a given suffix) of the source symbol.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` > `1` some scalar debugging output will be written to the log. For `trace` > `2` the intermediate arrays and data frames will be written abbreviated to the log. For `trace` > `3` the intermediate arrays and data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

4.46.5.15 PythonCode

The PythonCode agent allows to execute arbitrary Python code. From within the code, it is possible to access the GAMS database via `gams.db` (if the PythonCode agent is running in a GAMS context) and the Connect database via `connect.container`. The GAMS database is an instance of **GamsDatabase**, whereas the Connect database is a GAMS Transfer `Container` object. Furthermore there is a predefined `instructions` list that can be filled with tasks that are automatically executed.

Option	Default	Description
code	None	Python code to be executed.

Detailed description of the options:

code = *string* (required)

Python code to be executed. The YAML syntax offers the pipe character (|) for specifying multi-line strings:

```
- PythonCode:
    code: |
      print("Print from Python")
      # insert more Python code here
```

It is possible to generate instructions by appending tasks to the Python `instructions` list. A task is specified by using Python data structures that match the schema of a specific Connect agent. At the end of the Python code, all tasks in the `instructions` list are automatically generated and executed. The following example shows how to fill the `instructions` list with three ExcelWriter tasks that write different parameters (`p0`, `p1`, `p2`) into separate Excel workbooks (`data_p0.xlsx`, `data_p1.xlsx`, `data_p2.xlsx`).

```
- GAMSReader:
    readAll: True
- PythonCode:
    code: |
      symbols = [ 'p0', 'p1', 'p2' ]
      for s in symbols:
        instructions.append(
        {
          'ExcelWriter':
          {
            'file': 'data_{}.xlsx'.format(s),
            'symbols': [{'name': s}]
          }
        })
```

Using `connect.container` allows to access the Connect database directly in the Python code. The `connect.container` is a GAMS Transfer Container object and data within the container is stored as Pandas DataFrames. Please refer to the documentation of GAMS Transfer to learn more about GAMS Transfer and its functionalities.

The following complete example shows how to access and modify Connect container data and manually add a new symbol with the modified data to the Connect container:

```
Set i /1, 2, 3/
    i_mod;

$onembeddedCode Connect:
- GAMSReader:
    readAll: True
- PythonCode:
    code: |
        i_mod_records = [ 'i'+n for n in connect.container.data["i"].records.iloc[:,0] ]
        connect.container.addSet("i_mod", ["*"], records=i_mod_records)
- GAMSWriter:
    symbols:
      - name: i_mod
$offembeddedCode

display i, i_mod;
```

The first line takes the data of set `i` and adds an i at the beginning of each uel in the first column of the dataframe. The last line writes the modified dataframe as set `i_mod` to the Connect database.

Here is another complete example modifying Connect container data and adding a new symbol with the modified data:

```
Parameter p;

$onEcho > p_raw.csv
i,j,2000,2001
i1,j1,1,2
i2,j2,3,4
i3,j3,5,6
$offEcho

$onEmbeddedCode Connect:
- CSVReader:
    file: p_raw.csv
```

```
    name: p_raw
    header: True
    indexColumns: "1,2"
    valueColumns: "2:lastCol"
- PythonCode:
    code: |
        p_records = [ [r[0] + '_' + r[1]] + list(r) for i,r in connect.container.data["p_raw"].r
        connect.container.addParameter("p", ["*"]*4, records=p_records)
- GAMSWriter:
    symbols:
      - name: p
$offEmbeddedCode

display p;
```

In this example, we take the data of parameter `p_raw` and insert a column of the concatenated row dimensions into the first column of the dataframe. The modified dataframe is then added to the Connect container as records for the new symbol `p`. Here is a display of GAMS parameter `p`:

```
INDEX 1 = i1_j1

             2000        2001

i1.j1       1.000       2.000

INDEX 1 = i2_j2

             2000        2001

i2.j2       3.000       4.000

INDEX 1 = i3_j3

             2000        2001

i3.j3       5.000       6.000
```

4.46.5.16 RawCSVReader

The RawCSVReader allows reading of unstructured data from a specified CSV file into the Connect database. Due to performance issues this agent is recommended for small to medium sized unstructured CSV only. This reader works similarly compared to the RawExcelReader agent. It reads the entire CSV file and represents its content in a couple of GAMS sets:

- `r / r1, r2, ... /` (rows)
- `c / c1, c2, ... /` (columns)
- `vs(r,c) / s1.r1.c2 "cell text", ... /` (cells with explanatory text)
- `vu(r,c,*) / s1.r1.c1."cell text" "cell text", ...` (cells with potential GAMS label)

and a parameter `vf(r,c) / r2.c2 3.14, ... /` (cells with numerical values). Unlike RawExcelReader cells with a *date* will be not interpreted and stored in `vs` and `vu`. Cells with a string value will be stored in `vs`. If the string length exceeds the maximum length allowed for elements text, it will be truncated.

RawCSVReader will try to represent the cell value as a number and if this succeeds stores the number in `vf`. Strings of GAMS special values `INF`, `-INF`, `EPS`, `NA`, and `UNDEF` as well as `TRUE` and `FALSE` will be also converted to its numerical counterpart. It will also try to represent the cell value as a string and stores this as a label in the third position in `vu`. GAMS labels have a length limitation and hence RawCSVReader automatically shortens the label to fit this limit. RawCSVReader will provide a unique label (ending in ~`n` where `n` is an integer for strings exceeding the label length limit) for each string in the CSV file. The full string (if it fits) will be available as the element text of the `vu` record.

Option	Default	Description
cName	`c`	Symbol name for columns.
columnLabel	`C`	Label for columns.
file	None	Specify a CSV file path.
readAsString	`True`	Control the automatic pandas type conversion of cells.
readCSVArguments	None	Dictionary containing keyword arguments for the `pandas.read_csv` method.
rName	`r`	Symbol name for rows.
rowLabel	`R`	Label for rows.
trace	inherited	Specify the trace level for debugging output.
vfName	`vf`	Symbol name for cells with a numerical value.
vsName	`vs`	Symbol name for cells with an explanatory text.
vuName	`vu`	Symbol name for cells with a potential GAMS label.

Detailed description of the options:

cName = *string* (default=`c`)

Control the name of the set of columns.

columnLabel = *string* (default=`C`)

Control the labels for the set of columns (`c`).

file = *string* (required)

Specify a CSV file path.

readAsString = *boolean* (optional)

Control the type of cells returned by `pandas.read_csv`. If this is set to `True` (default) all cells are returned as string and the agent tries to interpret the string itself. If this is set to `False` pandas will try to infer the type. In such a case the agent can't distinguish between cells with `1` and `1.00` because pandas turned the integer `1` already into a float and the agent has no way for recovering the original cell.

readCSVArguments = *dictionary* (optional)

Dictionary containing keyword arguments for the `pandas.read_csv` method. By specifying `readCSVArguments`, it is possible to pass arguments directly to the `pandas.read_csv` method. For example, `readCSVArguments:` {delimiter: ';'}.

rName = *string* (default=`r`)

Control the name of the set of rows.

rowLabel = *string* (default=`R`)

Control the labels for the set of rows (`r`).

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace > 1` some scalar debugging output will be written to the log. For `trace > 2` the cell values and it's processing will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

vfName = *string* (default=`vf`)

Control the name of the parameter for cells with a numerical value.

vsName = *string* (default=`vs`)

Control the name of the set for cells with an explanatory text.

vuName = *string* (default=`vu`)

Control the name of the set for cells with a potential GAMS label.

4.46.5.17 RawExcelReader

The RawExcelReader allows reading of unstructured data from a specified Excel file into the Connect database. Due to performance issues this agent is recommended for small to medium sized unstructured Excel files only. This reader works similarly compared to the xlsdump tool. It reads the entire spreadsheet and represents its content in a couple of GAMS sets:

- `s /s1, s2,.../` (workbook sheets)
- `w / Sheet1, Sheet2, ... /` (workbook sheets by name)
- `ws(s,w) / s1.Sheet1, s2.Sheet2, ... /` (workbook map)
- `r / r1, r2, ... /` (rows)
- `c / c1, c2, ... /` (columns)
- `vs(s,r,c) / s1.r1.c2 "cell text", ... /` (cells with explanatory text)
- `vu(s,r,c,*) / s1.r1.c1."cell text" "cell text", ...` (cells with potential GAMS label)

and a parameter `vf(s,r,c) / s1.r2.c2 3.14, ... /` (cells with numerical values). Cells with a *date* will be stored in it's string representation in `vu` and as a Julian date in `vf`. Cells with a string value will be stored in `vs`. If the string length exceeds the maximum length allowed for elements text, it will be truncated. Excel offers many other cell value types. RawExcelReader will try to represent the cell value as a number and if this succeeds stores the number in `vf`. Strings of GAMS special values `INF`, `-INF`, `EPS`, `NA`, and `UNDEF` will be also converted to its numerical counterpart. It will also try to represent the cell value as a string and stores this as a label in the fourth position in `vu`. GAMS labels have a length limitation and hence RawExcelReader automatically shortens the label to fit this limit. RawExcelReader will provide a unique label (ending in `~n` where n is an integer for strings exceeding the label length limit) for each string in the workbook. The full string (if it fits) will be available as the element text of the `vu` record.

Option	Default	Description
cName	`c`	Symbol name for columns.
columnLabel	`C`	Label for columns.

Option	Default	Description
file	None	Specify an Excel file path.
mergedCells	`False`	Control the handling of empty cells that are part of a merged Excel range.
rName	`r`	Symbol name for rows.
rowLabel	`R`	Label for rows.
sheetLabel	`S`	Label for workbook sheets.
sName	`s`	Symbol name for workbook sheets.
trace	inherited	Specify the trace level for debugging output.
vfName	`vf`	Symbol name for cells with a numerical value.
vsName	`vs`	Symbol name for cells with an explanatory text.
vuName	`vu`	Symbol name for cells with a potential GAMS label.
wName	`w`	Symbol name for workbook sheets by name.
wsName	`ws`	Symbol name for workbook map.

Detailed description of the options:

cName = *string* (default=`c`)

Control the name of the set of columns.

columnLabel = *string* (default=`C`)

Control the labels for the set of columns (`c`).

file = *string* (required)

Specify an Excel file path.

mergedCells = *boolean* (default=`False`)

Control the handling of empty cells that are part of a merged Excel range. If `False`, the cells are left empty. If `True`, the merged value is used in all cells. Note that setting this option to `True` has an impact on performance since the Excel workbook has to be opened in a non-read-only mode that results in non-constant memory consumption (no lazy loading).

rName = *string* (default=`r`)

Control the name of the set of rows.

rowLabel = *string* (default=`R`)

Control the labels for the set of rows (`r`).

sheetLabel = *string* (default=`S`)

Control the labels for the set of workbook sheet (`s`).

sName = *string* (default=`s`)

Control the name of the set of workbook sheets.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace > 1` some scalar debugging output will be written to the log. For `trace > 2` the cell values and it's processing will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

vfName = *string* (default=`vf`)

Control the name of the parameter for cells with a numerical value.

vsName = *string* (default=`vs`)

Control the name of the set for cells with an explanatory text.

vuName = *string* (default=`vu`)

Control the name of the set for cells with a potential GAMS label.

wName = *string* (default=`w`)

Control the name of the set of workbook sheets by name.

wsName = *string* (default=`ws`)

Control the name of the set of the workbook map.

4.46.5.18 SQLReader

The SQLReader agent allows to read symbols (sets and parameters) from a specified database management system into the Connect database. It connects to MySQL, Postgres, MS-SQL (SQL-Server), SQLite and PyODBC through their respective python packages to provide native SQL query support. Further, it also utilizes `pandas.DataFrame` class' I/O API method `read_sql` to connect to any other database provided the relevant drivers are present on the system. See Simple Connect Example for SQL for a simple example that uses the SQLReader.

Note

The connectivity to MS-Access databases is available on Windows only and requires a 64-bit MS-Access ODBC driver. See connection for more information.

Option	Scope	Default	Description
connection	root	None	Connection dictionary to specify credentials for the database.
connectionArguments	root	None	Dictionary containing keyword arguments for the `connect` constructor of the respective SQL library or for the `sqlalchemy.create_engine` constructor.
connectionType	root	`sqlite`	Specify the connection type to be used in order to connect to that database.
dTypeMap	root/symbols	None	Dictionary used to specify the dtype of columns.
indexSubstitutions	symbols	None	Dictionary used for substitutions in the index columns.
name	symbols	None	Specify the name of the symbol in the Connect database.
query	symbols	None	Specify the SQL query.
readSQLArguments	symbols	None	Dictionary containing keyword arguments for the
			`pandas.read_sql` method.
symbols	root	None	Specify symbol specific options.
trace	root	inherited	Specify the trace level for debugging output.
type	root/symbols	`par`	Control the symbol type.

Detailed description of the options:

connection = *dict* (required)

Allows to specify the credentials to access the database. Below are examples for connection dictionaries based on the selected `connectionType`.

SQLite:

```
connection: {'database': 'absolute//path//to//datafile.db'}
```

Postgres/MySQL/SQLServer:

```
connection: {'user': <username of the database server>, 'password': <password>, 'host': <hostnam
```

MS-Access:

```
connection: {'DRIVER': 'Microsoft Access Driver (*.mdb, *.accdb)', 'DBQ': 'absolute//path//to//d
```

Note

The connectivity to MS-Access databases is available on Windows only and requires a 64-bit MS-Access ODBC driver. If no MS-Access is installed or a 32-bit version of MS-Access, download and install a 64-bit MS-Access ODBC Driver as a redistributable package from MS (e.g. MS-Access 2013 Runtime_x64).

SQLAlchemy:

Use `connectionType: sqlalchemy` to connect to various databases. Note that the argument `drivername: <dialect+driver>` is required.

For dialect *SQLite*:

```
connection: {'drivername': 'sqlite', 'database': 'absolute//path//to//datafile.db'}
```

For dialect *Postgres*:

```
connection: {'drivername': 'postgresql+psycopg2', 'username': <username of the database server>,
```

For dialect *MySQL*:

```
connection: {'drivername': 'mysql+pymysql', 'username': <username of the database server>, 'pass
```

For dialect *MS-SQL(SQLServer)*:

```
connection: {'drivername': 'mssql+pymssql', 'username': <username of the database server>, 'pass
```

For dialect *MS-Access*:

```
connection: {'drivername': 'access+pyodbc', 'query': {'odbc_connect': 'DRIVER={Microsoft Access
```

Note

Connecting to databases other than SQLite, Postgres, MySQL and MS-SQL through SQLAlchemy would require the respective driver of that database to be present on the system. For example, in the above example `sqlalchemy-access` must be available in order to connect to MS-Access through SQLAlchemy.

PyODBC:

Use `connectionType: pyodbc` to connect to various databases. Therefore, either specify the DSN (Data Source Name):

```
connection: {'DSN': <Data Source Name>}
```

or use:

```
connection: {'user': <username of the database server>, 'password': <password>, 'host': <hostnam
```

connectionArguments = *dict* (optional)

Dictionary containing keyword arguments for the `connect` constructor of the respective SQL library or for the `sqlalchemy.create_engine` constructor.

connectionType = *string* (default=`sqlite`)

Following is the list of valid options for connection type.

1. `sqlite` (default)
2. `postgres`
3. `mysql`
4. `sqlserver`
5. `access`
6. `sqlalchemy`
7. `pyodbc`

A connection using `pyodbc` can be established to any database for which a DSN (Data Source Name) is available on the system. The `pyodbc` connection works readily on windows platform where DSN is already setup. Further, for non-windows platforms, the user is responsible for installing the `pyodbc` package and handling any other dependencies for the local python installation.

dTypeMap = *dict* (optional)

Allows to specify the dtype of columns in a dictionary as `key: value` pairs, i.e. <`column`>: <`dtype`>.

indexSubstitutions = *dict* (optional)

Dictionary used for substitutions in the index columns. Each key in `indexSubstitutions` is replaced by its corresponding value. This option allows arbitrary replacements in the index columns.

name = *string* (required)

Name of the symbol in the Connect database. The name must be unique for each symbol.

query = *string* (required)

Specify the SQL query which will fetch the desired table from the database system.

readSQLArguments = *dict* (optional)

Dictionary containing keyword arguments for the `pandas.read_sql` method. Not all arguments of that method are exposed through the YAML interface of the SQLReader agent. By specifying `readSQLArguments`, it is possible to pass arguments directly to the `pandas.read_sql` method that is used by the SQLReader agent if `connectionType` is `sqlalchemy`. If `connectionType` is not `sqlalchemy`, this option can be used to parameterize the SQL query. A small example showing how this option can be used is given below.

```
query: "SELECT i,j FROM stock_table WHERE value_col > %(value)s"
readSQLArguments: {'value': 10}
```

Note

Different Database management systems use different parameter markers and some do not support named parameter markers, e.g., `PyODBC`.

symbols = *list* (required)

A list containing symbol specific options.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace > 1` some scalar debugging output will be written to the log. For `trace > 2` the intermediate arrays and data frames will be written abbreviated to the log. For `trace > 3` the intermediate arrays and data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

type = *string* (default=`par`)

Control the symbol type. Supported symbol types are `par` for GAMS parameters and `set` for GAMS sets.

valueColumns = *list or string* (optional)

Specify columns to get the values from. The value columns contain numerical values in case of a GAMS parameter and set element text in case of a GAMS set. The columns are given as column names represented as a list of strings. For example: `valueColumns: ["i1", "i2"]`. If there is more than one value column specified, the column names are stacked to index automatically. As string one can specify the symbolic constant `lastCol` (i.e. `valueColumns: "lastCol"`) or an empty string (i.e. `valueColumns: ""`). When `lastCol` is passed the last column will be treated as a value column and all the other columns will be treated as index columns. When an empty string is passed all columns will be treated as index columns. Specifying an empty string is only valid for symbol type `set` since symbol type `par` requires at least one value column. The default for symbol type `par` is `lastCol` and the default for symbol type `set` is an empty string.

valueSubstitutions = *dict* (optional)

Dictionary used for mapping in the value column of the `DataFrame`. Each key in `valueSubstitutions` is replaced by its corresponding value. The replacement is only performed on the value column of the `DataFrame` which is the numerical value in case of a GAMS parameter and the set element text in case of a GAMS set. While it is possible to make arbitrary replacements this is especially useful for controlling sparse/dense reading.

4.46.5.19 SQLWriter

The SQLWriter agent allows to write symbols (sets and parameters) from the Connect database to a specified database management system. Variables and equations need to be turned into parameters with the Projection agent before they can be written. It connects to MySQL, Postgres, MS-SQL (SQL-Server), SQLite and PyODBC through their respective python packages to provide faster write operations. Further, it also utilizes `pandas.DataFrame` class' I/O API method `to_sql` to connect to any other database provided the relevant drivers are present on the system. See Simple Connect Example for SQL for a simple example that uses the SQLWriter.

Note

The connectivity to MS-Access databases is available on Windows only and requires a 64-bit MS-Access ODBC driver. See connection for more information.

Option	Scope	Default	Description
connection	root	None	Connection dictionary to specify credentials for the database.
connectionArguments	root	None	Dictionary containing keyword arguments for the `connect` constructor of the respective SQL library or for the `sqlalchemy.create_engine` constructor.
connectionType	root	None	Specify the connection type to be used in order to connect to that database.
ifExists	root/symbols	`fail`	Specify the behavior when a table with the same name exists in the database/schema.
insertMethod	root/symbols	`default`	Specify the insertion method to be used to write the table in the database.
name	symbols	None	Specify the name of the symbol in the Connect database.
schemaName	root/symbols	None	Specify the schema name.
symbols	root	None	Specify symbol specific options.
tableName	symbols	None	Specify the SQL table/relation in the provided database/schema.
toSQLArguments	root/symbols	None	Dictionary containing keyword arguments for the `pandas.to_sql` method.
trace	root	inherited	Specify the trace level for debugging output.
unstack	root/symbols	`False`	Indicate if the last index column will be used as a header row.
valueSubstitutions	root/symbols	None	Dictionary used for mapping in the value column of the `DataFrame`.
writeAll	root	`auto`	Indicate if all set and parameter type symbols in the Connect database will be written to the specified DBMS.

Detailed description of the options:

connection = *dict* (required)

Allows to specify the credentials to access the database. See SQLReader connection for further details and some examples.

Note

The connectivity to MS-Access databases is available on Windows only and requires a 64-bit MS-Access ODBC driver. If no MS-Access is installed or a 32-bit version of MS-Access, download and install a 64-bit MS-Access ODBC Driver as a redistributable package from MS (e.g. MS-Access 2013 Runtime_x64).

connectionArguments = *dict* (optional)

Dictionary containing keyword arguments for the `connect` constructor of the respective SQL library or for the `sqlalchemy.create_engine` constructor.

connectionType = *string* (default=`sqlite`)

Following is the list of valid options for connection type.

1. `sqlite` (default)
2. `postgres`
3. `mysql`
4. `sqlserver`
5. `access`
6. `sqlalchemy`
7. `pyodbc`

A connection using `pyodbc` can be established to any database for which a DSN (Data Source Name) is available on the system. The `pyodbc` connection works readily on windows platform where DSN is already setup. Further, for non-windows platforms, the user is responsible for installing the pyodbc package and handling any other dependencies for the local python installation.

Note

`pyodbc` provides a simple and consistent API to connect to many different databases using the Open Database Connectivity (ODBC) interface. This introduces the challenge that database specific properties are not taken into consideration. For example, different databases have different escaping syntax in order to escape special characters in column names of a table. Thus, the implementation for connection type `pyodbc` is kept as a general purpose connector to different databases while not escaping the special characters.

ifExists = *string* (default=`fail`)

Specify the behavior when a table with the same name exists in the database/schema. Valid values are `fail`, `replace` and `append`.

insertMethod = *string* (default=`default`)

Specify the insertion method to be used to write the table in the database. Valid values are `bcp`, `bulkInsert`, and `default`.

Note

The options `bulkInsert` and `bcp` is not available with connection type `sqlalchemy`. Further, the option `bcp` is reserved for connection type `sqlserver`. This is useful when the SQL-Server database is available remotely. The `bcp` method uses MS-SQL's command line utility of the same name to insert huge amount of data. In order to use this insertion method, the user must download and install the utility as per the system requirements. The utility is available for `Windows` as well as for `Linux and macOS`. On the other hand, if the SQL-Server database is available locally, then the `bulkInsert` method can be used and the `bcp` utility is not required.

The option `bulkInsert` uses special SQL queries for connection type `mysql`, `postgres`, `sqlserver` and `access`. It creates a temporary csv file and then imports the same in the database. It is to be noted that for MySQL the option `LOCAL_INFILE` must be enabled at the server side for this method to work successfully.

name = *string* (required)

Specify the name of the symbol in the Connect database.

schemaName = *string* (optional)

Specify the schema name for writing the table to the correct location. In Postgres, by default, it writes to a public schema already present in every database.

symbols = *list* (required)

A list containing symbol specific options.

tableName = *string* (required)

Name of the SQL table/relation in the provided database/schema.

toSQLArguments = *dict* (optional)

Dictionary containing keyword arguments for the `pandas.to_sql` method. Not all arguments of that method are exposed through the YAML interface of the SQLWriter agent. By specifying `toSQLArguments`, it is possible to pass arguments directly to the `pandas.to_sql` method that is used by the SQLWriter agent if `connectionType` is `sqlalchemy`.

trace = *integer* (optional)

Specify the trace level for debugging output. For `trace` > `1` some scalar debugging output will be written to the log. For `trace` > `2` the intermediate arrays and data frames will be written abbreviated to the log. For `trace` > `3` the intermediate arrays and data frames will be written entirely to the log (potentially large output). If `trace` has not been set, the `trace` value, set by the Options agent, will be used.

unstack = *boolean* (default=`False`)

Indicate if the last index column will be used as a header row.

valueSubstitutions = *dict* (optional)

Dictionary used for mapping in the value column of the `DataFrame`. Each key in `valueSubstitutions` is replaced by its corresponding value. The replacement is only performed on the value column of the `DataFrame` which is the numerical value in case of a GAMS parameter, variable or equation and the set element text in case of a GAMS set.

writeAll = *boolean or* ***auto*** (default=`auto`)

Indicate if all set and parameter type symbols in the Connect database will be written to the specified DBMS. The default `auto` becomes `True` if there are no symbol options specified, otherwise `False`. If `True`, each symbol is written to a table in the specified DBMS where the table name is defined by the name of the symbol.

4.46.6 Examples

This section provides a collection of more complex examples. For simple examples see section Getting Started.

4.46.6.1 Connect Example for Excel (executeTool win32.ExcelMerge)

The following example shows how to read and write Excel files in Connect. On Windows with Excel installed, the output sheets are merged back into the input workbook using tool win32.ExcelMerge. The entire code is listed at the end of the example. This model is part of DataLib as model **connect05**. First, the original matrix `a` is read using the GAMSReader and is then written to input.xlsx using the ExcelWriter. After clearing symbols `i` and `a` in the GAMS database, the ExcelReader is used to read file input.xlsx back in and create parameter `a` in the Connect database. The Projection agent extracts set `i` from parameter `a`. With the GAMSWriter, symbols `i` and `a` are written to the GAMS database. The tool linalg.invert calculates the inverse `inva` of `a` which is then written to output.xlsx using Connect's GAMSReader and ExcelWriter at execution time. The following lines then check if the code is not executed on a UNIX system and if Excel is available. If both is true, output.xlsx is merged into input.xlsx using tool win32.ExcelMerge and both symbols `inva` and `a` can be read from input.xslx with a single instance of the ExcelReader. If the code is executed on a UNIX system and/or Excel is not available, output.xlsx can not be merged into input.xlsx, and both files need to be read to create the symbols `inva` and `a`. The last part makes sure that `inva` is the inverse of `a`.

```
set i / i1*i3 /; alias (i,j,k);
table a(i,j) 'original matrix'
      i1     i2     i3
i1    1      2      3
i2    1      3      4
i3    1      4      3
;
$onEmbeddedCode Connect:
- GAMSReader:
    symbols:
      - name: a
- ExcelWriter:
    file: input.xlsx
    symbols:
      - name: a
$offEmbeddedCode

$onMultiR
$clear i a

$onEmbeddedCode Connect:
- ExcelReader:
    file: input.xlsx
    symbols:
      - name: a
- Projection:
    name: a(i,j)
    newName: i(i)
    asSet: True
- GAMSWriter:
    writeAll: True
$offEmbeddedCode i a
```

```
parameter
  inva(i,j) 'inverse of a'
  chk(i,j)  'check the product a * inva'
  ;

executeTool.checkErrorLevel 'linalg.invert i a inva';

EmbeddedCode Connect:
- GAMSReader:
    symbols:
      - name: inva
- ExcelWriter:
    file: output.xlsx
    symbols:
      - name: inva
endEmbeddedCode

Scalar mergedRead /0/;
executeTool 'win32.msappavail Excel';
mergedRead$(errorLevel=0) = 1;

if (mergedRead,
    executeTool.checkErrorLevel 'win32.excelMerge output.xlsx input.xlsx';
    EmbeddedCode Connect:
    - ExcelReader:
        file: input.xlsx
        symbols:
          - name: a
          - name: inva
    - GAMSWriter:
        writeAll: True
    endEmbeddedCode a inva
else
    EmbeddedCode Connect:
    - ExcelReader:
        file: input.xlsx
        symbols:
          - name: a
    - ExcelReader:
        file: output.xlsx
        symbols:
          - name: inva
    - GAMSWriter:
        writeAll: True
    endEmbeddedCode a inva
);

chk(i,j) = sum{k, a(i,k)*inva(k,j)};
chk(i,j) = round(chk(i,j),15);
display a,inva,chk;
chk(i,i) = chk(i,i) - 1;
abort$[card(chk)] 'a * ainv <> identity';
```

4.46.6.2 Connect Example for Excel

The following example shows how to read and write Excel files in Connect. The entire code is listed at the end of the example. This model is part of DataLib as model **connect01**. The example (inspired by

the model **herves**) reads a 3-dimensional parameter from a spreadsheet that has one row index (`code`) at the left side of the table and the other row index (`labId`) at the right of the table. A column index (`cut`) is at the top of the table. The column index consists of floating-point numbers. The goal it to read the data into GAMS but modify the labels of some sets: Only the first two decimal digits of the elements in `cut` are significant. Moreover, the `labId` should be prefixed with an `L`. A new spreadsheet with the new labels should be written. The layout of the table should remain with the exception of moving the `labId` column also to the left. Here is a screenshot of the original table:

The following GAMS code uses a separate GAMS program (`getdata.gms`) to get the raw data from the original spreadsheet. Connect runs inside a compile-time embedded code section and uses the Connect agent RawExcelReader to get the raw Excel data. In some subsequent GAMS code the sets `rr` and `cut[Id]` as well as the parameter `raw` are filled knowing the layout of the table (the code is written in a way that the table can grow). This GAMS program gets executed and instructed to create a GDX file. In a compile-time embedded Connect section the relevant symbols (`rr`, `cutId`, and `raw`) are read from this GDX file. The Projection agent extracts the domain `labid` from the set `rr` and some Python code using Connect agent PythonCode makes the label adjustments and sorts the data nicely. The Python code uses the connect.container methods to read from and write to the Connect database. Finally, the GAMSWriter agent sends the data to GAMS. In the main program at execution-time an embedded Connect code section exports the `labdata` parameter in the required form (after reading it from GAMS with the GAMSReader agent). Here is a screenshot of the resulting table:

In the remainder of the GAMS code another execution-time embedded Connect code is used to read the data back from the newly created spreadsheet using Connect agent ExcelReader. The set `rr` is created from parameter `labdata` using the Projection agent and everything is written back to GAMS with Connect agent GAMSWriter. The original data and the data from the newly created spreadsheet are exported to GDX (using execute_unload) and compared to verify that the data is identical by calling gdxdiff.

```
Set code, labId, cut, rr(code<,labId);
parameter labdata(code,labid,cut);
$onEcho > getdata.gms
* Symbols for RawExcelReader
alias (u,*); Set s,w,r,c,ws(s,w),vs(s,r,c),vu(s,r,c,u); Parameter vf(s,r,c);
$onEmbeddedCode Connect:
- RawExcelReader:
    file: labdata.xlsx
- GAMSWriter:
    writeAll: True
$offEmbeddedCode

* Symbols to be filled
alias (*,code,labId,cut); Parameter raw(code,labId,cut); Set cutId, rr(code,labId)
Set cX(c,cut) 'column index', rX(r,code,labId) 'row index';
Singleton set cLast(c); Scalar lastPos;
loop(ws(s,'ZAg'),
  lastPos = smax(vu(s,r,c,u), c.pos); cLast(c) = c.pos=lastPos;
  loop(r$(ord(r)>4),
    rX(r,code,labId) $= vu(s,r,'C1',code) and vu(s,r,cLast,labId));
  loop(c$(ord(c)>1 and not cLast(c)),
    cX(c,cut) $= vu(s,'R4',c,cut));
  loop((rX(r,code,labId),cX(c,cut)),
    raw(code,labId,cut) = vf(s,r,c))
  loop(cX(c,cut),
    cutId(cut) = yes)
);
option rr<rX;
$offEcho
$call.checkErrorLevel gams getdata.gms lo=%gams.lo% gdx=getdata.gdx
```

```
$onEmbeddedCode Connect:
- GDXReader:
    file: getdata.gdx
    symbols: [ {name: rr}, {name: raw}, {name: cutId, newName: cut} ]
- Projection:
    name: rr(code,labid)
    newName: labid(labid)
- PythonCode:
    code: |
      labid_records = sorted([ 'L'+t[0] for t in connect.container.data['labid'].records.values ], ke
      rr_records = sorted([ (t[0],
                             'L'+t[1]) for t in connect.container.data['rr'].records.values ], key=
      # Trim elements of set cut to two decimal places
      cut_records = sorted([ '{:.2f}'.format(float(t[0])) for t in connect.container.data['cut'].reco
      labdata_records = [ (t[0],
                           'L'+t[1],
                           '{:.2f}'.format(float(t[2])),
                           t[-1]) for t in connect.container.data['raw'].records.values ]

      connect.container.addSet('labid_mod', ['*'], records=labid_records)
      connect.container.addSet('rr_mod', ['*']*2, records=rr_records)
      connect.container.addSet('cut_mod', ['*'], records=cut_records)
      connect.container.addParameter('labdata', ['*']*3, records=labdata_records)
- GAMSWriter:
    symbols: [ {name: labid_mod, newName: labid}, {name: rr_mod, newName: rr}, {name: cut_mod, newNam
$offEmbeddedCode
execute_unload 'labdata.gdx', labdata, cut, rr;

* Reintroduce 0 (zeros)
labdata(rr,cut) = labdata(rr,cut) + eps;

execute 'rm -f labdatanew.xlsx';
* Write new workbook with good table
EmbeddedCode Connect:
- GAMSReader:
    symbols: [ {name: labdata} ]
- ExcelWriter:
    file: labdatanew.xlsx
    valueSubstitutions: {EPS: 0}
    symbols:
      - name: labdata
        range: ZAg!A4
endEmbeddedCode
option clear=rr, clear=labdata;

EmbeddedCode Connect:
- ExcelReader:
    file: labdatanew.xlsx
    symbols:
      - name: labdata
        rowDimension: 2
        range: ZAg!A4
- Projection:
    name: labdata(code,labid,cut)
    newName: rr(code,labid)
    asSet: True
- GAMSWriter:
    writeAll: True
```

```
endEmbeddedCode
execute_unload 'labdatanew.gdx', labdata, cut, rr;
execute.checkErrorLevel 'gdxdiff labdata.gdx labdatanew.gdx > %system.NullFile%';
```

4.46.6.3 Connect Example for CSV

The following example shows how to read and write CSV files in Connect. The entire code is listed at the end of the example. This model is part of DataLib as model **connect02**. It starts out with defining some data (`stockprice`) in a table statement in GAMS. With compile-time embedded Connect code utilizing the GAMSReader agent to bring this data into Connect and exporting it as a CSV file with agent CSVWriter. The GDXWriter agent also creates a GDX file with the data which is then used in a subsequent call to feed gdxdump that produces the same CSV file as CSVWriter. The text comparison tool `diff` is used to compare the two CSV files. The CSV file look as follows:

```
"date_0","AAPL","GOOG","MMM","MSFT","WMT"
"2012-20-11",12.124061,314.008026,60.966354,21.068886,46.991535
"2112-20-11",12.139372,311.741516,60.731037,20.850344,47.150307
"2212-20-11",12.203673,313.674286,61.467381,20.890808,46.991535
"2312-20-11",12.350039,315.387848,62.401108,21.068886,47.626663
"2712-20-11",12.448025,318.929565,62.461876,21.076981,47.499634
"2812-20-11",12.328911,318.655609,61.604042,20.898905,47.420238
"2912-20-11",12.404848,320.000549,62.332813,21.060795,47.626663
"3012-20-11",12.401172,321.744019,62.044331,21.012224,47.444057
```

In remainder of the example this CSV file is read back via the Connect agent CSVReader. The code also utilizes the tool csv2gdx to read the CSV file into a GDX file. The code compares the results of both methods. Csv2gdx also creates sets with the index elements as `Dim1`, `Dim2`, ... Therefore, Connect utilizes the Projection agent to extract the index sets `date` and `symbol` from the parameter `stockprice` as sets `Dim1` and `Dim2`. The Connect agent GDXWriter creates a GDX file of the Connect database which then can be compared with the GDX file created by csv2gdx. The GDX comparison tool gdxdiff is used to compare the two GDX files.

```
Set date,symbol;
Table stockprice(date<,symbol<)
                AAPL       GOOG       MMM       MSFT       WMT
2012-20-11 12.124061 314.008026 60.966354 21.068886 46.991535
2112-20-11 12.139372 311.741516 60.731037 20.850344 47.150307
2212-20-11 12.203673 313.674286 61.467381 20.890808 46.991535
2312-20-11 12.350039 315.387848 62.401108 21.068886 47.626663
2712-20-11 12.448025 318.929565 62.461876 21.076981 47.499634
2812-20-11 12.328911 318.655609 61.604042 20.898905 47.420238
2912-20-11 12.404848 320.000549 62.332813 21.060795 47.626663
3012-20-11 12.401172 321.744019 62.044331 21.012224 47.444057
;

* Use Connect CSVWriter to write GAMS data in CSV format moving the symbol index into the column (uns
$onEmbeddedCode Connect:
- GAMSReader:
    symbols: [ {name: stockprice} ]
- GDXWriter:
    file: sp_connect.gdx
    writeAll: True
- CSVWriter:
    file: sp_connect.csv
```

```
    name: stockprice
    header: True
    unstack: True
    quoting: 2
$offEmbeddedCode

* Use gdxdump to create a CSV file and text compare the Connect and gdxdump CSV files
$call.checkErrorLevel  gdxdump sp_connect.gdx output=sp_gdxdump.csv symb=stockprice format=csv colum
$call.checkErrorLevel  diff -q sp_connect.csv sp_gdxdump.csv > %system.nullFile%

* Use Connect CSVReader to read the newly created CSV file and deposit the result in a csv2gdx compat
$onEmbeddedCode Connect:
- CSVReader:
    file: sp_connect.csv
    name: stockprice
    indexColumns: 1
    valueColumns: "2:lastCol"
- Projection:
    name: stockprice(date,symbol)
    newName: Dim1(date)
    asSet: True
- Projection:
    name: stockprice(date,symbol)
    newName: Dim2(symbol)
    asSet: True
- GDXWriter:
    file: sp_connect.gdx
    writeAll: True
$offEmbeddedCode

* Use csv2gdx to create a GDX file and compare the Connect and csv2gdx GDX files
$call.checkErrorLevel csv2gdx sp_connect.csv output=sp_csv2gdx.gdx id=stockprice index=1 value=2..las
$call.checkErrorLevel gdxdiff sp_connect.gdx sp_csv2gdx.gdx > %system.NullFile%
```

4.46.7 Text Substitutions in YAML File

In many cases one would like to *parameterize* the text in the Connect instruction file. For example, some of the Connect agents require a file name. Instead of hard coding the file name into the YAML instructions, text substitutions allow to have a *place holder* for the attribute that is substituted out before giving the instructions to Connect. The place holder in the YAML file uses the syntax `%SOMETEXT%`, similar to the GAMS compile-time variables. For example:

```
- CSVReader:
    file: %MYFILENAME%
    name: distance
    indexColumns: [1, 2]
    valueColumns: [3]
```

Depending on how Connect runs, the substitution is done in various ways. The section Substitutions in Embedded Connect Code described the substitution mechanisms for embedded Connect code. When Connect is initiated via the command line parameters connectIn or connectOut, the user defined parameter specified by double-dash command line parameters and the given GAMS command line parameters, e.g. `%gams.input%` will be substituted in the YAML file. The list of parameters available for substitution is printed to the GAMS log at the beginning of the job in the section `GAMS Parameters defined`.

When Connect is initiated via the shell command `gamsconnect` all substitutions need to be specified on the command line:

```
gamsconnect myci.yaml key1=val1 key2=val2 ...
```

`key` can be just `MYFILENAME` or be composed like `gams.Input` or `system.dirSep`.

4.46.8 Use Connect Agents in Custom Python Code

Instead of passing instructions via one of the Connect interfaces, users can execute tasks directly in their Python code by creating an instance of `ConnectDatabase` and calling method `.exec_task(task)`. The `task` argument is expected to be a Python dictionary of form:

```
{
   '<agent name>':
   {
      '<root option1>': <value>,
      '<root option2>': <value>,
      ... ,
      '<root option3>':
      [
         {
             '<option1>': <value>,
             '<option2>': <value>,
             ...
         },
         {
             '<option1>': <value>,
             '<option2>': <value>,
             ...
         },
         ...
      ]
   }
}
```

Users can either construct the Python dictionary themselves or let YAML create the dictionary from a YAML script. The following example creates an instance of ConnectDatabase and executes two tasks: First, the CSV file `stockprice.csv` is read into the Connect database and second, the symbol `stockprice` is written to the GAMS database. In this example, the tasks are directly specified as Python dictionaries.

```
Set dates, stocks;
Parameter stockprice(dates<,stocks<);

$onEcho > stockprice.csv
date;symbol;price
2016/01/04;AAPL;105,35
2016/01/04;AXP;67,59
2016/01/04;BA;140,50
$offEcho

$onEmbeddedCode Python:
from gams.connect import ConnectDatabase
cdb = ConnectDatabase(gams._system_directory, ecdb=gams)
cdb.exec_task({'CSVReader': {'file': 'stockprice.csv', 'name': 'stockprice', 'indexColumns': [1, 2],
                             'valueColumns': [3], 'fieldSeparator': ';', 'decimalSeparator': ','}})
cdb.exec_task({'GAMSWriter': {'symbols': [{'name': 'stockprice'}]}})
$offEmbeddedCode

display stockprice;
```

We can also construct the Python dictionaries by using YAML:

```
Set dates, stocks;
Parameter stockprice(dates<,stocks<);

$onEcho > stockprice.csv
date;symbol;price
2016/01/04;AAPL;105,35
2016/01/04;AXP;67,59
2016/01/04;BA;140,50
$offEcho

$onEmbeddedCode Python:
import yaml
from gams.connect import ConnectDatabase
cdb = ConnectDatabase(gams._system_directory, gams)

inst = yaml.safe_load('''
- CSVReader:
    file: stockprice.csv
    name: stockprice
    indexColumns: [1, 2]
    valueColumns: [3]
    fieldSeparator: ';'
    decimalSeparator: ','
- GAMSWriter:
    symbols:
      - name: stockprice
''')
for task in inst:
  cdb.exec_Task(task)
$offEmbeddedCode

display stockprice;
```

Here YAML creates a list of dictionaries (i.e. a list of tasks) from the given YAML script.

4.46.9 Command Line Utility gamsconnect

The GAMS system directory contains the utility `gamsconnect` to run Connect instructions directly from the command line. On Windows the utility has the callable extension `.cmd`. This script wraps the Python script `connectdriver.py` by calling the Python interpreter that ships with GAMS. `gamsconnect` operates as the other Connect drivers on a YAML instruction file. The agents GAMSReader and GAMSWriter are not available from `gamsconnect` and will trigger an exception. Substitutions can be passed to `gamsconnect` via command line arguments as `key=value`, e.g. `filename=myfile.csv` and even `gams.scrdir=/tmp/`. `gamsconnect` is called like this:

```
gamsconnect <YAMLFile> [key1=value1 [key2=value2 [key3=value3 [...]]]]
```

4.47 Extrinsic Functions

4.47.1 Introduction

Mathematical functions play an important role in the GAMS language, especially for nonlinear models. Like other programming languages, GAMS provides a number of built-in or intrinsic functions. GAMS is used in an extremely diverse set of application areas and this creates frequent requests for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS Function Library Facility provides the means for managing this trade-off, since it allows users to import functions from an external library into a GAMS model. However, these external libraries can currently only provide functionality for the evaluation of functions (incl. their first and second derivatives) in a point. Solvers that need to analyze the algebraic structure of the model instance are therefore not able to work with extrinsic functions. This includes the class of *deterministic* global solvers, see column "Global" in this table, while, for example, *stochastic* global solvers can work with extrinsic functions.

In this chapter we will demonstrate how to access functions from an extrinsic function library in a GAMS model and we will describe the extrinsic function libraries that are included in the GAMS distribution. In addition, we will provide some pointers for users who wish to build their own extrinsic function library.

4.47.2 Using Function Libraries

Function libraries are made available to a model with the following compiler directive:

```
$funcLibIn <InternalLibName> <ExternalLibName>
```

Here `InternalLibName` is a handle that will be used to refer to the library inside the model source code, `ExternalLibName` is the file name of the shared library that implements the extrinsic functions. If no path is given, GAMS will look for the library in the directory `extrinsic_functions` in the GAMS standard locations and in the GAMS system directory. To access a library that does not reside in these standard places, the external name should include a relative or absolute path to the location of the library. GAMS will then search for the specified library using the mechanisms specific to the host operating system. When processing the directive $funcLibIn, GAMS will validate the library, make the included functions available for use and add a table of the included functions to the listing file.

Note

The function library facility gives users complete control over naming so that potential name conflicts between libraries can be avoided.

Before the individual functions may be used, they have to be declared in the following way:

```
Function <InternalFuncName> /<InternalLibName>.<FuncName>/;
```

Here `InternalFuncName` is the name of the individual function that will be used in the GAMS code. The user may choose this internal name freely and thus avoid potential naming conflicts. `InternalLibName` is the name of the function library as defined by the $funcLibIn directive and `FuncName` is the name of the individual function in the external function library. Once functions have been declared in this way they may be used like intrinsic functions.

Consider the following simple example:

```
$funcLibIn myLib tricclib

Function myCos /myLib.Cosine/
         mySin /myLib.Sine/
         myPi  /myLib.Pi/;

Scalar d;
d = myCos(myPi/3);
display d;
```

Note that in the first line the external trigonometric library `tricclib` is activated and the internal name `myLib` is specified for it. Then the functions are declared. Observe that `Cosine`, `Sine` and `Pi` are functions in the trigonometric library. After the library has been loaded and the functions have been declared, the functions may be used as usual. The trigonometric library is discussed in section Example: Trigonometric Library below.

4.47.3 Libraries that are included in the GAMS Distribution

In this section we will present the libraries that are included in the GAMS distribution: the Fitpack Library, the Piecewise Polynomial Library, the Stochastic Library and the LINDO Sampling Library. Note that the LINDO Sampling Library is only available for LINDO license holders. In addition, we will provide details on the Mutex Library.

In the tables that follow, the "Endogenous Classification" (last column) specifies in which models the function may legally appear. In order of least to most restrictive, the choices are *any*, *DNLP*, *NLP*, *none*. See section Classification of Models for details on model types in GAMS. Note well that functions classified as *any* are only permitted with exogenous (constant) arguments.

A word on the notation in the tables below: for function arguments, lower case indicates that an endogenous variable is allowed. For details on endogenous variables, see section Functions in Equation Definitions. Upper case function arguments indicate that a constant is required. Arguments in square brackets may be omitted: the default values used in such cases are specified in the function description provided.

4.47.3.1 The Fitpack Library

FITPACK by Paul Dierckx [43] is a Fortran-based library for one-dimensional and two-dimensional spline interpolations. This library has been repackaged to work with the GAMS Function Library Facility. The model **[FITLIB01]** from the GAMS Test Library is an example of the use of the library FITPACK inside GAMS.

Note that the supporting points to which the function will be fit need to be stored in a three-dimensional parameter, `fitdata`, in a GDX file `fit.gdx`. The first dimension is a function index, the second dimension is the index of the supporting point and the third dimension takes one of the following four values: `"w"` (weight), `"x"` (x-value), `"y"` (y-value) or `"z"` (z-value).

The FITPACK library is made available with the following directive:

```
$funcLibIn <InternalLibName> fitfclib
```

It provides the following functions:

Function	Description	End. Classif.
`fitFunc(FUNIND,x[,y])`	Evaluate spline	DNLP
`fitParam(FUNIND,PARAM[,VALUE])`	Read or set parameters	none

The function `FitParam` may be used to change certain parameters that are used for the evaluation. The following values are defined:

- 1: Smoothing factor (S)
- 2: Degree of spline in direction x (Kx)
- 3: Degree of spline in direction y (Ky)
- 4: Lower bound of function in direction x (LOx)
- 5: Lower bound of function in direction y (LOy)
- 6: Upper bound of function in direction x (UPx)
- 7: Upper bound of function in direction y (UPy)

4.47.3.2 The Piecewise Polynomial Library

The Piecewise Polynomial Library may be used to evaluate piecewise polynomial functions. An example is given in the model **[PWPLIB01]** in the GAMS Test Library. Note that the functions that are to be evaluated need to be defined and stored in a GDX file. The following code snippet serves as illustration:

```
* Define two piecewise polynomial functions
Table pwpdata(*,*,*) '1st index: function number, 2nd index: segment number, 3rd index: degree'
                 leftBound       0          1         2
       1.1       1               2.4       -2.7       0.3
       1.2       4               5.6       -4.3       0.5
       2.1       0               0         -6.3333    0
       2.2       0.3333          1.0370   -12.5554    9.3333
       2.3       0.6667          9.7792   -38.7791   29
;
* Write pwp data to gdx file, which will be read by external library (pwpcclib)
$gdxout pwp.gdx
$unload pwpdata
$gdxout
```

Observe that on each row of the table `pwpdata` we have the following entries:

```
FuncInd.SegInd    leftBound    Coef0    Coef1    Coef2
```

Here `FuncInd` sets a function index and `SegInd` defines the index of the segment (or interval) which is described. Further, `LeftBound` gives the lower bound of the segment. The upper bound will be taken from the lower bound on the following segment, or set to infinity in case it is the last segment. Finally, `CoefX` defines the `X`-th degree coefficient of the polynomial corresponding to this segment.

The Piecewise Polynomial Library is made available with the following directive:

```
$funcLibIn <InternalLibName> pwpcclib
```

It provides the following function:

Function	Description	End. Classif.
pwpFunc(FUNIND,x)	Piecewise polynomials	DNLP

4.47.3.3 The Stochastic Library

The Stochastic Library provides random deviates, probability density functions, cumulative density functions and inverse cumulative density functions for certain continuous and discrete distributions. This library is made available with the following directive:

```
$funcLibIn <InternalLibName> stodclib
```

The continuous distributions that are available with this library are the following:

Distribution	Description
beta(SHAPE_1,SHAPE_2)	**Beta distribution** with shape parameters SHAPE_1 and SHAPE_2, see MathWorld
cauchy(LOCATION,SCALE)	**Cauchy distribution** with location parameter LOCATION and scale parameter SCALE, see MathWorld
ChiSquare(DF)	**Chi-squared distribution** with degrees of freedom DF, see MathWorld
exponential(LAMBDA)	**Exponential distribution** with rate of change LAMBDA, see MathWorld
f(DF_1,DF_2)	**F-distribution** with degrees of freedom DF_1 and DF_2, see MathWorld
gamma(SHAPE,SCALE)	**Gamma distribution** with shape parameter SHAPE and scale parameter SCALE, see MathWorld
gumbel(LOCATION,SCALE)	**Gumbel distribution** with location parameter LOCATION and scale parameter SCALE, see MathWorld
invGaussian(MEAN,SHAPE)	**Inverse Gaussian distribution** with mean MEAN and scaling parameter SHAPE, see MathWorld
laplace(MEAN,SCALE)	**Laplace distribution** with mean MEAN and scale parameter SCALE, see MathWorld
logistic(LOCATION,SCALE)	**Logistic distribution** with location parameter LOCATION and scale parameter SCALE, see MathWorld
logNormal(LOCATION,SCALE)	**Lognormal distribution** with location parameter LOCATION and scale parameter SCALE, see MathWorld
normal(MEAN,STD_DEV)	**Normal distribution** with mean MEAN and standard deviation STD_DEV, see MathWorld
pareto(SCALE,SHAPE)	**Pareto distribution** with scaling parameter SCALE and shape parameter SHAPE, see MathWorld
rayleigh(SIGMA)	**Rayleigh distribution** with parameter SIGMA, see MathWorld
studentT(DF)	**Student's t-distribution** with degrees of freedom DF, see MathWorld
triangular(LOW,MID,HIGH)	**Triangular distribution** between LOW and HIGH, where MID is the most probable number, see MathWorld
uniform(LOW,HIGH)	**Uniform distribution** between LOW and HIGH, see MathWorld
weibull(SHAPE,SCALE)	**Weibull distribution** with shape parameter SHAPE and scaling parameter SCALE, see MathWorld

Further, the following discrete distributions are available:

Distribution	Description
`binomial(N,P)`	**Binomial distribution** with number of trials `N` and success probability `P` in each trial, see MathWorld
`geometric(P)`	**Geometric distribution** with success probability `P` in each trial, see MathWorld
`hyperGeo(TOTAL,GOOD,TRIALS)`	**Hypergeometric distribution** with total number of elements `TOTAL`, number of good elements `GOOD` and number of trials `TRIALS`, see MathWorld
`logarithmic(P-FACTOR)`	**Logarithmic distribution** with parameter `P-FACTOR`, also called log-series distribution, see MathWorld
`negBinomial(FAILURES,P)`	**Negative Binomial distribution** with `FAILURES` being the number of failures until the experiment is stopped and success probability `P` in each trial. The generated random number describes the number of successes until the defined number of failures is reached, see MathWorld
`poisson(LAMBDA)`	**Poisson distribution** with mean `LAMBDA`, see MathWorld
`uniformInt(LOW,HIGH)`	**Integer Uniform distribution** between `LOW` and `HIGH`, see MathWorld

Note that for each distribution the library offers the following four functions, where `DistributionName` is the name of the distribution as listed in the tables above, `parameters` are the parameters associated with each distribution, and `x` is the point at which the function is to be evaluated. Note that `x` may be an endogenous variable.

Function	Description	End. Classif. for Continuous Distributions	End. Classif. for Discrete Distributions
`d<DistributionName>(parameters)`	Generate a random deviate (sample from the distribution)	none	none
`pdf<DistributionName>(x,parameters)`	Probability density function	DNLP	none
`cdf<DistributionName>(x,parameters)`	Cumulative distribution function	DNLP	none
`icdf<DistributionName>(x,parameters)`	Inverse cumulative distribution function	DNLP	none

For example, the functions for the Normal distribution are

Function	Description	End. Classif.
`dNormal(MEAN,STD_DEV)`	Samples a random number from the Normal distribution	none
`pdfNormal(x,MEAN,STD_DEV)`	Probability density function for Normal distribution	DNLP
`cdfNormal(x,MEAN,STD_DEV)`	Cumulative distribution function for Normal distribution	DNLP
`icdfNormal(x,MEAN,STD_DEV)`	Inverse cumulative distribution function for Normal distribution	DNLP

Finally, the seed for the various random number generators can be set by using the following function:

Function	Description	End. Classif.
SetSeed(SEED)	Defines the seed for random number generator	none

In the following example, a sample of size 20 is generated from the Normal, Binomial, Cauchy, and Lognormal distributions each:

```
$funcLibIn stolib stodclib
Functions randnorm     /stolib.dnormal    /
          randbin      /stolib.dbinomial  /
          randcauchy   /stolib.dcauchy    /
          randlognorm  /stolib.dlognormal /;

Set i / i1*i20 /;
Set j / norm, binomial, cauchy, lognorm /;
Parameter randx(i,j)    "distribution sample";

randx(i,"norm")     = randnorm(5,2);
randx(i,"binomial") = randbin(10,0.5);
randx(i,"cauchy")   = randcauchy(5,1);
randx(i,"lognorm")  = randlognorm(1.2,0.3);
display randx;
```

In the example, first the stochastic library is made available in GAMS, then the functions that will be used from the library are declared, giving them names under which to refer to them in the GAMS model.

The output generated by the display statement is the following:

```
----     15 PARAMETER randx  distribution sample

           norm    binomial      cauchy     lognorm

i1        4.373       4.000       5.520       3.132
i2        5.655       6.000       6.813       4.192
i3        5.927       6.000       5.426       2.801
i4        1.340       4.000       5.898       3.689
i5        3.537       3.000      -3.069       2.746
i6        3.057       5.000       5.518       3.430
i7        4.212       3.000       0.136       1.577
i8        6.869       7.000       5.068       3.857
i9        3.481       4.000     -13.856       3.977
i10       5.001       4.000       5.274       2.261
i11       3.182       5.000       4.383       2.686
i12       5.688       6.000       2.914       2.102
i13       3.675       6.000       4.693       3.105
i14       4.028       5.000       1.813       2.418
i15       8.767       5.000      12.190       2.182
i16       3.558       3.000    -112.644       2.092
i17       2.402       4.000       4.078       2.523
i18       2.249       2.000       0.996       2.777
i19       5.639       4.000       3.931       2.678
i20       7.374       4.000       4.671       3.159
```

4.47.3.4 The LINDO Sampling Library

The LINDO Sampling Library provides samples of random numbers for certain distributions.

It is made available by the following directive:

```
$funcLibIn <InternalLibName> lsadclib
```

Observe that a LINDO license is required to use this library.

The following table list the LINDO sampling functions.

Function	Description	End. Classif.
sampleLS<DistributionName>(parameters)	Creates a sample using the distribution DistributionName, according to the distribution parameters, and returns a HANDLE that references the sample, as illustrated in the example below.	none
getSampleValues(HANDLE)	Retrieves sampling created by the function sampleLS. See example below.	none
induceCorrelation(CORTYPE)	Induces the correlation that was set with the function setCorrelation before. CORTYPE describes the correlation type: 0 (Pearson), 1 (Kendall) or 2 (Spearman). See example below.	none
setCorrelation(SAMPLE1,SAMPLE2,COR)	Defines correlation between two samplings. See example below.	none
setSeed(SEED)	Specifies the seed for the random number generator.	none
setRNG(RNG)	Specifies the random number generator that will be used. Possible values are -1 (FREE), 0 (SYSTEM), 1 (LINDO1), 2 (LINDO2), 3 (LIN1), 4 (MULT1), 5 (MULT2), and 6 (MERSENNE).	none

The following tables list the available continuous and discrete distributions, respectively. Note that the parameter SAMSIZE must be specified and describes the size of the sample. However, the parameter VARRED is optional and facilitates choosing a variance reduction method. The values are 0 (meaning "none"), 1 (meaning "Latin Hyper Square") and 2 (meaning "Antithetic"). The default is Latin Hyper Square sampling, it will be used if no variance reduction method is specified.

Continuous Distribution	Description
beta(SHAPE_1,SHAPE_2,SAMSIZE[,VARRED])	**Beta distribution** specified by two shape parameters.
cauchy(LOCATION,SCALE,SAMSIZE[,VARRED])	**Cauchy distribution** specified by the location and the scale parameter.
chisquare(DF,SAMSIZE[,VARRED])	**Chi-Squared** distribution specified by degrees of freedom.
exponential(RATE,SAMSIZE[,VARRED])	**Exponential distribution** specified by rate of change.

Continuous Distribution	Description
f(DF_1,DF_2,SAMSIZE[,VARRED])	**F distribution** specified by degrees of freedom. Note that the function sampleLSf uses another version of the F distribution than the function dF from the Stochastic Library.
gamma(SHAPE,SCALE,SAMSIZE[,VARRED])	**Gamma distribution** specified by shape and scale parameter. Note that the function sampleLSgamma(A,B) is equivalent to the function dGamma(B,A) from the Stochastic Library.
gumbel(LOCATION,SCALE,SAMSIZE[,VARRED])	**Gumbel distribution** specified by location and scale parameter.
laplace(LOCATION,SCALE,SAMSIZE[,VARRED])	**Laplace distribution** specified by location and scale parameter.
logistic(LOCATION,SCALE,SAMSIZE[,VARRED])	**Logistic distribution** specified by location and scale parameter.
lognormal(LOCATION,SCALE,SAMSIZE[,VARRED])	**Log Normal distribution** specified by location and scale parameter.
normal(MEAN,STD_DEV,SAMSIZE[,VARRED])	**Normal distribution** specified by given mean and standard deviation.
pareto(SCALE,SHAPE,SAMSIZE[,VARRED])	**Pareto distribution** specified by shape and scale parameter.
studentt(DF,SAMSIZE[,VARRED])	**Student's t-distribution** specified by degrees of freedom.
triangular(LOW,MID,HIGH,SAMSIZE[,VARRED])	**Triangular distribution** specified by lower and upper limit and mid value.
uniform(LOW,HIGH,SAMSIZE[,VARRED])	**Uniform distribution** specified by the given bounds.
weibull(SCALE,SHAPE,SAMSIZE[,VARRED])	**Weibull distribution** specified by scale and shape parameter. Note that the function sampleLSweibull(A,B) is equivalent to the function dWeibull(B,A) from the Stochastic Library.

Discrete Distribution	Description
binomial(N,P,SAMSIZE[,VARRED])	**Binomial distribution** specified by number of trials N and success probability P in each trial.
hypergeo(TOTAL,GOOD,TRIALS,SAMSIZE[,VARRED])	**Hypergeometric distribution** specified by total number of elements, number of good elements, and number of trials.
logarithmic(P-FACTOR,SAMSIZE[,VARRED])	**Logarithmic distribution** specified by P-Factor. Note that the function sampleLSlogarithmic uses another version of the logarithmic distribution than dLogarithmic from the Stochastic Library.
negbinomial(SUCC,P,SAMSIZE[,VARRED])	**Negative Binomial distribution** specified by the number of successes and the probability of success. The generated random number describes the number of failures until the defined number of successes is reached. Note that the function sampleLSnegbinomial(R,P) is equivalent to the function dNegBinomial(R,P-1) from the Stochastic Library.
poisson(MEAN,SAMSIZE[,VARRED])	**Poisson distribution** specified by mean.

The following example illustrates the use of the sample generator and shows the effect of the functions `setCorrelation` and `induceCorrelation`:

```
$funcLibIn lsalib lsadclib

Functions normalSample   / lsalib.SampleLSnormal    /
          getSampleVal   / lsalib.getSampleValues   /
          setCor         / lsalib.setCorrelation    /
          indCor         / lsalib.induceCorrelation /;

Scalars d "dummy"
        h "handle for first sample"
        k "handle for second sample" ;

Set i "sample index" / i01*i12 /;

Parameters sv_h(i)  "sample values for handle h"
           sv_k(i)  "sample values for handle k";

* generate two handles for 12 samples from normal distribution with mean 5 and std.dev. 2 each
h = normalSample(5,2,12);
k = normalSample(5,2,12);

* retrieve sample values from Lindo library
loop(i, sv_h(i) = getSampleVal(h) );
loop(i, sv_k(i) = getSampleVal(k) );

display sv_h, sv_k;

* set and induce a correlation between samples h and k
d = setCor(h,k,-1);
d = indCor(1);

* retrieve sample values again from correlated distribution
loop(i, sv_h(i) = getSampleVal(h) );
loop(i, sv_k(i) = getSampleVal(k) );

display sv_h, sv_k;
```

The resulting output shows that the values of `sv_k` are reordered according to the desired correlation:

```
----     25 PARAMETER sv_h  sample values for handle h

i01 2.079,    i02 6.454,    i03 4.437,    i04 2.747,    i05 5.339,    i06 4.059,    i07 6.311,    i08


----     25 PARAMETER sv_k  sample values for handle k

i01 5.509,    i02 3.021,    i03 7.550,    i04 6.002,    i05 4.227,    i06 0.704,    i07 3.890,    i08


----     35 PARAMETER sv_h  sample values for handle h

i01 2.079,    i02 6.454,    i03 4.437,    i04 2.747,    i05 5.339,    i06 4.059,    i07 6.311,    i08
```

```
----     35 PARAMETER sv_k  sample values for handle k

i01 7.550,    i02 3.021,    i03 9.474,    i04 6.442,    i05 4.592,    i06 0.704,    i07 3.890,    i08
```

4.47.3.5 The Mutex Library

The Mutex Library allows us to work with a process-level *mutex* - a program object that can be used to synchronize multiple processes. For example, a mutex can be used to coordinate *mutually exclusive* access to a resource shared between processes and, thus, prevent multiple processes from accessing the shared resource at the same time. A mutex object is created with a unique name and can be acquired or locked by only one process at a time. Each process waits to acquire the lock for (i.e. ownership of) the mutex object before executing the code that accesses the shared resource. After accessing the shared resource, the process releases or unlocks the mutex object and other processes waiting for the mutex can acquire the lock and continue execution.

The Mutex Library may be used to prevent simultaneous access to shared resources when running GAMS programs concurrently and the errors and hard-to-explain behavior that result. Such behavior is not unusual or surprising: GAMS even has some language features to spawn programs asynchronously (e.g. Asynchronous Execution), and these concurrent GAMS programs often share files. Consider the following example where we have two GAMS programs A and B concurrently running a common code that writes a GDX file containing a scalar `x`:

```
scalar x;
repeat
  execute_load 'x.gdx', x;
  x = x + 1;
  execute_unload 'x.gdx', x;
until x > 100;
```

To ensure correct execution of the code, we want to ensure that programs A and B have mutually exclusive access to the GDX file. In addition, we want the load-increment-unload sequence to occur atomically (i.e. no other process reads or writes the GDX file during this sequence). The extrinsic function library `mtxcc` provides us with some functions to accomplish this and similar tasks:

- `Create(x)`: Creates a mutex with id `x` in the system. Returns `0` on success and `UNDF` in case of error.
- `Lock(x)`: Request to aquire or lock a mutex with id `x`: it returns immediately if the mutex is available, otherwise it waits until the mutex becomes available. Returns `0` on success and `UNDF` in case of error. Any process waiting in this call will wait forever if the mutex is never unlocked.
- `Unlock(x)`: Unlocks or releases a previously locked mutex with id `x`. Returns `0` on success and `UNDF` in case of error. If processes are waiting to acquire a lock on `x`, one of them will acquire the lock. Note that the lock should be held by the unlocking process: behavior is undefined otherwise.
- `TryLock(x)`: Like `Lock`, but if the mutex is not available, return `1` immediately instead of waiting.
- `TimedLock(x,y)`: Like `Lock`, but instead of waiting forever, return `1` after waiting unsuccessfully for `y` milliseconds.
- `Delete(x)`: Erases a mutex with id `x` from the system. Returns `0` on success and `UNDF` in case of error. Note that a process does not need to own the mutex to erase it. If a mutex is deleted and there are remaining processes queued up waiting for the unlock, these processes will wait forever.

Continuing our previous example, we need to make the functions `Lock` and `Unlock` available so we can use them to protect the entire loop body by enclosing it in `Lock`/`Unlock` calls.

```
$funcLibIn mtxlib mtxcclib
Function lock        / mtxlib.Lock /
         unlock      / mtxlib.Unlock /;
scalar x;
repeat
  abort$lock(12345) 'Problems locking mutex';
  execute_load 'x.gdx', x;
  x = x + 1;
  execute_unload 'x.gdx', x;
  abort$unlock(12345) 'Problems unlocking mutex';
until x > 100;
```

Before we can use the mutex, we first have to create it. Also if both programs are done, we need to make sure that the named mutex is erased at the end of the programs, otherwise named mutexes collect in your system and consume resources (memory or disk depending on the implementation). Because the deletion of a named mutex might be forgotten or a program exits while holding the lock to the mutex or another concurrent system uses the same id, it is a good idea to find a unique mutex name, create it once, and erase it at the end as shown in the following complete example:

```
* generate a unique mutex ID
$eval mtxID round(frac(jnow)*24*60*60*1000)

$onEcho > x.gms
$funcLibIn mtxlib mtxcclib
Function lock        / mtxlib.Lock /
         unlock      / mtxlib.Unlock /;
scalar x;
repeat
  abort$lock(%mtxID%) 'Problems locking mutex';
  execute_load 'x.gdx', x;
  x = x + 1;
  put_utility 'log' / 'Program %prgID% increased x to ' x:0:0;
  execute_unload 'x.gdx', x;
  abort$unlock(%mtxID%) 'Problems unlocking mutex';
until x >= 100;
$offEcho
* Initialize x.gdx
scalar x /1/;
$gdxOut x.gdx
$unload x
$gdxOut

$onEcho > create.gms
$funcLibIn mtxlib mtxcclib
Function create / mtxlib.Create /;
abort$create(%mtxID%) 'problems creating mutex';
$offEcho
* Create the mutex
$call.checkErrorLevel gams create.gms lo=2

* Asynchronously spawn A and B
$call.async 'gams x.gms --prgID=A fileStem=A'
$eval jhA JobHandle
$call.async 'gams x.gms --prgID=B fileStem=B'
$eval jhB JobHandle

* Wait until both jobs are back
```

```
$set statA 0
$set statB 0
$label l1
$eval x sleep(1)
$if not %statA% == 2 $eval statA JobStatus(%jhA%)
$if not %statB% == 2 $eval statB JobStatus(%jhB%)
$if not %statA%%statB% == 22 $goto l1

$onEcho > erase.gms
$funcLibIn mtxlib mtxcclib
Function delete / mtxlib.Delete /;
abort$delete(%mtxID%) 'problems erasing mutex';
$offEcho
* Erase the mutex
$call.checkErrorLevel gams erase.gms lo=2
```

The log indicates which program updated `x` and the GDX file `x.gdx` at what stage:

```
Program B increased x to 20
Program B increased x to 21
Program A increased x to 22
Program A increased x to 23
```

Models **dicegrid** and **asyncincbi** show more realistic uses for the `mtxcc` library.

4.47.4 Build Your Own Library

This section discusses the creation of a custom extrinsic function library. Before attempting to implement such a library, we suggest to study the example libraries for which source code and test models are available. These libraries are studied below.

Attention

> Building extrinsic function libraries requires the knowledge of a regular programming language (like C/C++, Fortran, ...) and experience with handling compilers and linkers to build dynamically linked libraries. In some situation, it may be easier to use the simpler GAMS macros or batinclude files to define own functions.

Note

> Extrinsic functions are limited to 20 scalar arguments and return a scalar value.

An extrinsic function library consists of a specification part and a number of callbacks to evaluate the defined functions at an input point.

The specification part is implemented by a callback `querylibrary`. It returns information about the library itself, available functions, their arguments, endogenous classification, etc. to the GAMS execution system. C, Delphi, or Fortran source code for this callback can be generated automatically by using the Python helper script `ql.py`. The script processes a specification file `*.spec`, which is specified as first argument. The format of this file is documented in the file `tri.spec`. Both `ql.py` and `tri.spec` are contained in the source of the trigonometric library examples in the GAMS test library, obtainable via

```
$ testlib trilib01
$ unzip trisource.zip
```

If an extrinsic function will be used within equations of a GAMS model, next to the function value evaluation callback, also callbacks that compute first and second derivatives with respect to all endogenous arguments at an input point should be provided. Occasionally, this can be inconvenient. Observe that GAMS can use the function values at points close to the input point to estimate the derivate values using finite differences. However, this method is not as accurate as analytic derivatives and requires a number of function evaluations, thus the convenience comes at a price. The attribute `MaxDerivative` in the specification of a function signals GAMS the highest derivatives this function will provide. For higher order derivatives, GAMS will use finite differences to approximate the derivative values. However, a better alternative is often the use of automatic differentiation when implementing the function evaluation. This is demonstrated in the CPP Library Example, see section Automatic Differentiation for more details.

GAMS offers some support to check the implementation of of derivatives for extrinsic functions via the function suffixes `grad`, `gradn`, `hess` and `hessn`. These function suffixes are defined for intrinsic and extrinsic functions. For example, for an extrinsic function `userfunc`, the gradient evaluation that the user implemented may be called with `userfunc.grad`. Further, an approximation of the gradient by finite differences is available by calling `userfunc.gradn`. Comparing the results of these two calls can often help to check the implementation of the gradient. The same principle applies for the Hessian and the function suffixes `.hess` and `.hessn`. The GAMS options FDDelta and FDOpt can be used to influence the finite difference calculations. For more details, see model [**DERIVTST**] in the GAMS Model Library.

4.47.4.1 Example: Trigonometric Library

The Trigonometric Library serves as an example of how to code and build an extrinsic function library. The library is included in the GAMS distribution in binary form. In addition, the source code in C, Delphi, and Fortran is available in the include files of the models [**TRILIB01**], [**TRILIB02**] and [**TRILIB03**] respectively. The library implements the following extrinsic functions:

Function	Description	End. Classif.
`setMode(MODE)`	Sets mode globally. Possible values are `0` for radian and `1` for degree. May be overwritten by the optional argument `MODE` in the functions `cosine` and `sine`.	none
`cosine(x[,MODE])`	Returns the cosine of the argument `x`. Note that the argument `MODE` is optional, default setting: `MODE = 0`.	NLP
`sine(x[,MODE])`	Returns the sine of the argument `x`. Note that the argument `MODE` is optional, default setting: `MODE = 0`.	NLP
`pi`	Value of $\pi = 3.141593...$	any

The C implementation of this extrinsic function library can be found in the files `tricclib.c` and `tricclibql.c`. Together with the API specification file `extrfunc.h`, these files document the callbacks that need to be implemented by a GAMS extrinsic function library. The file `tricclibql.c` implements the `querylibrary` callback, which provides information about the library itself and the extrinsic functions it implements. For example, the information that the function `cosine` has an endogenous required first argument and an exogenous optional second argument is available from the `querylibrary` callback. The file `tricclibql.c` (and also the Delphi and Fortran90 equivalents `tridclibql.inc` and `triifortlibql.f90`) has been generated by the script `ql.py` by processing the specification file `tri.spec`.

4.47.4.2 Example: The CPP Library

The CPP library serves both as an example of how to use C++ to obtain gradients and Hessians "for free" and as a source of functions based on the multivariate Normal distribution. The library is available in

compiled form and as C++ source. Test Library model [**CPPLIB00**] exercises the process of building a shared library from C++ source and doing some basic tests, while models [**CPPLIB01**], [**CPPLIB02**], [**CPPLIB03**], [**CPPLIB04**], and [**CPPLIB05**] are more thorough tests for the CPP library extrinsics shipped with the distribution. These functions are listed and described in the following table. Note that in keeping with the language conventions of statistics, *PDF* is shorthand for "probability density function" and *CDF* is shorthand for "cumulative distribution function".

Function	Description	End. Classif.
`pdfUVN(x)`	PDF of uni-variate Normal distribution, see MathWorld or R	NLP
`cdfUVN(x)`	CDF of uni-variate Normal distribution, see MathWorld or R	NLP
`pdfBVN(x,y,r)`	PDF of bivariate Normal distribution, see MathWorld or R	NLP
`cdfBVN(x,y,r)`	CDF of bivariate Normal distribution, see MathWorld or R	NLP
`pdfTVN(x,y,z,r21,r31,r32)`	PDF of trivariate Normal distribution, see MathWorldor R	NLP

Automatic Differentiation

Often, extrinsic functions are created in order to be used with endogenous arguments. In such cases it is necessary to provide first and second derivatives with respect to these arguments in addition to the function values themselves. One way to compute these derivatives is via automatic differentiation techniques (see the article in Wikipedia for details).

Note that with C++ it is possible to overload the usual arithmetic operators (assignment, addition, multiplication, etc.) so that automatic differentiation occurs with little or no change to the function-only source code. This is the technique used to compute the derivatives in the CPP Library. Observe that the Test Library model [**CPPLIB00**] includes all the source code for the CPP Library and illustrates the steps needed to build the library from this source. The source is a working self-documentation of how the process of automatic differentiation works.

Multi-Variate Normal Distributions

As shown in the table above, the CPP Library implements the PDF and CDF for the univariate, bivariate and trivariate standard Normal distributions. We use the standard Normal (mean of 0, standard deviation of 1) since intrinsic functions are limited to 20 arguments. The functions for the univariate case are included as convenient examples and should give results (nearly) identical to the functions pdfNormal and cdfNormal from the stochastic library. For the multivariate cases, the implementation is based on TVPACK from Alan Genz, with some modifications to allow for proper computation of derivatives. Note that we chose to implement the functions taking correlation coefficients as arguments, not a covariance matrix. The conversion from a covariance matrix to correlation coefficients is straightforward. The following `R` code describes this conversion. Here the package `mnormt` is used that computes the multivariate CDF with similar code to Genz:

```
# start with a mean mu and variance-covariance matrix S1
x <- c(1.0,3.0,3.0)
mu <- c(0.0,1.0,-1.0)
S1 <- matrix(c(1.0,1,1.5, 1,4,1.5,  1.5,1.5,9),3,3)
v1 <- pmnorm(x=x, mean=mu, varcov=S1)
# convert to std normal with 3 correlation coeffs
R <- cov2cor(S1)
sd <- sqrt(diag(S1))
xn <- (x-mu) / sd
v2 <- pmnorm(x=xn, mean=0, varcov=R)
```

Note that for the bivariate case there is one correlation coefficient `r`. The CDF implementation is not quite accurate to machine precision, it has 14 or 15 digits of accuracy. The trivariate case includes 3

correlation coefficients, the 3 off-diagonal elements from the lower triange of `R` above. The accuracy of the CDF depends on the inputs: it is higher when the correlation is nearly zero and lower as the condition number of `R` increases. Typically, an accuracy of 10^{-11} is all that can be achieved. In both multivariate cases, we recommend to avoid evaluating at or near points where the correlation matrix is degenerate. At nearly degenerate points, the accuracy of the distribution and density functions suffers. As the correlation matrix becomes degenerate, the distribution becomes degenerate too.

4.47.4.3 Remark: Stateful Libraries

While GAMS intrinsic functions are *stateless*, users may implement `stateful` extrinsic functions, i.e., functions that have some memory. There are two ways to achieve stateful extrinsic functions:

1. Library initialization (`LibInit`): at initialization time, the function library reads some data that is required to evaluate the provided functions. An example is the Piecewise Polynomial Library.
2. Previous function calls: function calls that alter the execution of successive function calls. An example is the function `SetMode` from the Trigonometric Library.

Attention

> Altering the state of an extrinsic function library by function evaluation calls is problematic, since different parts of the GAMS system potentially use different instances of the function library.

For example, consider that the function `SetMode` of the Trigonometric Library is called via `SetMode(1)` before a solve statement. Unless option solvelink is set to 5, the solver will run in a separate process with a new instance of the function library and therefore will use the default `mode`, which is zero. Further, if `solvelink` is set to zero, the GAMS process will terminate in order to execute the solve statement and will restart a new GAMS process after the solve. The restarted GAMS process will load a fresh instance of the extrinsic function library, which has no memory of the value of `mode` from before the solve statement. This problem is demonstrated in the GAMS Test Library model [**TRILIB04**].

4.47.5 Extrinsic Functions vs. External Equations

In addition to extrinsic functions, GAMS offers another facility to include additional mathematical functions in GAMS: external equations. These equations are denoted by equation type `=x=`. A feasible solution for a model instance must satisfy all internal and external equations. External equations are introduced and discussed in chapter External Equations.

Similar to extrinsic functions, it is the users responsibility to provide routines that evaluates the external equation. Further, both facilities are especially pertinent to nonlinear models.

An overview of some characteristics of extrinsic functions and external equations is given in the following table:

Characteristic	Extrinsic Function	External Equation
Maximum number of arguments	20	No limit
Available in statements	Yes	No
Debugging support	Yes	No
Returns Hessian to solver	Yes	No

Consider the following example, which demonstrates how an equation $z = \int_{x_1}^{x_2} x\,dx$ may be formulated using either an extrinsic function and an external equation:

```
* using an extrinsic function
e1.. integralx(x1,x2) =e= z;

* using an external equation
e1.. 1*x1 + 2*x2 + 3*z =x= 1;
```

Note that the function `integralx` is assumed to be a user-defined extrinsic function, probably implemented as $\texttt{integralx}(x_1, x_2) = \frac{1}{2}(x_2^2 - x_1^2)$.

Note further, that the integers 1 to 3 in the external equation are mapped to indices for variables that are used in external functions inside an external module. The right-hand side of the external equation denotes the external equation number. It is assumed that the user-defined external equation implements the function $(x_1, x_2, z) \mapsto \int_{x_1}^{x_2} x\,dx - z$.

As a simple example for the use of an extrinsic function in a statement, consider the following assignment, which sets the level of variable `y` to the value of $(\int_0^2 x\,dx)^2$:

```
y.l = sqr(integralx(0,2));
```

4.48 External Equations

GAMS provides a number of built-in or intrinsic functions for use in equations. Still, the extremely diverse set of application areas in which GAMS is used can create demand for the addition of new and often sophisticated and specialized functions. There is a trade-off between satisfying these requests and avoiding complexity not needed by most users. The GAMS External Equations Facility provides one means for managing this trade-off, since it allows users to import functions from an external library to define equations in a GAMS model. However, these external libraries can currently only provide functionality for the evaluation of functions (incl. their first derivatives) in a point. Solvers that need to analyze the algebraic structure of the model instance are therefore not able to work with external equations. This includes the class of *deterministic* global solvers, see column "Global" in this table, while, for example, *stochastic* global solvers can work with external equations.

Note

Both external equations and extrinsic functions aim to provide possibilities to extend GAMS by user-provided mathematical functions. However, there are fundamental differences in the use and implementation of both. For most situations, extrinsic functions should be preferred over external equations. See also Extrinsic Functions vs. External Equations.

Attention

Building external equation libraries requires the knowledge of a regular programming language (like C/C++, FORTRAN, ...) and experience with handling compilers and linkers to build dynamically linked libraries.

Note

The external equation interface is not intended as a way to bypass some of the very useful model checking done by GAMS for models that are solved with NLP solvers. External equations are still assumed to be continuous with accurate and smooth first derivatives. The continuity assumption implies that external equations must have very low noise levels, considerably below the feasibility tolerance used by the solver. The assumption about accurate derivatives implies that derivatives must be computed more accurately than can be done with standard finite differences. If these assumptions are not satisfied, then there is no guarantee that the NLP solver can find a solution that has the mathematical properties of a local optimum, i.e., a solution that satisfies the Karush-Kuhn-Tucker conditions within the standard tolerances used by the solver.

In the following, connecting code written in FORTRAN, C, Delphi, or some other programming language to equations and variables in a GAMS model is described. These GAMS equations will be referred to as *external equations* and the compiled version of the programming routines will be referred to as the *external module* that defines the external functions. The form of the external module depends on the operating system that is used. The external module under Windows is a Dynamic Link Library (`.dll`) and the external module under Unix is a shared object (`.so` or `.dylib`). In principle, any language or system may be used to build the DLL or shared object that defines the external module, as long as the interface conventions are not changed.

The GAMS Test Library provides examples of external equations consisting of GAMS models and C, Delphi, Java, and FORTRAN code. For more details, see Section Examples in the GAMS Test Library.

The basic mechanism of external equations is to declare all the equations and variables using the usual GAMS syntax. The interpretation of the external equations is done in a special way. Instead of the usual semantic content, the external equations specify the *mapping* between the equation and variable names used in GAMS and the function and variable indices used in the external module. This mapping is described in Section Model Interface. The external module may be written in C, FORTRAN, or most other programming languages. Section Programming Interface describes the general definitions for an external module for C, Delphi, and FORTRAN from a programming language perspective. Note that the way the program is compiled and converted into an external module is system and compiler specific. The following Section Implementation gives detailed advice on various aspects of the implementation of the external module.

4.48.1 Examples in the GAMS Test Library

Model **[TESTEXEQ]** gives an overview of all examples in the GAMS Test Library and may be used to compile and run them. Note that the remainder of this chapter will reference examples that are listed in this model. Further, model **[COMPLINK]** may be used as a script to compile and link external equation libraries. Note that these models hardcore the path to the Java compiler and libraries and these paths will need to be adapted by the user when running the Java examples.

Observe that regardless of how external libraries are built, the examples (e.g. **[EX1]**) will by default solve a model without using external equations. To solve the example models with all kinds of different external equation libraries, they may be run with the argument

```
--runall=1
```

Alternatively, only selected libraries may be used by using one or more of the following command line parameters:

```
--runC=1
--runC_cb=1
--runD=1
--runD_cb=1
--runF=1
--runF_cb=1
--runJ=1
```

4.48.2 Model Interface

4.48.2.1 External Equation Syntax

External equations that are used to specify the interface to the external module are *declared* in GAMS like any other equation. The syntax for the external equation *definition* statement is as follows:

```
eqn_name(index_list)[$logical_condition(s)].. expression =x= expression ;
```

Note that the only difference to the usual equation definition is the use of the equation type `=x=`.

The equations defined by an external module are always interpreted as equality constraints with zero right-hand sides. Thus inequalities have to be converted to equalities by adding explicit slack variables, which will serve as additional external variables. A nonzero right-hand side need to be taken care of in the external equation implementation.

4.48.2.2 Mapping of external equations and variables to indices

Some mappings must be specified to link an external module to a GAMS model. External equations are assumed to be defined in terms of indices `i = 1...m`. These indices must be mapped to GAMS equation names. Similarly, the variables used inside the external functions are assumed to be defined in terms of indices `j = 1...n`. These indices must be mapped to GAMS variable names. Finally, the name of the external module must be specified. Note that GAMS solvers are typically designed for large models and rely on sparsity. The last part of the specification of a set of external equations is therefore the sparsity pattern of the external equations, i.e., which variables appear in which equations.

The value of the *constant* term of the external equation must be an integer, since the value of the constant maps the row of the GAMS equation to the index (in `1...m`) of the external equation. Several blocks of GAMS equations may be mapped to external equations using the `=x=` notation. The mapping between GAMS equations of type `=x=` and indices `1...m` must be bijective (one-to-one). This means that two GAMS equations may not be mapped into the same external equation index and that there may not be any holes in the list of external equation indices. Although there may be any number of blocks of GAMS external equations, they must all map into and be implemented by one single external module.

The *variable* part of each external equation defines both the sparsity pattern of the external equation and the mapping from GAMS variables to the indices of the external variables. The variable part must be a sum of terms where each term is an integer times a variable. The existence of the term indicates that the variable involved is used in the external equation and that there is a corresponding derivative. The value of the coefficient defines the index of the external variable (in `1...n`) that the GAMS variable is mapped to. For example, the term `5*Y` indicates that the external equation depends on the GAMS variable `Y` and that `Y` is mapped to the 5th element in the vector of external variables. Clearly, if a variable appears in more than one external equation, then the value of its coefficient must be the same in each case.

Note that several blocks of GAMS variables may be used in external equations. In contrast to equations, where all rows in an equation block are either external or not, some columns in a variable block may be external while others are not. The mapping between GAMS variables that appear in equations of type `=x=` and external variable indices `1...n` must be bijective (one-to-one). This means that two GAMS columns may not be mapped into the same external variable index and that there may not be any holes in the list of external variable indices. Although there may be any number of blocks of GAMS variables mapped to external variables, they must all map into one single vector passed to the subroutine in the external module.

Observe that while some GAMS variables are external, there is no syntax provided to mark them as external variables. They may be used in non-external GAMS equations as well as external equations. Indeed, without this capability the model would be separable and the external equations and the functions they map to would be of little use.

Note

- As the coefficients and right-hand sides in the GAMS definition of external equations are interpreted as indices, users are not allowed to scale external equations and variables.
- External equations are treated in a special way, therefore the command line parameter and model attribute HoldFixed will not treat any fixed external variables as constants.

4.48.2.3 Name of external module

The name of the external module in which the external equations are implemented may be defined in a number of ways. By default, the external module is assumed to have the same name as the GAMS model with an extension that is operating system dependent. The extension is `.dll` for Windows, `.dylib` for macOS, and `.so` for any other Unix.

A custom name for the external module may be specified with a file statement. In this case the file name has to be listed as an additional item in the model statement. If the library extension is omitted in the file statement, GAMS will add the system-dependent extension automatically. This helps to make the model portable between different operating systems.

Consider the following simple example:

```
File  myextfile / extern /;
Model mymodel   / all, myextfile /;
```

When model `mymodel` is solved, GAMS will try to load the an external module file named `extern.so`, `extern.dylib`, or `extern.dll`, depending on the current operating system.

By default, the external module is assumed to be located in the directory `external_equations` in the GAMS standard locations or in the directory from which GAMS is called. A different location may be specified with an added path in the file statement.

4.48.3 Programming Interface

This section discusses C, Delphi, and FORTRAN interfaces to the GAMS external equations facility.

The external equation module need to provide a function called `GEFUNC`. The beginning of the external equation module typically looks as follows:

C:

```
#define GE_EXPORT
#include "geheader.h"
GE_API int GE_CALLCONV
gefunc(int* icntr, double* x, double* f, double* d, msgcb_t msgcb)
```

The header file `geheader.h` can be found in the `testlib_ml` subdirectory of the GAMS distribution. It defines `GE_API` and `GE_CALLCONV` and the signature of the function `gefunc`. `GE_API` is used to indicate to the compiler whether the function should be exported or imported. Due to defining `GE_EXPORT` before including `geheader.h`, `GE_API` is defined such that the function will be marked for export (`__declspec(dllexport)` on Windows and `__visibility__("default")` with GCC). Further, `GE_CALLCONV` indicates the calling convention that should be used on Windows. Currently, this is defined to be `__stdcall`. On other operating systems, it is empty.

FORTRAN:

```
      Integer Function gefunc (icntr, x, f, d, msgcb)
C Control Buffer:
      Integer icntr(*)
C Numerical Input and Output:
      Double Precision x(*), f, d(*)
C Message Callback Routine
      External msgcb
```

Delphi:

```
uses
    geheader_d;
Function GeFunc(var Icntr: ticntr;
                var x:     tarray;
                var F:     double;
                var D:     tarray;
                MsgFunc:   tMsgCallBack): integer; stdcall;
```

The unit file `geheader_d.pas` can be found in the `testlib_ml` subdirectory of the GAMS distribution.

In the following, the arguments of `GEFUNC` are described in detail.

4.48.3.1 Control vector icntr

The array `icntr` is a control vector that is used to pack and communicate control information between GAMS and the external module. Some helpful definitions to work with the `icntr` array are provided by the files `geheader.h` (C), `geheader_d.pas` (Delphi), and `gehelper.f90` (Fortran 90). The array elements are the following:

Element	Description
`icntr[I_Length]`	Holds the length of array `icntr` in number of elements. This is provided by GAMS.
`icntr[I_Neq]`	Number of external equation rows seen in the GAMS model. This is provided by GAMS.
`icntr[I_Nvar]`	Number of external variables seen in the GAMS model. This is provided by GAMS.
`icntr[I_Nz]`	Number of nonzero derivatives or Jacobian elements seen in the GAMS model. This is provided by GAMS.
`icntr[I_Mode]`	Current mode of operation. This is provided by GAMS. The following values are possible: `DOINIT`: Initialize. This will be the first call of `GEFUNC`, where initializations needed by the external module may be performed. `DOTERM`: Terminate. This will be the last call of `GEFUNC`, where cleanup tasks needed by the external module may be performed. `DOEVAL`: Function evaluation. External equations should be evaluated. `DOCONSTDERIV`: Constant Derivatives. Information about constant derivatives should be provided. DOHVPROD: Hessian-Vector product. The product between the Hessian of an external equation and a vector should be computed. See Second Derivatives: Hessian times Vector for details.
`icntr[I_Eqno]`	Index of the external equation to be evaluated during this call to `GEFUNC`. This is provided by GAMS in function evaluation mode (`icntr[I_Mode]=DOEVAL`) and is a number between 1 and icntr[I_Neq], inclusive. Note that the external equation interface allows to communicate information about only one function at a time.
`icntr[I_Dofunc]`	Flag whether function value should be computed. `icntr[I_Dofunc]` is provided by GAMS in function evaluation mode (`icntr[I_Mode]=DOEVAL`). If set to 1, then `GEFUNC` must return the numerical value of the function indexed by icntr[I_Eqno] in the scalar `f`.

Element	Description
icntr[I_Dodrv]	Flag whether derivative should be computed. icntr[I_Dodrv] is provided by GAMS in function evaluation mode (icntr[I_Mode]=DOEVAL). If set to 1, then GEFUNC must return the numerical values of the derivatives of the function indexed by icntr[I_Eqno] in the array d.
icntr[I_Newpt]	Flag for new point. icntr[I_Newpt] is provided by GAMS in function evaluation mode (icntr[I_Mode]=DOEVAL). If set to 1, then the point x may be different from the previous call of GEFUNC. If set to 0, then x will not have changed since the previous call.
icntr[I_Debug]	If icntr[I_Debug] is set to a nonzero value by the external equation module, then the functions GEstat and GElog will write all strings to a file called debugext.txt and flush the buffer immediately after writing. The string debugger may be used when a shared object crashes before GAMS has had an opportunity to display the messages. In FORTRAN, the string debugger will use FORTRAN unit icntr[I_Debug]. For more details see Section Message Output.
icntr[I_Getfil]	Flag to request the name of a special directory or file from GAMS. The following values are possible: I_Scr: Scratch directory, I_Wrk: Working directory, I_Sys: GAMS system directory, I_Cntr: Control file. For more information, see Section Communicating Data to the External Module via Files.
icntr[I_Smode]	Flag for string mode. This is provided by GAMS. For details see Section Communicating Data to the External Module via Files.
icntr[I_ConstDeriv]	Number of constant derivatives. Specifying this number during intialization is optional. For details see Section Constant Derivatives below.
icntr[I_HVProd]	Indicator for use of Hessian-Vector product for second order derivatives. This entry is optional. For details see Section Second Derivatives: Hessian times Vector below.

Observe that FORTRAN programmers will have to replace the square brackets [] with parentheses ().

4.48.3.2 Evaluation point x

Argument x is an array with icntr[I_Nvar] elements and is provided by GAMS if GEFUNC is called in function evaluation mode (icntr[I_Mode] = DOEVAL). Typically, GAMS and the solvers ensure that the individual elements of x are in between, or very close to, the variable bounds defined in the GAMS model. During initialization and termination calls, x is not defined and the external module must not reference x. C programmers should index this array starting at zero, i.e., the first external variable is referenced as x[0].

4.48.3.3 Function value f

If icntr[I_Mode] = DOEVAL and icntr[I_Dofunc] = 1, then the external module must return the value of the external equation icntr[I_Eqno] in the scalar f. During initialization and termination calls, f must not be referenced.

4.48.3.4 Derivative vector d

If `icntr[I_Mode] = DOEVAL` and `icntr[I_Dodrv] = 1`, then the external module must return the values of the derivatives of external function icntr[I_Eqno] with respect to all variables in the array `d`. The derivative with respect to variable `x[i]` is returned in `d[i]`. It is sufficient to set only those positions in `d` that correspond to variables actually appearing in equation icntr[I_Eqno]. Other positions are not being used by GAMS and may be left undefined. During initialization and termination calls, `d` must not be referenced.

4.48.3.5 Message callback msgcb

This argument is the address of a message callback routine that can be used to write messages to the status and/or log files of the GAMS process. Its type definition in C is as follows:

```
typedef void (GE_CALLCONV * msgcb_t) (const int* mode, const int* nchars, const char* buf, int len);
```

The argument `mode` is used to point to an integer which indicates where messages should be written to. This integer can be set to the following values:

- LOGFILE (1): Write the message to the log file only.
- STAFILE (2): Write the message to the status file only.
- LOGFILE | STAFILE (3): Write the message to both the log file and the status file. Observe that the symbol | denotes the bitwise logical `OR` in C.

The argument `nchars` points to an integer that specifies the number of bytes contained in the message (excluding the \0-terminator if there is one present). Thus, in C, `nchars` is typically set to `strlen(buf)`. The argument `buf` is a pointer to the character array containing the message to be printed. Finally, `len` is the size or length of the string `buf`, thus it is typically the same as `*nchars`.

Calling the message callback msgcb from C is straightforward. Note that the arguments `mode`, `nchars`, and `buf` are all call-by-reference and that addresses, not values, must be used. However, the argument `len` is call-by-value and `*nchars` should be passed as its value.

If the implementation is done in Delphi or Visual Basic, observe that pointers of all types are 4-byte quantities on a 32bit system and 8-byte quantities on a 64bit system. Integers are 4 bytes.

Calling this routine from a FORTRAN environment is a bit more complicated due to the different ways that FORTRAN compilers handle strings. The Unix convention - at least the convention observed on all systems for which GAMS is built - is that strings are passed by reference. In addition, the length of the string is passed by value as a hidden 4-byte quantity appended to the end of the argument list. This is the reason for including `len` as the last argument in `msgcb`. The argument `len` facilitates making FORTRAN callbacks in a Unix environment like the following:

```
character*(*) msgbuf
int nchars, charcount
nchars = charcount(msgbuf)
call MSGCB (mode, nchars, msgbuf)
```

Status Code	Definition

4.48.3.6 Return code

The function GEFUNC must return one of the following status codes:

Status Code	Definition
0	No error occurred.
1	A function evaluation error was encountered. GAMS should not use the content of f and/or d, but GEFUNC has recovered from the error and is ready to be called at a new point. This status code should only be used in function evaluation mode (icntr[I_Mode]=DOEVAL).
2	Fatal error. If this value is returned during the initialization call, then GAMS should abort immediately. It may be returned by GEFUNC during the initial call if some initializations did not work correctly, or if some of the size values in icntr had unexpected values. It may also be returned during function evaluation mode (icntr[I_Mode]=DOEVAL) if the external module has experienced problems from which it cannot recover.

4.48.4 Implementation

After describing the function GEFUNC in Section Programming Interface above, this section offers some practical comments on implementing GEFUNC.

4.48.4.1 Compiling and Linking

The examples for GAMS external equations contain a set of GAMS models for compiling the code on various systems using various compilers. Note that the compiler and linker flags shown in these examples should be used to ensure that the modules conform to the interface standard. In addition, the appropriate include file (geheader.h, geheader_d.pas, gehelper.f90) should be used.

4.48.4.2 Initialization Mode

The initialization mode should always check whether the external equations have the expected size: icntr[I_Neq], icntr[I_Nvar] and icntr[I_Nz] have to be tested against fixed expected values or values derived from some external data set.

The initialization mode may be used for several purposes like allocating memory and initializing numerical information or mapping information needed by the function evaluations that will follow. Data can be computed or read from external data sources or it can be derived from calls to an external database. Note that data that is shared with GAMS may be written to a file from GAMS using the put statement and then read in GEFUNC. Note further, that users must close the put file with a putclose statement before the solve statement. Observe that memory used to hold information from one invocation of GEFUNC to the next should be static. For FORTRAN it should either be in a Common block or it should be included in a Save statement.

4.48.4.3 Termination Mode

The termination mode may be used to perform some clean-up tasks like computing statistics, closing files, and returning memory.

4.48.4.4 Evaluation Mode

The bulk of the computational work will usually be in evaluation mode. Observe that `GEFUNC` only works with one equation at a time. One of the reasons for this choice is that the addressing of derivatives becomes very simple: there is one derivative for each variable and they have the same index in `d` and `x`, respectively.

In some applications several functions are naturally evaluated together, for example, because all functions are computed in some joint integration routine. The icntr[I_Newpt] flag is included for these applications. Using this flag, an implementation could evaluate all functions using a common routine when icntr[I_Newpt] equals 1 and saves the function and derivative values. Additionally, it returns the values corresponding to equation icntr[I_Eqno]. In subsequent calls to `GEFUNC`, `icntr[I_Newpt]` will likely be zero and the function and derivative values can quickly be extracted from the previously computed (and saved) information.

4.48.4.5 Evaluation Errors

It is good modeling practice to add bounds to the variables in such a way that all nonlinear functions are defined for all values of the variables within the bounds. Most solvers will also guarantee that nonlinear functions are called only when all entries of the vector `x` are between the bounds. However, it may not be practical to add all the necessary bounds and the implementation of `GEFUNC` should therefore capture evaluation errors such as division by zero, taking the logarithm of non-positive numbers, overflow in exponentiation, etc. If an equation cannot be evaluated at the given point, function `GEFUNC` should simply let the solver know about this situation by returning the value 1. The solver may then be able to backtrack to a safe point and continue the optimization from there.

Attention

> System-default or user-defined functions that handle evaluation errors (for example, the C library function `matherr()`) will sometimes not work in the same way inside a `DLL` or a shared object as they do in a self-contained program or a static library.

4.48.4.6 Message Output

External modules can send messages to the GAMS status file (usually the listing file) and the GAMS log file (usually the screen). Messages to be included in the GAMS status file can be buffered using the GEstat utility routine described below and messages to be included in the GAMS log file can be buffered using the GElog utility routine. Note that it is not possible to open these files for writing in the external module since GAMS or the solver process controls them.

Moreover, messages may be sent to both the status and log file without buffering, using the message callback msgcb. This removes the limit imposed by the size of the message buffer and may also make debugging somewhat simpler, since there is no need to worry about messages that never got flushed from the buffer. As it may be difficult or impossible to use the message callback from some environments, both the buffered and unbuffered techniques are provided.

Note that the two techniques for sending messages (buffered via `GEstat` and `GElog` and unbuffered via the message callback `msgcb`) are complementary. Either one or the other may be used, but if both are used in the same external module, the buffered messages will be printed after the unbuffered messages.

GEstat: The Utility Routine for Writing Messages to the Status File

GEstat is provided in the appropriate include file (Fortran 90: gehelper.f90, C: geheader.h, Delphi: geheader_d.pas). It is used to communicate messages that should be written to the GAMS status file. The function definition follows:

FORTRAN:
```
      subroutine gestat (icntr, line)
C Control Buffer:
      Integer icntr(*)
C input parameters:
      character*(*) line
```

C:
```
void GEstat(int* icntr, char* line)
```

Delphi:
```
Procedure GeStat(var icntr: ticntr; const s: shortstring);
```

Note that the first argument, icntr, must be passed through from the call of the function GEFUNC. The content of the argument line (or s in Delphi) is packed into the control buffer as one line. When GEFUNC returns, the content of the buffer will be written to the GAMS status file. GEstat may be called several times, each time with one line. Observe that line should not be longer than 132 characters and the overall amount of information written in one call to GEFUNC should not exceed 1000 characters. Further, line should not contain any special characters such as new-line or tab.

In practice, GEstat is often used with calls like the following:

FORTRAN:
```
call GESTAT (icntr, ' ')
call GESTAT (icntr, '**** External module based on abc.for')
```

C:
```
GEstat (icntr, " ")
GEstat (icntr, "**** External module based on abc.c")
```

Delphi:
```
gestat(icntr,' ');
gestat(icntr,'**** External module based on abc.dpr');
```

GElog: The Utility Routine for Writing Messages to the Log File

Like GEstat, GElog is provided in the appropriate include file. It is used to communicate messages that should be written to the GAMS log file. Note that by default, the log file is the screen. Alternatively, log may be written to a file that is specified with the GAMS command line parameter LogFile. The function definition of GElog follows.

FORTRAN:
```
      subroutine gelog( Icntr, line )
C Control Buffer:
      Integer Icntr(*)
C input parameters:
      character*(*) line
```

C:
```
void GElog(int* icntr, char* line)
```

Delphi:
```
Procedure GeLog(var icntr: ticntr; const s: shortstring);
```

Note that GElog behaves exactly like GEstat, with the status file replaced by the log file. The content of line is written to a buffer that in turn is written to the log file when GEFUNC returns.

Observe that it is not possible to write directly to the screen with some combinations of operating system and compiler. This may also depend on the options or flags that are used to build the external module.

Attention

> On some systems writing directly to the screen may cause the external module to crash. Therefore, it is advised not to write to the screen as a method for debugging, unless it is clear that it works. Otherwise the module may continue to crash because of the debugging statements after all other errors have been removed. Writing to a file and flushing the buffer is recommended as a safe alternative.

4.48.4.7 Communicating Data to the External Module via Files

Some external equations will need data from the GAMS program. This data may be passed on via one or more files written using put statements. Usually, such put files will be written in the current directory and the external module will look for them in the current directory. However, if users need to run multiple copies of the same model at the same time, data files should be written in the GAMS scratch directory and the external module should be directed to look for the data files in the scratch directory.

Note that a put file may be defined to be located in the scratch directory with the following file statement in the GAMS model:

```
File f / '%gams.scrdir%filename' /;
```

Observe that if the extension `.dat` is used, GAMS will remove the file from the scratch directory after the run. If another extension is used and the file is not deleted, GAMS will complain about an unexpected file when it cleans up after the run. The external module can receive the name of the scratch directory from GAMS during initialization by setting icntr[I_Getfil] to `I_scr` and returning immediately. GAMS will then store the name of the scratch directory and length of the name in the communication buffer and call `GEFUNC` in initialization mode again. Note that `GEFUNC` will now be called with the sub-mode icntr[I_Smode] set to `I_Scr`. Then the name may be extracted using the following FORTRAN call:

```
call GENAME( Icntr, Scrlen, Scrdir )
```

Here, `Scrdir` (declared as `character*255`) will receive the scratch directory and `Scrlen` (declared as integer) will receive the actual length of `Scrdir`. In C, the call takes the following form:

```
char scratchDir[255];
int scratchDirLen;
scratchDirLen = GEname(icntr, scratchDir, sizeof(scratchDir));
```

Here the routine will return the number or characters transferred to the buffer `scratchDir` if successful and the value -1 otherwise. If there is space, a terminating '\0'-byte will be written to `scratchDir`. If the value returned is equal to `sizeof(scratchDir)`, then the string returned will not be '\0'-terminated and may have been truncated as well.

Observe that it is possible to get other directory or file names by specifying other values in icntr[I_Getfil]. After setting this flag, `GEFUNC` must always return immediately.

For examples, see models **[EX5]** and **[EXMCP3]** and their respective FORTRAN and C source files.

4.48.4.8 Constant Derivatives

Some solvers, like the CONOPT solvers, can take advantage of the knowledge about constant derivatives in equations, which are a result of linear terms. This can be especially useful if an external equation represents an equation like `Y=f(X)`, where `Y` is unbounded, since variable `Y` can then be substituted by `f(X)`.

However, with the external module interface as described so far, the solver cannot know which variables appear linearly in the external equations. An optional extension allows to indicate that some of the relationships are linear. This can be activated by returning the number of constant derivatives in `icntr[I_ConstDeriv]` during the call of `GEFUNC` in initialization mode.

If the solver can use this information (not all solvers will), then `GEFUNC` will be called again repetitively with icntr[I_Mode] set to `DOCONSTDERIV`, once for each external equation, with its index specified as usual in icntr[I_Eqno]. For each of these calls, values of all constant derivatives must be specified in the array d. The remaining elements of `d`, both those corresponding to varying derivatives and to zeros, must be left untouched. These special calls will take place after the initialization call and before the first function evaluation call. Note that in these calls other flags like icntr[I_Dofunc] and icntr[I_Dodrv] and the array x will not be defined.

For an example, see model **[EX4X]** with the corresponding Fortran 90 and C source files `ex4xf_cb.f90` and `ex4xc_cb.c`, respectively. It is instructive to compare these files to the corresponding files without constant derivatives: `ex4f_cb.f90` and `ex4c_cb.c`.

4.48.4.9 Second Derivatives: Hessian times Vector

External modules cannot provide a solver with the Hessian matrix of external equations. However, some solvers have particular options for internally approximating the Hessian. For example, see the hessian_approximation option for IPOPT or hessopt for KNITRO. Further, the solver CONOPT can take advantage of second order information in the form of the product Hessian matrix $\nabla^2 f(x)$ times a vector v. This special form can be used for external equations by setting icntr[I_HVprod] to 1 during the call of `GEFUNC` in initialization mode.

If the solver can use this information (not all solvers will), then `GEFUNC` may be called with icntr[I_Mode] set to DOHVPROD to request this operation. icntr[I_Eqno] will hold the equation number and the array `x` will hold the values of the variables (x) in its first `icntr[I_NVar]` positions and a vector v in the following `icntr[I_Nvar]` positions. `GEFUNC` should evaluate and return $d = \nabla^2 f(x)\, v$ for the particular external equation f at the particular point x. Note that d (which is otherwise used for the derivative vector) will have been initialized to zero by GAMS.

$\nabla^2 f(x)\, v$ will often be needed for several vectors `v` at the same point `x`. Therefore, icntr[I_Newpt] will be used to indicate changes in `x` in the usual way.

Note that model **[EX1X]** with the corresponding Fortran 90 source file shows how to use both constant and second derivatives.

4.48.4.10 Debugging

Implementing external equations brings a number of new potential error sources which GAMS cannot protect against as well as with pure GAMS models. For example, the argument lists in the C or FORTRAN code may be incorrect or the linking process may create an incorrect external module. There is little GAMS can do to help users with this type of errors. It is recommended to carefully follow the examples and output debug messages during the setup calls, for example using the utility routines GEstat and GElog.

Once the overall setup is correct and GAMS can establish proper communication with the external module, there may still be numerical errors where the function values and the derivatives do not match.

Note that the solver CONOPT will by default call its Function and Derivative Debugger in the initial point, if a model has any external equations. The debugger will check that the functions only depend on the variables that are defined in the `sparsity` pattern and that derivatives computed by numerical perturbation are consistent with the derivatives computed by the external module. If an error is found, CONOPT will stop immediately with an appropriate message. For examples, see the GAMS Test Library models **[er1]**, **[er2]**, and **[er3]**, which illustrate different types of errors. The respective error messages will appear if CONOPT is used as the NLP solver. Note that comments about the errors may be found in the C or FORTRAN source code.

Observe that several types of errors cannot be detected. Derivatives that are computed in the external module and are returned in positions that were not defined in the `sparsity` pattern in GAMS will be filtered out by the interface and will therefore not be detected. Similarly, derivatives that should be computed but are forgotten, may inherit values from the same derivatives in another equation computed earlier. Finally, fixed variables cannot be perturbed, thus errors related to these variables will usually not be detected.

4.49 GAMS Return Codes

The most convenient way to embed a GAMS program in a different program (e.g. C#, Java, Python, ...) is the object orient API to GAMS. A much simpler but less powerful way is to make a call to the GAMS executable (`gams.exe` (Windows) or `gams` (Unix)) with appropriate parameters from your program. Different languages and operating systems have different ways of accomplishing such a task. A common way to communicate a small piece of information from the GAMS program to the caller program is the *exit status* or *return code* (see e.g. [https://en.wikipedia.org/wiki/Exit_status] (https://en.wikipedia.org/wiki/Exit_status)). GAMS return codes allow the caller to get some information about the status of the finished GAMS job. Note that return codes do not provide information about a model inside the GAMS job: the model may have been infeasible or may have failed in another way while the return code says all is fine. In fact, there may be multiple solves in a GAMS job, so even conceptually it is not possible to return solution status codes in the return code. The user cannot explicitly set the return code but can trigger action (e.g. `abort 'stop';`) that result in a specific return code (here *execution error* (3)).

Note

In general, the value of zero for return codes denotes success and non-zero values denote failure.

We first demonstrate how to access return codes from within GAMS with a self-explanatory example and then list all return codes. GAMS has the ability to call other programs via the `$call` (compile time) and `execute` (execution time) commands. Naturally, GAMS can call GAMS in a recursive fashion. There are different methods to access the return code of such a sub-GAMS job:

```
$onecho > x.gms
set a /1,2,3/;
$offecho

* Compile time access inside to return values

* Example for non-zero return code
$call gams x.gms lo=0 this_causes_a_parameter_error_with_return_code_6=1 > %system.nullfile%

* Check the return code via if [not] errorlevel n
$if not errorlevel 1 $abort expect a errorlevel >= 1

* Access the return code as function value of function errorLevel
$eval MYERRORLEVEL errorLevel
$log %MYERRORLEVEL%

* Example for zero return code
$call gams x.gms lo=0 > %system.nullfile%

* Check the return code via if [not] errorlevel n
$if errorlevel 1 $abort expect a errorlevel <= 0

* Access the return code as function value of function errorLevel
$eval MYERRORLEVEL errorLevel
$log %MYERRORLEVEL%

* Runtime

* Example for non-zero return code
execute 'gams x.gms lo=0 this_causes_a_parameter_error_with_return_code_6=1 > %system.nullfile%';
```

```
* Access the return code as function value of function errorLevel
scalar myerrorlevel;
myerrorlevel = errorlevel;
display 'should be 6:', myerrorlevel;

* Example for zero return code
execute 'gams x.gms lo=0 > %system.nullfile%';

* Access the return code as function value of function errorLevel
myerrorlevel = errorlevel;
display 'should be 0:', myerrorlevel;
```

Command line interpreters or shells are a powerful way for job control and can naturally also run GAMS jobs. Here are two examples that demonstrate how to access (actually echo) the return code from GAMS in such environments:

Unix shell (e.g. `bash`):

```
$ gams mymodel ...
$ echo $?
```

Here `$?` is the environment variable that stores the return code from the last run.

Windows (`cmd.exe`):

```
C:\tmp> gams mymodel ...
C:\tmp> echo %errorlevel%
```

Here `%errorlevel%` is the environment variable that stores the return code from the last run.

Note

On UNIX, return codes are treated modulo 256, so the return code 400 will be 144 on Unix. The return code modulo 256 is given in parenthesis in the table, if different from the return code.

4.49.1 List of the Error/Return Codes

The following table gives the list of the GAMS return codes:

Return Code	Description
0	Normal return
1	Solver is to be called, the system should never return this number
2	There was a compilation error
3	There was an execution error
4	System limits were reached
5	There was a file error
6	There was a parameter error
7	There was a licensing error
8	There was a GAMS system error
9	GAMS could not be started

Return Code	Description
10	Out of memory
11	Out of disk
109	Could not create process/scratch directory
110	Too many process/scratch directories
112	Could not delete the process/scratch directory
113	Could not write the script `gamsnext`
114	Could not write the `parameter` file
115	Could not read environment variable
400 (144)	Could not spawn the GAMS language compiler (`gamscmex`)
401 (145)	Current directory (`curdir`) does not exist
402 (146)	Cannot set current directory (`curdir`)
404 (148)	Blank in system directory (UNIX only)
405 (149)	Blank in current directory (UNIX only)
406 (150)	Blank in scratch extension (`scrext`)
407 (151)	Unexpected `cmexRC`
408 (152)	Could not find the process directory (`procdir`)
409 (153)	CMEX library not be found (experimental)
410 (154)	Entry point in CMEX library could not be found (experimental)
411 (155)	Blank in process directory (UNIX only)
412 (156)	Blank in scratch directory (UNIX only)
909 (141)	Cannot add path / unknown UNIX environment / cannot set environment variable
1000 (232)	Driver error: incorrect command line parameters for `gams`
2000 (208)	Driver error: internal error: cannot install interrupt handler
3000 (184)	Driver error: problems getting current directory
4000 (160)	Driver error: internal error: GAMS compile and execute module not found
5000 (126)	Driver error: internal error: cannot load option handling library

Note that error 3000 is sometimes caused by specifying the current directory in Microsoft UNC format. The return codes smaller than 100 come from the GAMS compiler and execution system (`gamscmex`) while the codes above 100 come from the GAMS driver program (`gams`).

4.50 GAMS Data eXchange (GDX)

This document describes the **GDX** (**G**AMS **D**ata e**X**change) facilities available in GAMS. The GDX facilities provide basic functionalities for exchanging GAMS data such as read and write. In addition to these facilities, there are a number of GDX Tools for exchanging data between GAMS and other data sources as well as for performing specific operations on a GDX file.

A GDX file is a file that stores the values of one or more GAMS symbols such as sets, parameters variables and equations. GDX files can be used to prepare data for a GAMS model, present results of a GAMS model, store results of the same model using different parameters etc. A GDX file does not store a model formulation or executable statements. Among other usages a GDX file can also be used to prepare data for a GAMS model, pass results of a GAMS model into different programs, and pass results into GAMS from different programs.

GDX files are binary files that are portable between different platforms. They are written using the byte ordering native to the hardware platform they are created on, but can be read on a platform using a different byte ordering. See also General notes on GDX files .

4.50.1 Reading a GDX file

Reading a GDX file into a GAMS model can be done during the compile phase or the execution phase.

4.50.1.1 Compile Phase

During compilation, we can use Dollar Control Options to read data from a GDX file. Reading during the compile phase also allows us to define the elements of a set and the subsequent use of such a set as a domain. For a complete list of available Dollar Control Options for reading data from a GDX file into GAMS during compilation of a GAMS model please refer to Dollar Control Options for GDX Operations.

In addition to the chapters with examples linked from the list you will find further examples below.

4.50.1.2 Execution Phase

When reading data from a GDX file during execution phase the data in the GDX file will be the data present in the GDX file at the time that the statement is executed. The results of all prior calculations and the most recent solve for any model will be reflected. The statements to read data from a GDX file during execution phase are:

- execute_load
- execute_loaddc
- execute_loadpoint
- gdxLoad

execute_load

The `execute_load` statement acts like an assignment statement, except that it does not merge the data read with the current data; it is a full replacement. The same restrictions apply as in an assignment statement: we cannot assign to a set that is used as a domain, or to a set used as a loop control.

Sets defining domains cannot be loaded. However sets that are subsets of existing sets and do not define new elements can be loaded at execution time (Domain defining sets can be loaded at compile time using $load).

The statement with `execute_load` can be used in one of the following forms:

```
execute_load 'file_name', id1[, id2, ..., idn] ;
execute_load 'file_name', id1=gdxid1[, id2=gdxid2, ..., idn=gdxidn] ;
execute_load 'file_name', setid=* ;
```

where

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension `.gdx`; read from the current working directory)
id1, id2, ..., idn	Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional
id1=gdxid1, id2=gdxid2, ..., id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2, ..., gdxidn in the GDX file; note that commas are optional

Note

- Items must be declared with Set, Parameter, Scalar, Variable or Equation statements before the `execute_load` appears.
- When loading data domain checking is not enforced so that when an item is resident in a GDX file for set elements not present in the current file these items are ignored and do not create errors or cause generation of any messages. The execute_loaddc variant checks to see that the domains match.

See [**load11**] and [**qp1x**] from the GAMS Model Library on how to use `execute_load`.

execute_loaddc

With `execute_loaddc` statement any domain violation will be reported and flagged as execution error. In contrast, the `execute_load` statement ignores all domain violations and loads only data that meets the domain restrictions. In addition to loading data for sets, parameters and variables, we can load a field of a variable into a parameter.

Warning: when loading a single field, all other fields are reset to their default value.

The statement with `execute_loaddc` can be used in one of the following forms:

```
execute_loaddc 'file_name', id1[, id2, ..., idn] ;
execute_loaddc 'file_name', id1=gdxid1[, id2=gdxid2, ..., idn=gdxidn] ;
```

where

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension `.gdx`; read from the current working directory)
id1, id2, ..., idn	Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional
id1=gdxid1, id2=gdxid2, ..., idn=gdxidn	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2, ..., gdxidn in the GDX file; note that commas are optional
setid=*	Allow to load the universe of labels from a GDX file into a set. Note, that only labels known to the GAMS program will be loaded.

See [**load7**] example on how to use `execute_loaddc`.

execute_loadpoint

The `execute_loadpoint` is similar to execute_load, however, the new values are merged with the old values.

If no arguments besides the name of the GDX file are given, all variables and equations that match variables and equations of the calling GAMS programs will be merged with the GDX level and marginal values. Bounds, scales and priorities will remain unchanged. Note that implicitAssign can be particularly useful in this context.

If one or more symbols to be loaded are specified explicitly, values for these symbols will be merged. For variables and equations only the levels and marginals will be merged. To merge other variable or equation attributes like bounds, scales and priorities, they have to be listed explicitly (e.g. `execute_loadpoint 'file_name', x.lo;` to merge the lower bound of variable `x`).

The statement with `execute_loadpoint` can be used in one of the following forms:

```
execute_loadpoint 'file_name'[, id1, id2, ..., idn] ;
execute_loadpoint 'file_name'[, id1=gdxid1, id2=gdxid2, ..., idn=gdxidn] ;
```

where

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension `.gdx`; read from the current working directory)
id1, id2, ..., idn	Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional
id1=gdxid1, id2=gdxid2	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 in the GDX file; note that commas are optional

gdxLoad

By default, `gdxLoad` does the same as execute_load. However, while `execute_load` always replaces existing data and loads data filtered (so records with domain violations are ignored), the behavior of `gdxLoad` can be changed using the options replace and filtered.

If no arguments besides the name of the GDX file are given, all symbols that match symbols of the calling GAMS programs will be loaded. Note that implicitAssign can be particularly useful in this context.

The statement with `gdxLoad` can be used in one of the following forms:

```
gdxLoad 'file_name'[, id1, id2, ..., idn] ;
gdxLoad 'file_name'[, id1=gdxid1, id2=gdxid2, ..., idn=gdxidn] ;
```

where

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension `.gdx`; read from the current working directory)
id1, id2, ..., idn	Read GAMS symbols id1, id2, ..., idn from the GDX file; note that commas are optional
id1=gdxid1, id2=gdxid2, ..., idn=gdxidn	Read GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2, ..., gdxidn in the GDX file; note that commas are optional
setid=∗	Allow to load the universe of labels from a GDX file into a set. Note, that only labels known to the GAMS program will be loaded.

4.50.1.3 Example 1 - Reading a GDX File

The file `trnsport.gms` (from [**TRNSPORT**]) has been modified to use the demand data from an external source. Only the relevant declarations are shown.

The parameter `B` is read from the GDX file using the name 'demand', and only those elements that are in the domain `j` will be used. Values for parameter `B` that are outside the domain `j` will be ignored without generating any error messages.

```
* Example 1
Set
   j    markets  / new-york, chicago, topeka / ;
Parameter
   B(j) demand at market j in cases ;
$gdxin demanddata.gdx
$load  b=demand
$gdxin
```

4.50.1.4 Example 2 - Reading a GDX File

In this example, the set `J` is also read from the GDX file, and is used as the domain for parameter `B`. All elements read for the set `J` will be used. Values for the parameter `B` that are outside the domain J will be ignored. Note that the dimension of set `J` is set to one by specifying its domain.

```
* Example 2
$gdxin demanddata.gdx
Set
   J(*)    markets;
$load j=markets
Parameter
   B(j) demand at market j in cases ;
$load  b=demand
$gdxin
```

4.50.1.5 Example 3 - Reading a GDX File

The `$load` command without parameters can read a listing of all symbols in a GDX file. The following:

```
* Example 3
$gdxin trnsport.gdx
$load
```

writes the following to the listing file:

```
Content of GDX C:\XLSFUN\TRNSPORT.GDX

Number Type      Dim  Count  Name

    1 Set         1      2  i         canning plants
    2 Set         1      3  j         markets
    3 Parameter   1      2  a         capacity of plant i in cases
    4 Parameter   1      3  b         demand at market j in cases
    5 Parameter   2      6  d         distance in thousands of miles
    6 Parameter   0      1  f         freight in dollars per case per thousand miles
    7 Parameter   2      6  c         transport cost in thousands of dollars per case
    8 Variable    2      6  x         shipment quantities in cases
    9 Variable    0      1  z         total transportation costs in thousands of dollars
   10 Equation    0      1  cost      define objective function
   11 Equation    1      2  supply    observe supply limit at plant i
   12 Equation    1      3  demand    satisfy demand at market j
```

which lists the items present by `Type`, `Name`, Number of sets the item is defined over(`Dim`), number of elements in the file for this item (`Count`).

4.50.1.6 Example 4 - Reading a GDX File

Sometimes, a set is implicitly given by the elements of a parameter symbol. For example,

```
parameter a(i) / seattle 350, san-diego 600 / ;
```

in `trnsport.gms` implicitly defines the set of plants `i`. GAMS does not allow us to provide domain checked data, if the data for domain sets is unknown. So this code produces a compilation error:

```
Set i plant;
Parameter a(i) capacity / seattle 350, san-diego 600 /;
```

When entering data directly in the GAMS source adding the domain sets before the actual parameter declarations is usually not a problem, but when data comes from external sources (e.g. spreadsheets, databases, etc), this often results in an additional query to the database, spreadsheet etc. Nowadays, such data exchange happens mostly via the GD facility. With the domain load capability of the compile time load instructions ($load, $loadDC, $loadR, $loadM, $loadDCM, and $loadDCR) one can project an index position from a parameter or set symbol in the GDX container and load this slice into a one dimensional set. Here is a simple example:

```
Set i plant;
Parameter a(i) capacity;
$gdxin data
$load i<adata a=adata
```

This will try to load set elements from the GDX parameter symbol `adata` into the set `i` and next load the GDX parameter `adata` into the GAMS parameter `a`. The latter one is no problem anymore, since the data for set `i` is known when loading symbol `a`. GAMS will use the domain information stored in GDX of parameter adata to identify the index position to project on. If no appropriate domain information can be found in GDX, the GAMS compiler will generate an error. In such case the user can explicitly select an index position (here first index position) from the GDX symbol:

```
$load i<adata.dim1 a=adata
```

The automatic index position matching (i.e. no .dimN) using the domain information stored in GDX matches on the name of the set to be loaded and the domain set names stored in GDX for the symbol. The domain in GDX are searched from right to left (start with n=symbol dimension, then n-1, n-2, ...) and stops at the first match. With the projection symbol <=, the domain in GDX is searched from left to right. This follows the style of the GAMS run time projection operation:

```
option sym1<sym2, sym1<=sym2;
```

Here is an example how to load. The network is defined by the capacity parameter cap contained in a GDX container `net.gdx`:

```
parameter cap(n,n) / (1*3).4 5, 4.(5*9) 3 /;
```

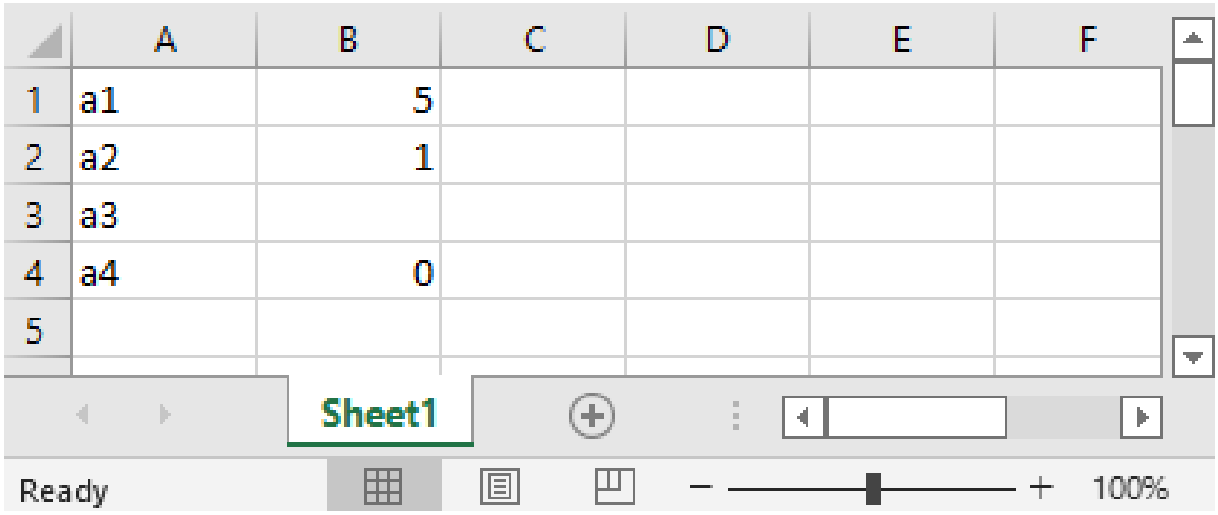
The following code loads the entire node set n of the network as well as the nodes with outgoing (`out`) and incoming (`in`) arcs and the capacity `c`.

```
set n nodes, out(n), in(n);
parameter c(n,n) capacity;
$gdxin net
$loadM n<=cap n<cap
$loadDC out<cap.dim1 in<cap.dim2 c=cap
display n, out, in;
```

The listing file looks as follows:

```
----      6 SET n  nodes
1,    2,    3,    4,    5,    6,    7,    8,    9

----      6 SET out  Domain loaded from cap position 1
1,    2,    3,    4

----      6 SET in  Domain loaded from cap position 2
4,    5,    6,    7,    8,    9
```

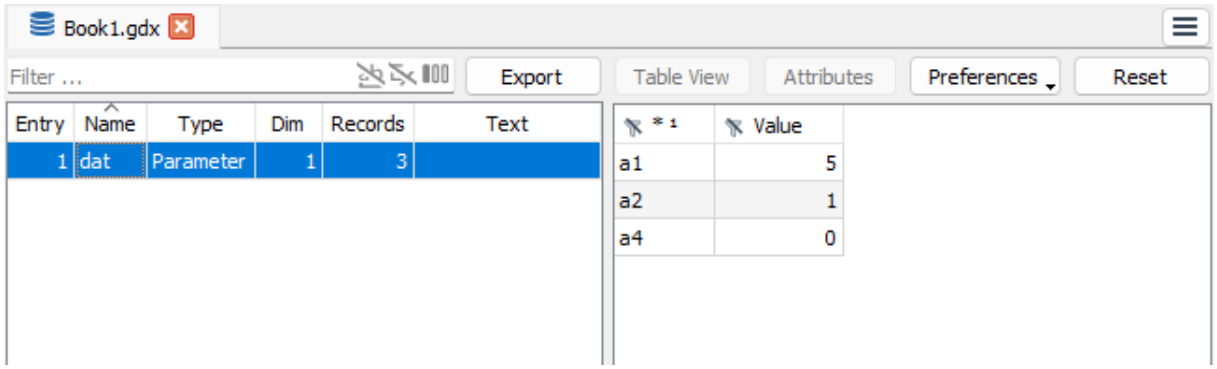
There is a potential issue with loading domains from parameters that have a zero value for some record. Since GAMS works with sparse data, it is sometime difficult to distinguish between a record with value zero (0) and the non-existence of a record. This is usually not a problem since we know the domain of a parameter and hence know all potential records. In case of using a parameter to define the domain this represents a source of confusion. Moreover, GDX has the capability of storing true zeros (most GDX tools like gdxxrw have options (Squeeze=Y or N) to either write a true 0 or squeeze the 0s when writing GDX). So in case GDX has a zero record, a domain load from such a parameter will include this record. Here is an example. The spreadsheet `Book1.xlsx` contains the following data:



The GDX utility GDXXRW with the following command line:

```
gdxxrw Book1.xlsx Squeeze=N par=dat rng=Sheet1!a1 rdim=1
```

reads the Excel data and produces a GDX container Book1.gdx with a one dimensional parameter dat(∗) which can be viewed in the GDX Viewer in GAMS Studio:



Notice that label a4 is present while label a3 is not part of GDX symbol dat. Without the Squeeze=N (the default is Squeeze=Y) we also would not have seen a4. If we load dat to define the domain (remember we need to use $load i<dat.dim1 since gdxxrw does not write domain information to GDX), we will miss out on a3 but have a4 (assuming `Squeeze=N`). Please also note that the zero record disappears on regular loading and is turned into an `EPS` when loading under `$OnEps`:

```
set i;
parameter a(i);
$gdxin Book1
$load i<dat.dim1 a=dat
display i,a;
parameter a0(i);
$OnEps
$load a0=dat
display a0;
```

This results in a listing file

```
----      5 SET i  Domain loaded from dat position 1
a1,    a2,    a4

----      5 PARAMETER a
a1 5.000,    a2 1.000

----      9 PARAMETER a0
a1 5.000,    a2 1.000,    a4   EPS
```

With gdxxrw parameter Squeeze=Y the listing file would look as follows:

```
----      5 SET i  Domain loaded from dat position 1
a1,    a2

----      5 PARAMETER a
a1 5.000,    a2 1.000

----      9 PARAMETER a0
a1 5.000,    a2 1.000
```

4.50.1.7 Example 5 - Reading a GDX File

The following statement reads gams element `k`, `d`, `f`, `a`, `b`, and `x` from file `trans2.gdx` during execution phase:

```
execute_loaddc 'tran2',k=j,d,f,a=sup,b=dem,x,supply;
```

where `k` be renamed from `j`, `a` be renamed from `sup`, and `b` be renamed from `dem` in the GDX file `tran2.gdx`.

Suppose there is one element `topeka` missing from the set definition but the element remains in the GDX file `tran2.gdx`. As a consequences the listing file contains an error message like:

```
**** GDX ERROR AT LINE 45 - Domain violation when loading from GDX file
**** 1 Domain errors for symbol k
     topeka

**** GDX ERROR AT LINE 45 - Domain violation when loading from GDX file
**** 2 Domain errors for symbol d
     seattle.topeka
     san-diego.topeka
```

and the job is aborted with an execution error.

Note

- domain errors occur whenever set element names are not spelled exactly the same as an element specified in the corresponding set in GAMS flagging alternative spellings or missing elements.
- domain errors do not arise when items are not specified with them set to zero (no entry for a set element leaves to a corresponding value of zero)

4.50.2 Writing a GDX file

Writing of GDX files in a GAMS model can be done during the compile phase or the execution phase. A GDX file can also be written after compilation and execution.

4.50.2.1 Compile Phase

During compilation, we can use a group of Dollar Control Options to write data to a GDX file. Writing during the compilation phase also allows us to define the elements of a set and the subsequent use of such a set as a domain. For a complete list of available Dollar Control Options for writing data to a GDX file during compilation of a GAMS model please refer to Dollar Control Options for GDX Operations.

In addition to the chapters with examples linked from the list you will find further examples below. Also see [**unload1**] - [**unload9**] and [**CompTimeWriteTrnsportGDX**] for more examples on how to use `$gdxOut` and `$unLoad`.

4.50.2.2 Execution Phase

When writing data to a GDX file during execution phase the data in the GDX file will be the data present in the GDX file at the time that the statement is executed. The results of all prior calculations and the most recent solve for any model will be reflected. The statements to write data to a GDX file during execution phase are

- execute_unload
- execute_unloaddi
- execute_unloadidx
- savepoint

execute_unload, execute_unloaddi, and execute_unloadidx

The `execute_unload` statement replaces an existing file with that name; it does not add symbols to or replace symbols in an existing GDX file. Without specifying any identifier, all sets, parameters, variables and equations will be written to the GDX file.

The `execute_unloaddi` statement replaces an existing file with that name; it does not add symbols to or replace symbols in an existing GDX file similar to execute_unload, but also writes the domains of all unloaded symbols to the same file.

The `execute_unloadidx` statement requires that each symbol written is a parameter; each parameter must have a domain specified for each index position. These domains have the requirement that they are formed using an integer sequence for the UELs that starts at 1 (one). The domain names are changed to indicate the size of each domain. This information is used when reading the data back from the GDX file using `$LoadIDX` during compilation. Using the special domain names, the UELs for the domains can be recovered without writing the domains to the GDX file; see example below.

The statement with `execute_unload` can be used in one of the following forms:

```
execute_unload  'file_name' , id1, id2, ..., idn ;
execute_unload  'file_name' , id1=gdxid1, id2=gdxid2, ... ;
execute_unload  'file_name' , setid=* ;
```

The statement with execute_unloaddi can be used in one of the following forms:

```
execute_unloaddi 'file_name' , id1, id2, ..., idn ;
execute_unloaddi 'file_name' , id1=gdxid1, id2=gdxid2, ... ;
execute_unloaddi 'file_name' , setid=* ;
```

The statement with execute_unloaddidx can be used in one of the following forms:

```
execute_unloadidx 'file_name' , id1, id2, ..., idn ;
execute_unloadidx 'file_name' , id1=gdxid1, id2=gdxid2, ... ;
execute_unloadidx 'file_name' , setid=* ;
```

where

Parameter(s)	Description
file_name	Specify the name of the GDX file (with or without the extension .gdx; written to from the current working directory)
id1, id2, ..., idn	Write GAMS symbols id1, id2, ..., idn into the GDX file
id1=gdxid1, id2=gdxid2	Write GAMS symbols id1, id2 with corresponding names gdxid1, gdxid2 into the GDX file

Note

- when only file_name is specified without other parameters all GAMS Symbols will be written into the GDX file file_name.
- The GAMS option gdxUELs controls which UELs are registered in file_name. With option gdxUELs = squeezed; (default) only the UELs that are required by the exported symbols are registered while all known UELs are registered if we set option gdxUELs = full;. See also [**unload10**].
- Variables and equations are always exported with all their attributes .l (level), .m (marginal), .lo (lower bound), .up (upper bound), .scale (scale factor). Specifying a suffix for variables or equations when calling execute_unload does not change this behavior. So the following two statements results in the same output:

  ```
  execute_unload 'dataL.gdx', x.L;
  execute_unload 'data.gdx', x;
  ```

See Example 2 - Writing a GDX file and [**qp1x**] on how to use execute_unload. See [**unload10**] on how to use execute_unloaddi. See Example 3 - Writing a GDX file on how to use execute_unloadidx.

Savepoint

A GDX file containing the marginals and levels for all variables and equations at the end of a solve will be created with the command line parameter, model attribute or option statement Savepoint. One can save the solution information from the last solve or from every solve. The points that are saved can be used to provide an advanced basis, integer program starting point or NLP starting point.

The basic command line form is:

```
gams mymodelname Savepoint=number
```

the model attribute form is

```
modelname.savepoint=number;
```

and the option statement form is

```
option savepoint=number
```

where

- when number equals 1/3 a point gdx file is saved from the last solution collected in the GAMS model and the file name will be `modelname_p.gdx` where model name is the name of the model identified in the solve statement.
- when number equals 2/4 a point gdx file is saved from every solve in the GAMS model and the file name will be `modelname_pnn.gdx` where model name is the name of the model identified in the solve statement and nn is the internal number of the solve. Thus if 10 solves occur one will get 10 files named `modelname_p1.gdx` through `modelname_p10.gdx`.

Note

When solving asynchronously, one should keep in mind, that the internal numbers mentioned for `savepoint=2/4` are generated when submitting the model, not when collecting it. And with `savepoint=1/3` it is the last solution collected, not the last model submitted that is written to the savepoint.

The following example:

```
model firm / all /;
firm.savepoint=1;;
solve firm using LP maximizing objfun;
```

saved a point gdx file `firm_p.gdx`.

and:

```
model transport /all/ ;
option savepoint=2;
set newseattle /s1,s2/;
parameter datador(newseattle) /s1 350, s2 450/;

loop(newseattle,
    a("seattle")=datador(newseattle);
    Solve transport using lp minimizing z ;
);
Display x.l, x.m ;
```

saved two point gdx files `transport_p1.gdx` and `transport_p2.gdx`.

4.50.2.3 Example 1 - Writing a GDX file

This example has modified the file `trnsport.gms` from [**TRNSPORT**] by adding the following statements after the last line.

```
...[TRNSPORT]...

d(i,j)=d(i,j)*10;

$gdxout tran
$unload i j
$unload d
$unload f
$unload b=dem a=sup
$gdxout
```

This example creates the GDX file `tran.gdx` containing the data for the sets `i` and `j` as well as the parameters `d`, `f`, `a` and `b` during the compile time, when `a` and `b` have been renamed in the GDX file to `dem` and `sup`. Also note the parameter `d` will not have been multiplied by 10 but rather take on their compile time value.

An `$unload` statement above can specify more than one symbol to be written to a GDX file and the similar result could also be accomplished using:

```
...[TRNSPORT]...

d(i,j)=d(i,j)*10;

$gdxout tran
$unload i j d f b=dem a=sup
$gdxout
```

4.50.2.4 Example 2 - Writing a GDX file

This example has modified from the file `trnsport.gms` (from [**TRNSPORT**]) by adding the following statement right after the solve statement.

```
...[TRNSPORT]...

Solve trnsport using LP minimizing Z;
execute_unload 'results.gdx', i, j, z, x;
```

After solving the model, the sets `i` and `j` and the variables `z` and `x` with all the data available after the solve.

4.50.2.5 Example 3 - Writing a GDX file

This example shows the use of the indexed write and read data during execution phase:

```
Set I /1*100/,
    J /1*50 /;
parameter A(I,J) /1.1=11, 1.9=19, 10.1=101/;

execute_unloadidx 'data.gdx', A;
```

Viewing the file `data.gdx` in GAMS Studio shows the modified domain information:

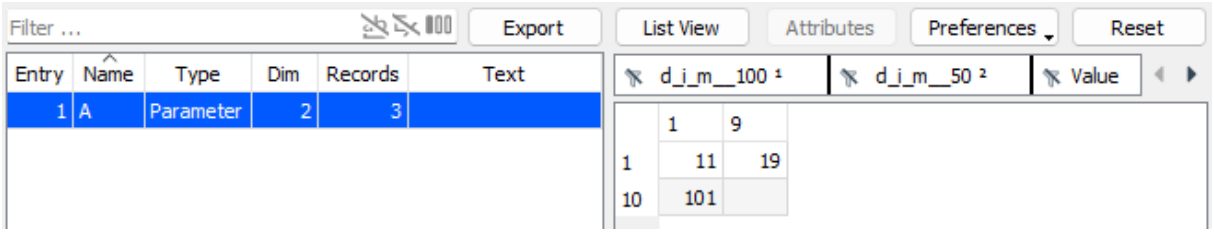


Figure 4.2 GAMS Studio showing data.gdx note the modified domains

To read from `data`.gdx, we use the indexed read:

```
Set I,J;
parameter A(I,J);
* load the data
$gdxin data.gdx
$loadidx A
$gdxin
* write all symbols so we can inspect in GAMS Studio
$gdxout test.gdx
$unload
$gdxout

execute_unloadidx 'data.gdx', A;
```

Viewing the file `test.gdx` in GAMS Studio shows that the domains have been populated:

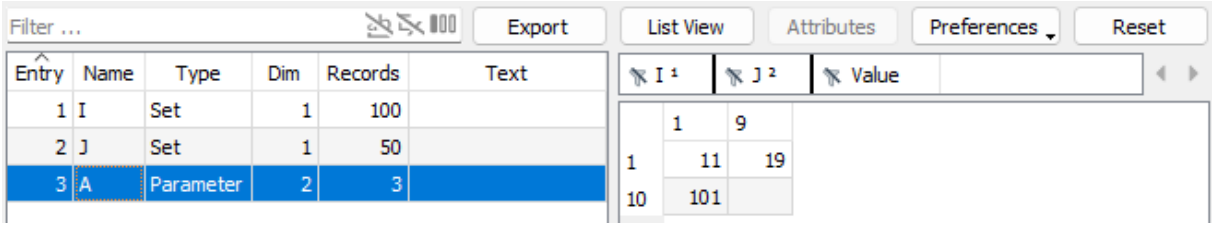


Figure 4.3 View test.gdx in GAMS Studio

4.50.2.6 Writing a GDX file after compilation or execution

A GDX file containing all data items resident at the end of the run of a GAMS code can be created using the gdx command line option either via GAMS call at the command line or via the GAMS Parameter Editor in GAMS Studio. This will cause all sets, parameters, variables and equations to be written to the GDX file.

For example:

```
gams mymodelname gdx=gdxfile_name
```

Or

```
gams mymodelname action=c gdx=gdxfile_name
```

where

- `mymodelname` specifies the name of the GAMS file
- `gdxfile_name` gives the file name and possible path where the GDX file is to be retained. When no path is specified the default directory is the current working directory where the main GAMS file associated with the project is executed via Studio.
- action=c indicates request to write a GDX file after compilation only
- setting GDX to the string "default" (i.e.`gdx=default`) causes GAMS to create a GDX file with the GAMS file root name and a GDX extension. Thus

  ```
  gams trnsport gdx=default
  ```

  will cause GAMS to write the gdx file `trnsport.gdx`.

When GAMS Studio is used, the GDX file creation can be invoked by running the main file with GDX creation.

Note

- When this option is used the GDX file is created just at the end of the GAMS execution so the data written will contain the current values for all sets, parameters, variables and equations that are on hand at the end of the GAMS job.
- The GDX data for the variables and equations contains the levels, marginals, lower bounds, upper bounds and scales for each item.
- This yields a file that is automatically opened in Studio.

Using the gdx command line parameter when running the model via Studio, the process log will show the GDX file name in green indicating that the file can be opened by clicking on the file link. See also Inspecting contents with GAMS Studio.

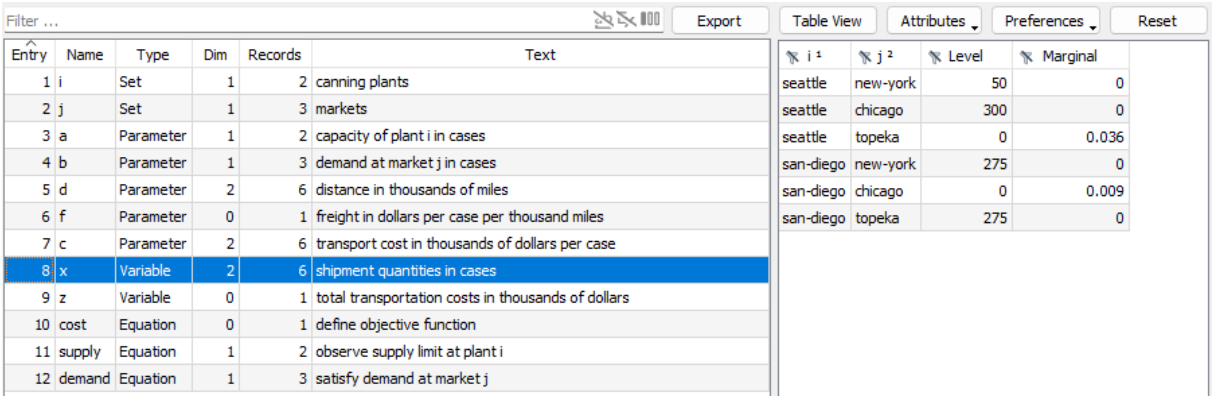
4.50.3 Inspecting contents of a GDX file

In addition to reading data from a GDX file during compile phase or execution phase there are a few ways to examine the contents of a GDX file.

- Inspecting contents with GAMS Studio
- Inspecting contents with $load
- Inspecting contents with GDXDUMP
- Inspecting contents with GDXDIFF

4.50.3.1 Inspecting contents with GAMS Studio

GAMS Studio has a built-in GDX Viewer, which offers extensive possibilities to view the content of a GDX file in a list or in a tabular view. For multidimensional data, you can rearrange the tabular view by simply dragging a column to a different position. You can also filter and sort, and quickly copy data to Excel or other spreadsheet programs. The GDX viewer is fast, which allows analysing big datasets. Learn more about the GDX viewer here.



Entry	Name	Type	Dim	Records	Text
1	i	Set	1	2	canning plants
2	j	Set	1	3	markets
3	a	Parameter	1	2	capacity of plant i in cases
4	b	Parameter	1	3	demand at market j in cases
5	d	Parameter	2	6	distance in thousands of miles
6	f	Parameter	0	1	freight in dollars per case per thousand miles
7	c	Parameter	2	6	transport cost in thousands of dollars per case
8	x	Variable	2	6	shipment quantities in cases
9	z	Variable	0	1	total transportation costs in thousands of dollars
10	cost	Equation	0	1	define objective function
11	supply	Equation	1	2	observe supply limit at plant i
12	demand	Equation	1	3	satisfy demand at market j

i [1]	j [2]	Level	Marginal
seattle	new-york	50	0
seattle	chicago	300	0
seattle	topeka	0	0.036
san-diego	new-york	275	0
san-diego	chicago	0	0.009
san-diego	topeka	275	0

4.50.3.2 Inspecting contents with $load

The $load command without any parameters will show a listing of all symbols in the GDX file. See Example 3 - Reading a GDX File on how to use $load to get a listing of all symbols.

4.50.3.3 Inspecting contents with GDXDUMP

The GDXDUMP utility can list the symbols in the file and it also can write sets and parameters formatted as a GAMS data statement.

```
gdxdump gdxfile_name format=choice symb=optional choice
```

where

- the `gdxfile_name` is the name of the GDX file to write data from.

- the output is created to the screen not to a file. One can also write the GDX file contents into a GAMS file using the command:

```
gdxdump gdxfile_name > filetouse.gms
```

- Data for a selected set, parameter, variable or equation (under all three of the output options when a specific item is named using the SYMB option)
- Data for all sets, parameters, variables and equations (Under normal option when the SYMB is not used)
- Data on solution items (variables and equations) formatted in a fashion suitable for import as a basis in another GAMS program where the marginals and levels are output. All of the scalars, sets and parameters (tables) in a GDX file to standard output formatted as a GAMS program with data statements or in CSV format. It skips information for variables and equations.
- Under the format=CSV choice it only creates output when a symbol is selected using the SYMB syntax.
- Under the format=CSV choice when the requested symbol is a variable or an equation one only gets the level values not the marginal, under the other formats one gets all items.
- Under the format=gamsbas choice one gets all variables and equations when the SYMB syntax is not used.

Suppose we wish to write out the GDX file `tran.gdx`, then we would use the command:

```
gdxdump tran
```

See more Examples on inspecting contents with GDXDUMP.

4.50.3.4 Inspecting contents with GDXDIFF

The GDXDIFF utility can be used to compare two GDX files by creating a third GDX file containing a list of differences between all symbols. In particular for all items with the same name, type and dimension in the two GDX files the differences in numerical values are written to a third GDX file with A summary report written to standard output (ordinarily the LOG file).

Besides the integrated solution in GAMS Studio, this utility can be used either at command line, or by $Call, or execute command.

```
gdxdiff file1 file2 {diffile} {Eps = value} {RelEps = value} {Field = FieldName} {ID=Identifier}
```

GDXDIFF requires the first two file name parameters,

- File1 Name of the first GDX file
- File2 Name of the second GDX file

The remaining parameters are optional

- Diffile An optional third file name that is the name of the GDX file that contains the differences found in the parameters. If that parameter, is absent the file will be named 'diffile.gdx' and placed in the current directory.
- Eps = value A tolerance that is the maximum amount that two numbers may differ by ie given a1 and a2 then abs(a1-a2) is reported as different if it exceeds this tolerance
- RelEps = value A tolerance that is the maximum percentage amount that two numbers may differ by ie given a1 and a2 then abs(a1-a2)/max(abs(a1),abs(a2)) is reported as different if it exceeds this tolerance.
- Field = FieldName A field that if specified limits doen between the information for variables and equations to specific attributes (Lo, L, Up, M, Scale and Prior)
- ID=Identifier Limits the comparisons to selected items; items not specified will be ignored. Multiple items can be specified as: ID=id1 ID=id2 or ID="id1 id2"

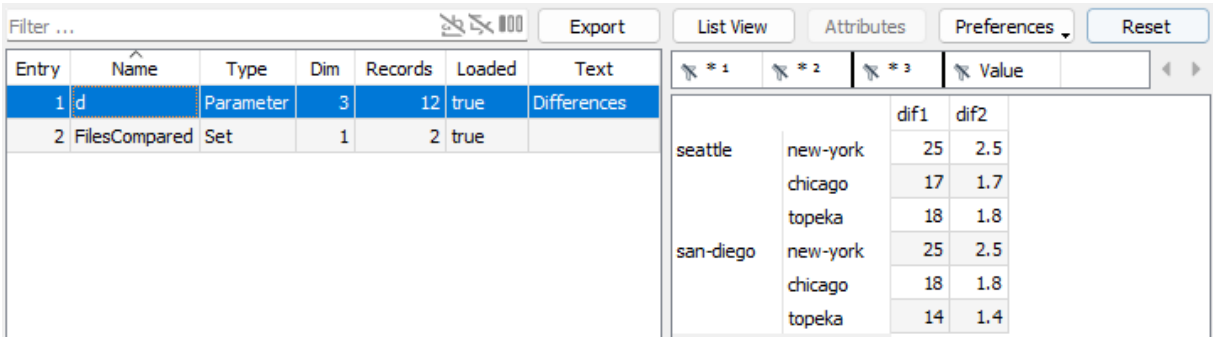
Suppose we wish to compare the GDX files `tran.gdx` and `tran2.gdx`, then we would use the command:

```
gdxdiff tran tran2
```

In turn the output to standard output (nominally the terminal screen) appears as follows:

```
Summary of differences:
     d   Data is different
   dem   Keys are different
   sup   Keys are different
supply   Symbol not found in file 1
     x   Symbol not found in file 1
```

and summarizes the differences found. Simultaneously the file `diffile.gdx` when examined in GAMS Studio contains the following:



which reports on the differences found in the two files.

Note

- Some new coding is introduced in the difference GDX file. Namely a new dimension is added to the parameters being compared which can contain 4 entries
  - dif1 indicates that the entry occurs in both files and shows the value found in the first file.
  - dif2 indicates that the entry occurs in both files and shows the value found in the second file.
  - ins1 indicates that the entry only occurs in the first files and shows the value found.
  - ins2 indicates that the entry only occurs in the second file and shows the value found.
- Only named items with the same name, type and dimension will be compared in the `diffile.gdx` output. Named items that are new or are deleted will only appear in the standard output summary report

See more Examples on inspecting contents with GDXDIFF.

4.50.4 General notes on GDX files

There are several things worth noting about GDX files:

- When working with GDX only one GDX file can be open at a time.
- When reading data from a GDX file, the symbol to be read must be declared before the reading statement appears.
- When the GDX file to be written has the same name as an existing GDX file the existing file will be overwritten. The resultant file will only contain the new data; there is no merge or append option.
- The $unload command to write a GDX during compile time will only write out data defined in the compilation at the point where the command appears. No results of any solves or calculations done within the current GAMS program will be reported with $unload. This is not true when using the execute_unload or execute_unloaddi commands.
- Both execute_unload and execute_unloaddi will write out data defined in the execution sequence at the point where the command appears. The results of the most recent solve command and any parameter calculations occurring before the GDX write will be reported.
- Any subsequent execute_unload or execute_unloaddi to a GDX file written earlier will totally overwrite that file so care must be taken to write all wanted information in the last appearing execute_unload or execute_unloaddi.
- A command line GDX write using the `gdx=file_name` command line parameter will write out data defined at the end of the execution sequence. The results of the most recent solve and any parameter calculations will be reported.
- When loading data domain checking will not be enforced. When items are resident in the GDX file for set elements that are not present in the current file these items will be ignored. GAMS will not generate any message telling you which items are ignored.
- Options Savepoint and execute_Loadpoint provide a GDX way of saving and loading a basis.
- The contents as they differ between GDX files can be examined with GDXMERGE or GDXDIFF.
- Starting with GAMS version 22.3, gdx files can be written in a compressed format unless the environment variable `GDXCOMPRESS` is set to zero. A value of 1 indicates compression.
- GDX files from a different GAMS version can possibly be incompatible due to compression among other changes. A current GAMS system can read all older GDX file formats. GDX files can be converted to a compressed format or an older format using GDXCOPY.
- Users can also write their own programs accessing data in a GDX file via the `expert-level GDX API` using `gdxcclib` library.

4.50.5 GAMS Data eXchange Tools

A number of GDX based tools and GDX related tools are included in the GAMS distribution and maintained by GAMS. See also Tools Manuals.

- The data exchange tools provide functionality to exchange data between GAMS and other data sources.
- The GDX service tools operate directly on GDX containers.
- Some of Data transformation tools perform very specific data transformation tasks that are either awkward or inefficient to implement in GAMS directly.

Some utilities are avaiable only on specific platform. See Supported Platforms for more details.

4.51 Extended Mathematical Programming (EMP)

Extended Mathematical Programming (EMP) is an extension to algebraic modeling languages that facilitates the automatic reformulation of new model types as models in more established mathematical programming classes, allowing them to be solved by mature solver algorithms. A number of important problem classes can be solved, e.g. Nash games and equilibria, bilevel programs, Disjunctive Programs and Stochastic Programs. EMP is independent of the modeling language used but is currently implemented only in GAMS. The new types of problems modeled with EMP are reformulated, using the GAMS solver JAMS, as well established types of problems, and the reformulated models are passed to a suitable GAMS solver to be solved.

EMP models are defined by information taken from two places: the traditional model definition and a text file containing annotations or additional information. The usual model definition contains variables, constraints and/or functions, and perhaps also an objective or matching information. Additional annotations to specify relationships that don't fit within this traditional definition are taken from the EMP info file. For example, the constraints for two optimizing agents in a competitive game can be specified with traditional algebra, while the structure (who owns what variables and constraints) can be specified in the info file. Together, this allows large, complex models to be specified in a convenient, precise, and flexible way.

This chapter is organized as follows.

Note

- At the end of each section we present and discuss the general syntax that EMP provides to write the annotations for that particular problem type. We use the usual GAMS syntax symbols: `[ ]` (the enclosed construct is optional), { } (the enclosed construct may be repeated zero or more times) and | (exclusive `OR`).
- Many EMP model examples are available in the **GAMS EMP Library**.

4.51.1 EMP Annotations: the EMP Info File

EMP models are defined by both the usual content of a GAMS model and annotations found in a simple text file named `emp.info` (aka the EMP info file). It is often most convenient to create this file via the GAMS put writing facility. The annotations primarily serve to *define* the model (e.g. to specify that a variable `u` is really the dual multiplier for a constraint `g`) but can also specify how a solver should process the model. The annotations make use of EMP keywords to do this.

A simple example will serve as illustration. Consider the following NLP:

$$\begin{array}{ll} \text{Min}_{x,y,z} & -3x + xy \\ \text{s.t.} & x + y \leq 1 \\ & x + y - z = 2 \\ & x, y \geq 0 \end{array} \tag{1}$$

We will use EMP annotations to automatically generate the first order conditions (KKT conditions) of this NLP and thus reformulate the NLP as an MCP:

```
Variables          f, z;
Positive Variables x, y;

Equations g, h, defobj;

g..       x + y =l= 1;
h..       x + y - z =e= 2;
defobj..  f =e= -3*x + x*y;

Model comp / defobj, g, h /;

File info / '%emp.info%' /;
putclose info / 'modeltype mcp';

solve comp using EMP minimizing f;
```

Observe that the model is defined in the usual way and the file `emp.info` contains just one line: `modeltype mcp`. The EMP keyword `modeltype` indicates that the value following the keyword is the model type to be used for the reformulation. In this example the model type is `mcp`. Here this specification is required: the sole point of our EMP annotations is to generate an MCP and not (as is usually the case) to define the model. Usually, the model algebra and annotations together imply the type of the reformulated model and so no `modeltype` specification is required or wanted. Finally, note that the model type in the solve statement is `EMP:` this is typical.

The solver JAMS implements the EMP framework. It processes the model and the annotations, automatically reformulates the original EMP model as a model of a different (more easily solved) type, passes the reformulated model on to an appropriate subsolver, and maps the resulting solution back into the original problem space.

In case users wish to inspect the (scalar) reformulated model, the JAMS option `FileName` may be used to specify the name of the file containing this model. Adding the following lines *before* the solve statement in the GAMS code above will cause the MCP reformulation to be saved in the file `myReform.gms`.

```
File empopt / 'jams.opt' /;
comp.optfile = 1;
putclose empopt / 'FileName myReform.gms';
```

The listing file will contain some additional information - the *EMP Summary* - as part of the output for each EMP model solved. We provide details on the EMP summary for each reformulation that we discuss below.

4.51.2 Soft Constraints

In many cases modelers wish to relax certain constraints: violation of these constraints is allowed but is associated with a well-defined penalty. The constraints that are allowed to be violated are called *soft constraints* and the constraints that continue to hold are called *hard constraints*.

In this section we present a mathematical formulation of soft constraints, give an example of how soft constraints can be modeled with GAMS EMP and introduce the EMP annotations specific to soft constraints.

4.51.2.1 Soft Constraints: Mathematical Formulation

A general formulation of a constrained minimization problem is:

$$\begin{array}{ll} \text{Min}_{x\in\mathbb{R}^n} & f(x) \\ \text{s.t.} & c_i(x) \leq 0, \quad \forall i \in \mathcal{I}, \end{array} \tag{2}$$

where f and the functions c_i are smooth, real-valued functions on a subset of $\mathbb{R}^n$ and $\mathcal{I}$ is a finite set of indices. Note that in this problem all feasible solutions must satisfiy all constraints c_i.

Now assume that some constraints are allowed to be violated (i.e. made soft), while the remaining constraints continue to hold for all feasible solutions. The soft constraints are associated with a *penalty function* that is added to the objective function. Since we have a minimization problem, the effect will be a balance or compromise between competing goals: minimizing the objective and minimizing the penalty functions.

Note

> The penalty terms for soft constraints are *added* to the objective function in a minimization problem and *subtracted* in a maximization problem.

Let $\mathcal{L} \in \mathcal{I}$ be the set of indices for the soft constraints: this imples $\mathcal{M} := \mathcal{I} \setminus \mathcal{L}$ is the index set for the hard constraints. Problem (2) becomes:

$$\begin{array}{ll} \text{Min}_{x\in\mathbb{R}^n} & f(x) + \sum_{i\in\mathcal{L}} w_i\, g_i\left(c_i(x)\right) \\ \text{s.t.} & c_i(x) \leq 0, \quad \forall i \in \mathcal{M}, \end{array} \tag{3}$$

where the penalty functions g_i are real-valued functions of $c_i(x)$, $i \in \mathcal{L}$. As we will see later, EMP has implementations of the following penalty functions: absolute value, least squares, and the maximum of a term and zero. Further, w_i is a multiplier associated with each penalty term, also called the *weight*. The weights facilitate prioritizing soft constraints: the weight of more important constraints will be greater than the weight of lesser constraints.

4.51.2.2 Soft Constraints with EMP: A Simple Example

The following simple example is adapted from the JAMS solver manual:

$$\begin{array}{ll} \text{Min}_{x_1,x_2} & -x_1^2 \\ \text{s.t.} & \log(x_1) = 1 \\ & x_2^2 \geq 2 \\ & 3x_1 + x_2 \leq 5 \\ & x_1,\, x_2 \geq 0 \end{array} \tag{4}$$

This problem can be formulated in GAMS as follows:

```
Positive Variables x1, x2;
Variables          obj;

Equations f0 "objective function", f1, f2, f3;

f0..   obj       =e= -sqr(x1);
f1..   log(x1)   =e= 1;
f2..   sqr(x2)   =g= 2;
f3..   3*x1 + x2 =l= 5;

Model m /all/;

x1.l = 1; x2.l = 1;

solve m using NLP min obj;
```

Note that this problem has no feasible solution. Thus we choose to relax the first two constraints by adding a penalty for their violation to the objective function. We also weight the relative importance or priority of the objective and the violations of these two constraints by introducing weights to go with these penalty functions. The resulting problem reads as follows:

$$\begin{array}{ll} \text{Min}_{x_1,x_2} & -x_1^2 + 5\,\|\log(x_1) - 1\|^2 + 2\max(x_2^2 - 2, 0) \\ \text{s.t.} & 3x_1 + x_2 \leq 5 \\ & x_1,\, x_2 \geq 0 \end{array} \tag{5}$$

Note that the first constraint is replaced by a least squares penalty of $(log(x_1) - 1)$ with a weight of 5 and the second constraint by the penalty term $\max(x_2^2 - 2, 0)$ with a weight of 2. However, the "max" penalty makes the objective function non-smooth. To rectify this, we implement the "max" penalty by introducing a new variable v that, due to its bounds and the direction of optimization, will take the value $\max(x_2^2 - 2, 0)$ at the solution. The result is:

$$\begin{array}{ll} \text{Min}_{x_1,x_2,v} & -x_1^2 + 5\,\|\log(x_1) - 1\|^2 + 2v \\ \text{s.t.} & 3x_1 + x_2 \leq 5 \\ & x_1,\, x_2 \geq 0 \\ & v \geq x_2^2 - 2 \\ & v \geq 0 \end{array} \tag{6}$$

This reformulation could of course be implemented directly in GAMS using standard GAMS syntax, but the EMP solution is easier to create and maintain, to read and understand, and to scale upwards as problem size and complexity increase. For the latter, we can simply add the following EMP annotations and updated solve statement (to specify solution as an `EMP`) to the GAMS code above:

```
File soft / '%emp.info%' /;
put soft;
$onput
adjustequ
f1 sqr 5
f2 maxz 2
$offput
putclose;
solve m using EMP min obj;
```

Note that the EMP annotations contain three lines: the first line, containing the EMP keyword `adjustequ` , indicates that the lines that follow specify soft constraints, i.e. equations to be converted to penalty terms; the second line specifies the name of the first soft constraint `f1`, the penalty function to use, and - optionally - the weight; and similarly the third line specifies the name of the second soft constraint `f2`, the penalty function to use, and the weight.

The solver JAMS will use the information in the EMP annotations to automatically reformulate problem (4) as problem (6) and pass it along to an NLP subsolver. The EMP Summary produced by JAMS will contain the following line:

```
--- EMP Summary
    ...
    Adjusted Constraint = 2
    ...
```

This output reflects that indeed two constraints were "adjusted", i.e. converted to penalty terms in the objective function.

4.51.2.3 EMP Syntax for Soft Constraints

The EMP framework provides the following general syntax to specify constraints that are converted to soft constraints:

```
AdjustEqu equ abs|sqr|maxz {weight}
         {equ abs|sqr|maxz {weight}}
```

The EMP keyword `AdjustEqu` indicates that the lines that follow specify equations that are "adjusted": they are moved from constraints to penalty terms in the objective function. `Equ` is the name of the equation to penalize, while the three EMP keywords that follow indicate the penalty function to use. If not specified, the weight used defaults to 1. For an example, see above or the model **[SIMPENLP]** in the GAMS EMP Model Library.

4.51.3 Variational Inequalities (VI)

Variational inequalities provide a general mathematical framework for many problems arising in optimization. For example, constrained optimization problems like LP and NLP are special cases of VI, and systems of equations and complementarity problems can be cast as VI. Thus VI problems have many applications, including those in transportation networks, signal processing, regression analysis, and game theory.

In this section we present a mathematical formulation of VI, give an example of how VI can be modeled with GAMS EMP, and introduce the EMP annotations specific to VI. Note that to fully and properly understand some of this section, the introduction to MCP provides a useful background.

4.51.3.1 Variational Inequalities: Mathematical Formulation

For a given continuous function $F : \mathbb{R}^n \to \mathbb{R}^n$ and a fixed closed convex set $K \subset \mathbb{R}^n$ the variational inequality problem $VI(F, K)$ is to find a point $x^* \in K$ such that:

$$\langle F(x^*), (x - x^*)\rangle \geq 0, \quad \forall x \in K, \tag{7}$$

where $\langle \cdot, \cdot \rangle$ denotes the usual inner product.

Observe that the VI generalizes many problem classes:

- If $F(x) = 0$ and $K \equiv \mathbb{R}^n$, the VI is a system of nonlinear equations.
- If $F(x) = \bigtriangledown f(x)$, the VI is a convex optimization problem.
- If $F(x) = \bigtriangledown f(x)$ and $K = \{x \,|\, Ax = b, Hx \leq h\}$, the VI is an NLP.
- If $F(x) = \bigtriangledown f(x) = p$ and $K = \{x \,|\, Ax = b, Hx \leq h\}$, the VI is an LP.
- If the feasible region is a box $B = \{x \in \mathbb{R}^n | \, l_i \leq x_i \leq u_i, \text{ for } i = 1, \ldots, n\}$, with $l_i \leq u_i,\ l_i \in \mathbb{R} \cup \{-\infty\}$ and $u_i \in \mathbb{R} \cup \{\infty\}$, the VI is an MCP, where $x^* \in B$ is a solution of the respective MCP if for each $i = 1, \ldots, n$ one of the following conditions holds:

$$\begin{array}{lll} F_i(x^*) = 0 & \text{and} & l_i \leq x_i^* \leq u_i, \\ F_i(x^*) > 0 & \text{and} & x_i^* = l_i, \\ F_i(x^*) < 0 & \text{and} & x_i^* = u_i. \end{array} \tag{4.18}$$

Note that the set K is frequently defined in the following way:

$$K = \{x \,|\, x \geq 0,\ h(x) \geq 0\}. \tag{8}$$

Further, note that the $VI(F, K)$ represents a wider range of problems than classical optimization whenever $F(x) \neq \bigtriangledown f(x)$ for some objective function f (or equivalently, the Jacobian of F is not symmetric). For example, problems that can be cast as VI include (generalized) Nash games and Nash equilibrium problems, systems of equations, complementarity problems, and fixed-point problems.

4.51.3.2 Variational Inequalities with EMP: A Simple Example

Consider the following simple three dimensional linear example (adapted from Yashtini & Malek (2007) [207]. Let

$$F(x) = \begin{bmatrix} 22x_1 - 2x_2 + 6x_3 - 4 \\ 2x_1 + 2x_2 \\ 6x_1 + 2x_3 \end{bmatrix}, \; K = \{x \in \mathbb{R}^{\not\models} \,|\, x_1 - x_2 \geq 1,\ -3x_1 - x_3 \geq -4,\ 2x_1 + 2x_2 + x_3 = 0,\ l \leq x \leq u\}, \tag{9}$$

where $l = (-6, -6, -6)^T$, $u = (6, 6, 6)^T$. N.B.: F is not the gradient of any function $\mathbb{R}^3 \to \mathbb{R}$. This $VI(F, K)$ has a unique solution: $x = (2/3, -1/3, -2/3)$. The problem can be implemented in GAMS with EMP as follows:

```
Set      i /1*3/;
Variable x(i);

x.lo(i) = -6;
x.up(i) =  6;

Equations F(i), h1, h2, h3;

F(i)..   (22*x('1') - 2*x('2') + 6*x('3') - 4)$sameas(i,'1')
         + (2*x('1') + 2*x('2'))$sameas(i,'2')
         + (6*x('1') + 3*x('3'))$sameas(i,'3')
         =n= 0;

h1..     x('1') -x('2') =g= 1;

h2..     -3*x('1') - x('3') =g= -4;

h3..     2*x('1') + 2*x('2') + x('3') =e= 0;

Model linVI / F, h1, h2, h3 /;

File annotations /'%emp.info%'/;
put annotations;
putclose 'vi F x h1 h2 h3';

solve linVI using EMP;
```

Observe that the function F and the constraints h are formulated using standard GAMS syntax. F is implemented as an equation of type =n=, which does not imply or enforce any relationship between the left-hand side and the right-hand side. Instead, this relationship is implied by the position of the matching variables (given in the EMP info file) relative to their bounds. The annotations in the EMP info file define the structure of the VI: what functions are matched to what variables, and what constraints serve to define the set K. The EMP keyword `vi` indicates that the model is a VI, that the VI function `F` is matched to the variable `x`, and that the constraints `h1 h2 h3` define the set K.

Alternatively, the EMP annotations could be written as follows:

```
putclose 'vi F x';
```

Here the equations after the equation-variable pair are omitted. This is acceptable, since by default any equations that are part of the model but are not matched with a variable are automatically used to define the set K.

Since VI problems have no objective, the short form of the solve statement is used.

The solver JAMS will reformulate the VI as an MCP and pass this on to an MCP subsolver. The EMP Summary produced by JAMS will contain the following line:

```
--- EMP Summary
    ...
    VI Functions        = 3
    ...
```

This output reflects the fact that there were three VI functions in the model above, one for each member of the set `i`.

Note that there are several VI models in the GAMS EMP Library. For example, the models [**SIMPLEVI**] and [**VI_MCP**] demonstrate how some models can be specified using either MCP or VI syntax. A simple nonlinear VI is given in model [**SIMPLEVI2**]. As the transportation model is so well known, model [**TRANSVI**] demonstrates how it can be cast as a VI.

4.51.3.3 EMP Syntax for Variational Inequalities

The general syntax of the EMP annotations used to specify variational inequalities is as follows:

```
VI [{var|*}] { [-] equ var} [{[-] equ}]
```

The EMP keyword `VI` indicates that this is a variational inequality specification. The core of the VI specification is the (list of) equation-variable pair(s): the other parts are optional. A pair matches the equation `equ` with the variable `var`. This indicates that `equ` defines part of the VI function F, and that these rows of F are perpendicular to columns taken from `var`. Multiple equation-variable pairs are allowed. The optional variables before the pairs are called *preceding variables*. These are variables that appear (and are often defined by) the constraints of the model, but they are not matched explicitly via the VI function F. Instead, they are automatically matched with the zero function. See model **[ZEROFUNC]** for an example and a more detailed discussion. The optional equations after the equation-variable pairs are called *trailing equations*. They define the set K and may be omitted from the VI specification. By default, any equations that are included in the model but are not matched with a variable are automatically used to define the set K. Even though both preceding variables and trailing equations may be omitted from the VI specification, we recommend to explicitly list them, since this clarifies intentions and eliminates ambiguity.

The "-" sign in the syntax above is used to flip (i.e. to reorient or negate) the marked equation, e.g. so that `x**1.5 =L= y` becomes `y =G= x**1.5`. Flipped equations in EMP behave in the same way as flipped equations in MCP.

Note

> More than one VI specification may appear in a model. Often, it makes no difference whether multiple equ-var pairs are part of the same or separate VI specifications, but this is not the case in general. For an example, see model **[SIMPLEVI4]**.

4.51.4 Quasi-Variational Inequalities (QVI)

Quasi-variational inequalities are a generalization of the variational inequality model: in a VI, the feasible set is fixed, while the QVI allows the feasible set to vary with or be a function of the variables in the model. To avoid repetition, we assume you are already familiar with the theory and notation for VI models. In this section, we present a mathematical formulation of QVI, give an example of how QVI can be modeled with GAMS EMP, and introduce the EMP annotations specific to QVI.

4.51.4.1 Quasi-Variational Inequalities: Mathematical Formulation

For a given continuous function $F : \mathbb{R}^n \to \mathbb{R}^n$ and a point-to-set mapping $K(y) : \mathbb{R}^n \to \mathbb{R}^n$, the quasi-variational inequality problem $QVI(F, K)$ is to find a point $y^* \in K$ such that:

$$\langle F(y^*), (y - y^*) \rangle \geq 0, \quad \forall y \in K(y^*), \tag{4.19}$$

where $\langle \cdot, \cdot \rangle$ denotes the usual inner product.

If the point-to-set mapping $K(y) : \mathbb{R}^n \to \mathbb{R}^n$ is constant, then the QVI above reduces to a VI. In order to define $K(x)$ as a function of x it is convenient to introduce a shadow copy x of the variable y. It should be understood that we have only one variable that appears in two forms: the variable of interest y and the parameter variable x. Where y appears, we have a variable in the usual sense, and we take derivatives wrt this variable when deriving optimality conditions. Where x appears, we have a constant variable, i.e. we assume it is fixed when deriving optimality conditions. We can now use a function $g(y, x)$ to define $K(\cdot)$ and express QVI as: find a point $y^* \in K$ (with its associated parameter variable x^*) such that:

$$\langle F(y^*), (y - y^*) \rangle \geq 0, \quad \forall y \text{ s.t. } g(y, x) <= 0. \tag{4.20}$$

4.51.4.2 Quasi-Variational Inequalities with EMP: A Simple Example

Consider the following simple two dimensional linear example. Let y be the variable of interest with its parameter variable x and let

$$F(y) = \begin{bmatrix} 2y_1 + \frac{8}{3}y_2 - \frac{100}{3} \\ 1.25y_1 + 2y_2 - 22.5 \end{bmatrix}, \; K(x) = \{y \in \mathbb{R}^{\not\perp} \mid y_1 + x_2 \leq 15, \; x_1 + y_2 \leq 20, \; 0 \leq y, x \leq 11\}. \quad (4.21)$$

This $QVI(F, K)$ has a solution $y = (10, 5)$. The problem can be implemented in GAMS with EMP as follows:

```
set i / 1*2 /;
alias(i,j);

table A(i,j)
      1       2
1     2     [8/3]
2   [5/4]     2   ;

parameters
  b(i) / 1 [100/3],  2 22.5 /
  Cy(i,j) / 1.1 1,  2.2 1 /
  Cx(i,j) / 1.2 1,  2.1 1 /
  rhs(i) / 1 15, 2 20 /
  ;
positive variables
  y(j)  'variable of interest, aka decision variable'
  x(j)  'parameter variable shadowing y'
  ;
y.up(j) = 11;   x.up(j) = 11;
equations
  F(i)   'FOC for agent optimization models'
  g(i)   'define feasible set K(x) for QVI'
  ;
F(i)..  sum{j, A(i,j)*y(j)} - b(i) =N= 0;

g(i)..  sum{j, Cy(i,j)*y(j)} + sum{j, Cx(i,j)*x(j)} =L= rhs(i);

model m / F, g /;
file annotations / '%emp.info%' /;
putclose annotations  'qvi  F y x  g'  ;
solve m using emp;
```

Observe that the function F and the constraints g are formulated using standard GAMS syntax. F is implemented as an equation of type =n=, which does not imply or enforce any relationship between the left-hand side and the right-hand side. Instead, this relationship is implied by the position of the matching variables (given in the EMP info file) relative to their bounds. The annotations in the EMP info file define the structure of the QVI: what functions are matched to what variables, and what constraints serve to define the mapping $K(x)$. The EMP keyword `qvi` indicates that the model is a QVI, that the QVI function `F` is matched to the variable-of-interest `y` with its parameter variable `x`, and that the constraints `g` define the mapping $K(x)$.

Since QVI problems have no objective, the short form of the solve statement is used.

The solver JAMS will reformulate the QVI as an MCP and pass this on to an MCP subsolver. The EMP Summary produced by JAMS will contain the following lines:

```
--- EMP Summary
    ...
    VI Functions        = 2
    QVI Parameters      = 2
    ...
```

This output reflects the fact that there were two VI functions in the model above, one for each member of the set `i`, and that each of the variables matched to these functions was shadowed by a parameter variable.

Note that there are two QVI models in the GAMS EMP Library, the models [**SIMPLEQVI1**] and [**SIMPLEQVI2**] (an expanded version of the example shown above).

4.51.4.3 EMP Syntax for Quasi-Variational Inequalities

The general syntax of the EMP annotations used to specify quasi-variational inequalities is as follows:

```
QVI {0 var [ parameterVar] | [-] equ var [ parameterVar] } { [-] equ}
```

The EMP keyword `QVI` indicates that this is a quasi-variational inequality specification. All variables and equations included in a QVI must be listed explicitly. First we have the VI functions, their matching variables, and (optionally) the parameter variables shadowing these variables. Note that in a QVI there are no *preceding variables* as we have in a VI spec:: instead, the implied match to the zero function is indicated by the digit `0` appearing where an equation symbol would otherwise appear. After the function/variable pairs have been listed, the *trailing equations* (i.e. the equations/constraints defining the mapping $K(\cdot)$) appear.

4.51.5 Equilibrium Problems

While optimization problems have *one* decision maker that controls all decison variables, equilibrium problems are a collection of optimization problems and variational inequalities, each controlled by a *different agent*. We typically assume that each variable and each equation is controlled by or belongs to exactly one agent. Variables that are controlled by one agent but appear in the equations of a second agent are regarded as fixed or exogenous variables by that second agent: when taking first-order conditions, the second agent won't take derivatives wrt these exogenous variables. Later we will relax this assumption and introduce equilibrium problems with shared constraints and shared variables. Note that in this section we will discuss equilibrium problems of the Nash type, i.e. where all agents are on the same level, each assuming the decisions or strategies of the other agents are known and fixed. Equilibrium problems of the Stackelberg type, where there are leaders and followers, are covered in section Bilevel Programs.

We start with the mathematical formulation of an equilibrium problem, next present two examples, and conclude with a description of the EMP annotations for equilibrium problems.

4.51.5.1 Equilibrium Problems: Mathematical Formulation

Consider the following equilibrium problem with N agents solving minimization problems and one agent solving a variational inequality:

$$\begin{array}{lll} \text{Find} & (x_1^*, \ldots, x_N^*, p^*) \text{ satisfying} & \\ x_i^* & \in \arg\min_{x_i} f_i(x_i, x_{-i}^*) & \\ & \text{s.t.} \quad g_i(x_i, x_{-i}^*) \leq 0, \text{ for } i = 1, \ldots, N, & (10) \\ p^* & \in \text{SOL}\,(H(p, x^*), K(x^*)), & \\ & \text{where } K(x^*) = \{p \,|\, w(p, x^*) \leq 0\}. & \end{array}$$

Note that $f_i(x_i, x_{-i})$ denotes the objective function of the problem of agent i, $g_i(x_i, x_{-i})$ are the constraints relating to this optimization problem and $x_{-i} = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_N)$ denotes the decisions of the other agents. Further, $\text{SOL}\,(H, K)$ represents the solution set of the variational inequality $VI(H, K)$.

This problem can be implemented with EMP as follows:

```
Set  i ;
Variables obj(i), x(i), p;
Equations deff(i), defg(i), defH, defw;

*Definitions of equations are omitted

Model equil / deff, defg, defH, defw /;

File myinfo /'%emp.info%'/;
put myinfo 'equilibrium';
loop(i,
   put / 'min', obj(i), x(i), deff(i), defg(i);
);
put 'vi defH p defw' /;
putclose myinfo;

solve equil using EMP;
```

Note that the GAMS variable `obj(i)` holds the value of $f_i(x)$, `x(i)` represents the variable x_i and `p` denotes the variable p, of course. The equations `deff(i)` and `defg(i)` are closed-form definitions of the objective function $f_i(x)$ and the constraint $g_i(x)$ respectively. The equation `defH` defines the variational inequality function H and the equation `defw` defines the set K. The EMP annotations found in the file `emp.info` specify the equilibrium structure of the model:

```
equilibrium
min obj('1') x('1') deff('1') defg('1')
...
min obj('N') x('N') deff('N') defg('N')
vi defH p defw
```

The EMP keyword `equilibrium` indicates that the annotations are for an equilibrium problem. Each of the EMP keywords `min` begins a new optimization problem (owned by a new agent), where each problem has its own objective variable, decision variables, and equations / constraints. For example, the first agent minimizes `obj('1')`, controls or owns `x('1')`, and is subject to the constraints `deff('1')` and `defg('1')`. If other variables like `x('2')` and `x('3')` appear in `deff('1')` and `defg('1')`, they will be treated as exogenous by the first agent. This specification is consistent with the formulation above in (9). Each agent's optimization problem can be easily (re)constructed given the EMP annotations. Following the optimization problems of the N agents we have the VI specification for agent p. The EMP keyword `vi` is followed by the equation-variable pair `defH p` defining the VI function H and the equation `defw` that defines the feasible set K.

Note that the short form of the solve statement is used for equilibrium problems. Objective variables (if and when they exist) belong to individual agents, not to the model as a whole.

4.51.5.2 Equilibrium Problems with EMP: A Simple Example

Consider the following example from Kim & Ferris (2017) [109]. In this economic equilibrium problem there are three agents: one profit-maximizing producer, one utility-maximizing consumer and a market that determines the price of three commodities based on production and demand. The problem data include a technology matrix A, where the entry $a_{ij} > 0$ denotes the output of the commodity i for each unit of activity of producer j and $a_{ij} < 0$ denotes the respective input. Further, an initial endowment b and the demand function $d(p)$ is given, where p is the price. The consumer maximizes her utility within her budget, which depends on the price p and the initial endowment b. Let y represent the activity of the producer, x represent the demand of the consumer and p represent the prices of commodities. Then (y^*, x^*, p^*) is a general equilibrium if it satisfies the following:

$$\begin{array}{rll} -A^T p^* & \geq 0 & \text{No positive profit for each activity} \\ b + Ay^* - d(p^*) & \geq 0 & \text{No excess demand} \\ p^* \geq 0,\ y^* & \geq 0 & \text{Nonnegativity} \\ -A^T p^* & \perp y^* & \text{No activity for earning negative profit and positive activity implies balanced profit} \\ b + Ay^* - d(p^*) & \perp p^* & \text{Zero price for excess supply and market clearance for positive price} \end{array} \tag{4.22}$$

The code for the respective model is given below. Note that instead of using the consumer demand function $d(p)$ in its explicit form, we introduce a utility-maximizing consumer with demand x.

```
set i 'commodities' / 1*3 /;

variable u 'consumer utility';
positive variables
  y        'activity of the producer'
  x(i)     'Marshallian demand of the consumer'
  p(i)     'prices'
  ;
parameters
  A(i) 'technology matrix'  / 1 1, 2 -1, 3 -1 /
  s(i) 'budget share'       / 1 0.9, 2 0.1, 3 0 /
  b(i) 'endowment'          / 1 0, 2 5, 3 3 /
  ;
equations
  profit   'profit of activity'
  mkt(i)   'constraint on excess demand'
  udef     'Cobb-Douglas utility function'
  budget   'budget constraint'
  ;
profit..   -sum(i, A(i)*p(i)) =g= 0;

mkt(i)..   b(i) + A(i)*y - x(i) =g= 0;

udef..     u =e= sum(i, s(i)*log(x(i)));

budget..   sum(i, p(i)*x(i)) =l= sum(i, p(i)*b(i));

model m / mkt, profit, udef, budget /;

file empinfo /'%emp.info%'/;  putclose empinfo
 'equilibrium' /
 ' max', u, 'x', udef, budget /
 ' vi profit y' /
 ' vi mkt p' /
```

```
 ;
* the second commodity is used as a numeraire
p.fx('2') = 1;
x.l(i) = 1;

solve m using EMP;
```

Observe that in the EMP annotations the problems of the three agents are specified after the EMP keyword `equilibrium:` the consumer solves a maximization problem (where the utility `u` is maximized) and the activities of the producer and the price-setting market are expressed as VI. As there are three commodities, the first VI actually generates three VI functions, one for each commodity. Thus there are three agents and four VI functions in the equilibrium problem. This is reflected in the EMP Summary in the listing file:

```
--- EMP Summary
    ...
    VI Functions        = 4
    Equilibrium Agent   = 3
    ...
```

In the GAMS EMP Library there are several models that have a similar form, e.g. Scarf's activity analysis model [**SCARFEMP-DEM**] and the simple equilibrium problem [**SIMPEQUIL**]. The latter demonstrates that there are equilibrium problems where the optimization problems of the individual agents are solvable, but the overall equilibrium problem does not have a solution.

4.51.5.3 Equilibrium Problems with EMP: Example with Dual Variables

In many applications equilibrium problems come with a twist: the dual variable associated with a constraint in the problem of one agent appears *exogenously* in the problem of another agent. The following simple example with two agents is from model [**DUALVAR**] in the GAMS EMP Library.

$$\begin{array}{lll} \text{Problem of the first agent:} \min_{(v,w)} & z = v + w & \\ \text{s.t.} & \sqrt{v+1} + 2w \geq 2 & (\perp\ u \geq 0) \\ & v, w \geq 0 & \end{array} \tag{11}$$

$$\text{Problem of the second agent:} VI: \quad F(y) := y - 4u + 1 \quad (\perp\ y \text{ free})$$

N.B.: the variable u that appears in the problem of the second agent is the dual multiplier (aka shadow price) of the first agent's constraint. This equilibrium problem can be modeled in GAMS with EMP as follows:

```
positive variables
  v     'belongs to min agent'
  w     'belongs to min agent'
  u     'dual of min constraint'
  ;
free variables
  y     'belongs to VI agent'
  z     'objective var'
  ;
equations
  defz  'objective def'
  g     'constraint for min agent'
```

```
  Fy    'VI function'
  ;
defz..   v + w =e= z;

g..      sqrt(v+1) + 2*w =g= 2;

Fy..     y - 4*u + 1 =n= 0;

Model opt 'min agent and VI agent' / defz, g, Fy /;

File empinfo / '%emp.info%' /;  putclose empinfo
'equilibrium' /
'  min z v w defz g' /
'  vi Fy y' /
'  dualvar u g' /
;
defz.m = -1;
g.m = 0.5;   Fy.m = 1;
v.l = 0;      w.l = 0.5;
y.l = 1;      u.l = 0.5;

solve opt using emp;
```

The EMP info file contains the EMP keyword `equilibrium` followed by the specifications for the two agents: a minimization problem for the first agent and a VI for the second agent. The special relationship between the variable `u` and the equation `g` is declared via the EMP keyword `dualvar` followed by the respective variable-equation pair. Recall our usual assumption that each variable and each equation is owned or controlled by exactly one agent. Since variable `u` is tied to equation `g` and `g` is owned by the first agent, variable `u` is owned by the first agent also.

Besides the number of equilibrium agents and the number of VI functions, the EMP Summary lists the number of dual variable maps:

```
--- EMP Summary
    ...
    Dual Variable Maps  = 1
    ...
    VI Functions        = 1
    Equilibrium Agent   = 2
    ...
```

Note

Although the example above contained an optimizing agent and a VI agent, dual variables most often occur in equilibrium problems with several optimizing agents.

Other examples with dual variables in the GAMS EMP Library include a formulation of the well-known transportation model as an equilibrium problem [**TRANSEQL**], Scarf's activity analysis model [**SCARFEMP-PRIMAL**] and the general equilibrium model [**TWO3EMP**].

4.51.5.4 EMP Syntax for Equilibrium Problems

The EMP framework provides the following general syntax to specify equilibrium problems:

```
Equilibrium
  {VIsol {equ}}
  {Implicit {var equ}}
  {MAX|MIN obj {var|*} {[-] equ}}
  {VI {var|*} {[-] equ var} {[-] equ}}
  {DualVar {var [-] equ}}
```

The EMP keyword `Equilibrium` indicates that the specifications that follow define the structure of an equilibrium problem. The `MAX`, `MIN`, and `VI` keywords specify agents in the problem, while the rest of the keywords are optional modifiers used to adjust the structure of the agent models or the meaning of the equations they contain.

Note

An equilibrium problem must contain at least one agent, i.e must contain one of the keywords `MAX`, `MIN`, or `VI`.

The EMP keyword `VIsol` identifies a shared constraint(s) and specifies the MCP reformulation to use for it: see section Equilibrium Problems with Shared Constraints below for details. The keyword `Implicit` identifies a shared variable and its defining constraint: see section Equilibrium Problems with Shared Variables below for details. The keywords `MAX` and `MIN` each begin the specification of an optimization agent and are followed by the objective variable `obj` and the other variables and equations owned by the agent, as described in the formulation and example above. The keyword `VI` begins the specification of a VI agent: see the section on VI above for details. Finally, the EMP keyword `DualVar` specifies that the variable `var` is the Lagrange multiplier for the equation `equ`. For examples, see sections Equilibrium Problems with EMP: Example with Dual Variables and Embedded Complementarity Systems.

The symbol '*' specifies that the default or automatic assignment of variables to this agent be used, i.e. the set of variables used in the equations owned by this agent but not explicitly or otherwise assigned to another agent. Note that if a variable occurs in equations owned by multiple agents and is not explicitly assigned to any agent, the default assignment is not well defined and using it will be flagged as an error. To avoid confusion and promote clarity, we recommend that modelers use explicit variable lists and avoid the '*' symbol.

The "-" sign in the syntax above is used to flip (i.e. to reorient or negate) the marked equation, e.g. so that `x**1.5 =L= y` becomes `y =G= x**1.5`. Flipped equations in EMP behave in the same way as flipped equations in MCP.

Note

When the EMP keyword `equilibrium` appears in the EMP annotations, the solve statement takes the short form also used for complementarity problems in GAMS.

4.51.6 Embedded Complementarity Systems

Embedded complementarity systems of the following form arise frequently in applications:

$$\begin{array}{lll} \text{Min}_x & f(x,y) & \\ \text{s.t.} & g(x,y) \leq 0 & (\perp\ \lambda \geq 0) \end{array} \tag{12}$$

$$H(x,y,\lambda) = 0 \qquad (\perp\ y \text{ free}) \tag{4.23}$$

Note that the optimization problem is over the variable x and it is parametrized by the variable y. The choice of y is determined by the complementarity relationships represented here by H.

From an EMP perspective, there are two ways to annotate a GAMS model to specify the model above: we will describe both below. These approaches provide equivalent additional information that prompts the EMP tool to automatically create the following MCP:

$$\begin{array}{lll} 0 = \bigtriangledown_x L(x,y,\lambda) & \perp & x \text{ free} \\ 0 \leq -\bigtriangledown_\lambda L(x,y,\lambda) & \perp & \lambda \geq 0 \\ 0 = H(x,y,\lambda) & \perp & y \text{ free}, \end{array} \tag{13}$$

where the Lagrangian is defined as

$$L(x,y,\lambda) = f(x,y) + \lambda^T g(x,y). \tag{4.24}$$

The first approach uses the EMP keywords `dualequ` and `dualvar`, as contained in the model [**FERRIS43**].

```
variables         obj, x, y;
positive variable lambda;

equations defobj, g, H;

* We omit the equation definitions here.

model ecs /defobj, g, H/;

file empinfo / '%emp.info%' /; putclose empinfo
'dualequ H y' /
'dualvar lambda g' / ;

solve ecs using EMP minimizing obj;
```

The external constraint H is expressed as a standard GAMS equation. The long form of the solve statement is used here, which implies the existence of a single optimizing agent that by default owns all equations and variables. The first EMP keyword `dualequ` indicates that the equation `H` and the variable `y` do not belong to the optimizing agent: instead, `y` is treated as an exogenous variable by this agent, and this agent is assumed to know nothing about the functional form of `H`, so that `H` will not appear in any first-order conditions. Instead, a complementarity relationship between the function defined by `H` and the variable `y` is required to exist at optimality. The EMP keyword `dualvar` indicates that the variable `lambda` is the dual of the equation `g`. As a result `lambda` will be treated exogenously wherever it appears. Given the EMP annotations for this model, JAMS will automatically reformulate the problem as the MCP in (12) and pass this model to an MCP subsolver.

The EMP Summary produced by JAMS contains the following lines:

```
--- EMP Summary
    Dual Variable Maps  = 1
    Dual Equation Maps  = 1
```

The second modeling approach recasts the problem above as an equilibrium problem with two agents: the first agent solves a minimization problem and the second agent solves a VI. The algebra in the model remains the same: only the EMP annotations and the solve statement change.

```
putclose empinfo
'equilibrium' /
' min obj x g defobj' /
' vi H y' /
' dualvar lambda g' /

solve ecs using EMP;
```

The EMP keyword `equilibrium` indicates we have an equilibrium problem. The EMP keyword `min` indicates that the first agent solves a minimization problem with the objective variable `obj`, the decison variable `x` and the equations `g` and `defobj`. In contrast to the first approach, where the default (because of the long-form solve statement) is one optimizing agent owning all equations, here we specify the first agent's minimization model explicitly and from the ground up. As a result it doesn't contain the equation `H` and we do not use the `dualequ` keyword to take `H` out. Instead, the EMP keyword `vi` specifies that the second agent solves a VI defined by `H` matched with the variable `y`. The `dualvar` keyword functions here as it did in the previous example.

The EMP Summary produced by JAMS contains the following lines:

```
--- EMP Summary
    ...
    Dual Variable Maps  = 1
    Dual Equation Maps  = 0
    VI Functions        = 1
    Equilibrium Agent   = 2
    ...
```

Other examples of embedded complementarity systems in the GAMS EMP Library include the simple equilibrium problem [**SIMPEQUIL2**], the equilibrium problem formulation of the well-known transportation model [**TRANSECS**], the PIES energy equilibrium problem [**PIES**], the pure exchange model [**NEGISHI**] and the spatial price equilibrium model [**HARK-MONOP**].

It's worthwhile highlighting the differences between the `dualvar` and `dualequ` keywords, as the two are easily and frequently confused. The `dualvar` keyword makes reference to a constraint owned by an optimizing agent. Derivatives of this constraint, multiplied by the variable referenced, appear in the first-order optimality conditions for this agent. The `dualvar` keyword allows us to use this variable or multiplier explicitly (and in a sense exogenously) in the model algebra. In constrast, the `dualequ` keyword indicates that, contrary to what the default is, an equation is not owned by any optimizing agent, so no derivatives of this equation will appear in any FOC or in the reformulated model. The two are similar in that if a variable `x` appears with either `dualvar` or `dualequ`, no derivatives w.r.t. `x` will appear in the model reformulation.

dualvar x F	**dualequ F x**
Variable symbol appears first	Equation symbol appears first
F is owned by an optimizing agent	F is a system constraint
Derivatives of F appear in FOC	No derivatives of F appear in derived model
Does not change ownership of F	F is taken away from an optimizing agent

Table 1: Differences between dualvar and dualequ

4.51.7 Equilibrium Problems with Shared Constraints

In the equilibrium problems we have discussed so far, each variable and each constraint was owned by a single agent. In this section we relax the assumption that each constraint has to be controlled by a single agent and introduce *shared constraints*. Shared constraints are constraints that appear in the problems of several agents. They are mainly used to model resources shared among agents. Note that the examples in this section are from Kim & Ferris (2017) [109].

Consider the following example:

$$\begin{array}{lll} \text{Find} & (x_1^*, \ldots, x_N^*) & \text{satisfying} \\ x_i^* \in & \arg\min_{x_i} & f_i(x_i, x_{-i}^*) \\ & \text{s.t.} & g_i(x_i, x_{-i}^*) \leq 0 \\ & & h(x_i, x_{-i}^*) \leq 0, \qquad \text{for } i = 1, \ldots, N. \end{array} \tag{14}$$

The constraints g_i are owned by agent i, while the constraint h is shared by all agents. This example can be reformulated in two different ways, each with its own EMP annotations and resulting MCP. However, both share the model formulation below:

```
variables obj(i), x(i);
equations deff(i), defg(i), defh;

model sharedc / deff, defg, defh /;
```

Observe that the equation `defh` implementing the shared constraint $h(x) \leq 0$ is not indexed: it is exists once, not once per agent. The first way to write the EMP annotations is as follows:

```
file empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
loop(i,
    put 'min', obj(i), x(i), deff(i), defg(i), defh; /;
);
putclose empinfo;
```

Here the equation `defh` appears in the minimization problem of each agent. As the equation `defh` is not indexed by agent and appears in the problem of each agent, it is easy to see that it is a shared constraint. Given these annotations, the EMP framework aggregates the FOC for each agent to create the following MCP:

$$\begin{array}{lll} F(z) = ((F_i(z)^T)_{i=1}^N)^T, & & z = ((z_i^T)_{i=1}^N)^T, \\ F_i(z) = \begin{bmatrix} \nabla_{x_i} f_i(x) - \nabla_{x_i} g_i(x)\lambda_i - \nabla_{x_i} h(x)\mu_i \\ g_i(x) \\ h(x) \end{bmatrix}, & z_i = \begin{bmatrix} x_i \\ \lambda_i \leq 0 \\ \mu_i \leq 0 \end{bmatrix}, & \text{for } i = 1, \ldots, N. \end{array} \tag{15}$$

Note that the equation $h(\cdot)$ is replicated for each agent and each agent is assigned a separate multiplier μ_i.

Attention

> By default, the solver JAMS does not allow shared constraints. The option `SharedEqu` needs to be specified in the option file `jams.opt` if shared constraints are intended, otherwise an error is reported when they are detected. This cautious default helps minimize the number of surprising results.

A full model including the JAMS option file is given below.

The second way to write the EMP annotations uses the EMP keyword `VIsol` to specify that a *variational equilibrium* be computed, i.e. a solution where the multipliers μ_i are all equal. The `VIsol` keyword should appear *after* the EMP keyword `equilibrium` and *before* the descriptions of agent models.

```
put / 'VIsol defh';
```

The multipliers to make equal (or, equivalently, the constraints not to replicate) are indicated by the equation name(s) following the keyword `VIsol`. These constraints appear once in the resulting MCP, not once per agent. The EMP tool creates the following MCP:

$$\begin{array}{ll} F(z) = ((F_i(z)^T)_{i=1}^N, F_h(z)^T)^T, & z = ((z_i^T)_{i=1}^N, z_h^T)^T, \\ F_i(z) = \begin{bmatrix} \bigtriangledown_{x_i} f_i(x) - \bigtriangledown_{x_i} g_i(x)\lambda_i - \bigtriangledown_{x_i} h(x)\mu \\ g_i(x) \end{bmatrix}, & z_i = \begin{bmatrix} x_i \\ \lambda_i \leq 0 \end{bmatrix}, \quad \text{for } i = 1, \ldots, N, \\ F_h(z) = \big[h(x)\big], & z_h = \big[\mu \leq 0\big]. \end{array} \tag{16}$$

If there are no constraints $g_i(x)$, then - assuming a constraint qualification - the equilibrium problem with the shared constraint corresponds exactly to the variational inequality $VI(F, X)$, where the set $X = \{x \,|\, h(x) \leq 0\}$ and $F(x) = ((\bigtriangledown_{x_i} f_i(x)^T)_{i=1}^N)^T$. This corresponds in turn to the solution of the first MCP (with the replicated shared constraints) in case that solution has the property that the multipliers μ_i are all equal. This is what gives the latter solution its name *variational equilibrium*. If the equilibrium problem with the shared constraint has a unique solution, both MCP reformulations will have the same solution. Otherwise, the two MCPs that correspond to the two different EMP annotations may have different solutions.

An example for shared constraints is the pollution of a river basin in Haurie & Krawczyk (1997) [96] and Krawczyk & Uryasev (2000) [111], where the total amount of pollutants that may be dumped in a river is restricted. This implies that the environmental constraints of some pollutant-producing agents are shared. This example can be formulated as follows, where i is the agent index and m denotes the number of shared constraints.

$$\begin{array}{llll} \text{Find} & (x_1^*, x_2^*, x_3^*) & \text{satisfying} & \\ x_i^* \in & \arg\min_{x_i} & (c_{1i} + c_{2i} x_i)\, x_i - \Big(d_1 - d_2\Big(\sum_{j=1,\, j\neq i}^3 x_j^* + x_i\Big)\Big)\, x_i & \\ & \text{s.t.} & \sum_{j=1,\, j\neq i}^3 x_j^* (u_{jm} e_j x_j^*) + u_{im} e_i x_i \leq K_m, & \\ & & \text{for } i = 1, 2, 3,\ m = 1, 2, & \\ & \text{where} & (c, d, e, u, K) \text{ is the problem data.} & \end{array} \tag{17}$$

Here three agents produce some commodities and aim to maximize their profit. The term $(c_{1i} + c_{2i} x_i)\, x_i$ denotes the total cost for each agent i and the term $(d_1 - d_2(\sum_{j=1,\, j\neq i}^3 x_j^* + x_i))\, x_i$ represents the revenue for each agent. The amount of pollutants each agent dumps in the river is limited by the two shared constraints. This problem can be implemented in GAMS with EMP as follows:

```
Sets i / 1*3 /
     m / 1*2 / ;
alias(i,j) ;

Variable          obj(i) ;
Positive Variable x(i) ;

Parameters K(m) / 1 100, 2 100 /
           d1   / 3 /
           d2   / 0.01 /
           e(i) / 1 0.5, 2 0.25, 3 0.75 / ;

Table c(m,i)
       1      2     3
  1  0.1   0.12  0.15
  2  0.01  0.05  0.01 ;

Table u(i,m)
       1      2
  1  6.5    4.583
  2  5.0    6.250
  3  5.5    3.750 ;

Equations  objdef(i)
           cons(m) ;

objdef(i)..  obj(i) =e= (c('1',i) + c('2',i)*x(i))*x(i) - (d1 - d2*sum(j, x(j)))*x(i);

cons(m)..    sum(i, u(i,m)*e(i)*x(i)) =l= K(m);

Model m_shared / objdef, cons /;

File empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
put 'VIsol cons' /;
loop(i,
    put 'min', obj(i), x(i), objdef(i), 'cons' /;
);
putclose empinfo;

$echo SharedEqu > jams.opt
m_shared.optfile = 1;

solve m_shared using emp;
display x.l, cons.m;
```

Note that the shared environmental constraints are expressed with the equation `cons`. In the EMP annotations, we have chosen the formulation with the EMP keyword `VIsol`. It leads to the solution $x^* = (21.145, 16.028, 2.726)$. Alternatively, we could delete the line with the EMP keyword `VIsol` and thus prompt the framework to compute the MCP in (15), resulting in the solution $x^* = (0, 6.473, 22.281)$.

4.51.8 Equilibrium Problems with Shared Variables

In the last section, we relaxed the restriction that each constraint has to be controlled by a single agent and introduced *shared constraints*. In this section, we go a step further and allow *shared variables*. Note that the content of this section is adapted from Kim & Ferris (2017) [109].

First, we introduce the notion of *implicit variables*. In mathematical terms, a variable y is called an implicit variable if for each value of x there is at most one value of y satisfying $(y, x) \in X$. For such implicit variables there exists one and only one function $g(\cdot)$ such that $(g(x), x) \in X$, where g is defined over the set $\{x \mid \exists\, y \text{ such that } (y, x) \in X\}$. The set X is called the *defining constraint* of the variable y: the value of y is implicitly defined by the value of x via the defining constraint. In the current implementation, the defining constraint may only be represented as a system of equations and implicit variables must be free variables in GAMS.

Shared variables in equilibrium problems are implicit variables that have the same defining constraint for all agents that share the variable. Hence, the defining constraint becomes a shared constraint. Consider the following equilibrium problem with the shared decision variable y and its defining constraint $X = \{(y, x) \mid H(y, x) = 0\}$:

$$\begin{array}{llll} \text{Find} & (y^*, x_1^*, \ldots, x_N^*) & \text{satisfying} & \\ (y^*, x_i^*) \in & \arg\min_{y, x_i} & f_i(y, x_i, x_{-i}^*) & \\ & \text{s.t.} & H(y, x_i, x_{-i}^*) = 0, \quad \text{for } i = 1, \ldots, N, & \\ & \text{where} & H : \mathbb{R}^{m+n} \to \mathbb{R}^m,\ y \in \mathbb{R}^m & \end{array} \tag{18}$$

Note that there are N agents, m is the dimension of the variable y and $n = \sum_i n_i$, where n_i is the dimension of x_i. Assuming we have 4 agents, this example can be implemented in GAMS EMP as follows:

```
Set       i / 1*4/;
Variables obj(i), x(i), y;

Equations deff(i), defH;

Model     sharedv / deff, defH /;

File empinfo / '%emp.info%' /;
put empinfo 'equilibrium' /;
put 'implicit y defH' /;
loop (i,
   put 'min', obj(i), x(i), y, deff(i) /;
);
putclose empinfo;

solve sharedv using EMP;
```

In the EMP annotations, the EMP keyword `implicit` is used to declare the implicit variable `y` and its defining constraint `defH`. Note that the keyword `implicit` must be followed by variable-constraint pairs. If multiple pairs are specified with a single keyword `implicit`, they will be augmented to form a single vector of implicit variables and its defining constraint.

Note

Implicit variables are declared *before* the agent problems are defined.

Observe that the shared variable `y` appears in the problem specification for each agent. However, the defining equation `defH` does not appear in each problem specification, since it is assumed to be part of the implicit variable.

4.51.8.1 Reformulation Strategies for Equilibrium Problems with Shared Variables

Like other equilibrium problems, equilibrium problems with shared variables are reformulated as MCPs by the EMP tool. Users can choose between three different reformulation strategies if shared decision variables are involved. In the first strategy, the shared variables are *replicated* for each agent and the respective KKT conditions are computed, resulting in the following MCP:

$$\begin{aligned} F(z) &= [(F_i(z)^T)_{i=1}^N]^T, & z &= [(z_i^T)_{i=1}^N]^T \\ F_i(z) &= \begin{bmatrix} \bigtriangledown_{x_i} f_i(x,y) - (\bigtriangledown_{x_i} H(y,x))\mu_i \\ \bigtriangledown_{y_i} f_i(x,y) - (\bigtriangledown_{y_i} H(y,x))\mu_i \\ H(y_i,x) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ y_i \\ \mu_i \end{bmatrix}. \end{aligned} \tag{19}$$

Note that in this MCP the constraint H and the variable y are replicated N times. The size of the MCP is $(n + 2mN)$. This reformulation is obtained by specifying the option `ImplVarModel=replication` in the JAMS options file.

The second reformulation strategy involves *switching* each shared variable with the multiplier associated with its defining equation. This technique can be applied if all of the following three conditions are met:

1. The defining constraint is given as an equation.
2. The dimension of the range of the defining constraint equals the dimension of the shared variable.
3. The shared variable is a free (unbounded) variable.

The switching strategy uses the fact that in an MCP, the matching between free variables and equations is somewhat arbitrary: it can be re-assigned without changing the solution. Applying this technique, we obtain the following MCP:

$$\begin{aligned} F(z) &= [(F_i(z)^T)_{i=1}^N, F_h(z)^T]^T, & z &= [(z_i^T)_{i=1}^N, z_h^T]^T \\ F_i(z) &= \begin{bmatrix} \bigtriangledown_{x_i} f_i(x,y) - (\bigtriangledown_{x_i} H(y,x))\mu_i \\ \bigtriangledown_{y_i} f_i(x,y) - (\bigtriangledown_{y_i} H(y,x))\mu_i \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \mu_i \end{bmatrix} \\ F_h(z) &= \begin{bmatrix} H(y,x) \end{bmatrix}, & z_h &= \begin{bmatrix} y \end{bmatrix}. \end{aligned} \tag{20}$$

Observe that in this MCP, the defining constraint H and the shared variable y appear only once. Thus the size of the problem is reduced to $(n + mN + m)$. The EMP framework will use this reformulation if the option `ImplVarModel=switching` is specified in the JAMS option file. This is currently the default strategy used.

The third strategy is selected by specifying option `ImplVarModel=substitution` in the JAMS option file. It uses the implicit function theorem to *substitute* the multipliers μ_i by new variables Λ_i, where $\Lambda_i = \bigtriangledown_{x_i} H(\bigtriangledown_y H)^{-1}$. This technique may be applied when the three conditions of the switching strategy are satisfied, and in addition, the implicit function theorem holds for the defining constraints. The basic idea is to regard the shared variable y as a function of other non-shared variables and apply the derivative. At each solution to the problem (y^*, x^*), there exists a locally defined implicit function $h_{x^*}(x)$ such that $y^* = h_{x^*}(x^*)$ and $H(h_{x^*}(x), x) = 0$ for each x in some neighborhood of x^*. It can be shown that given the implications of the implicit function theorem and the new variables Λ_i, the equilibrium problem can be reformulated as the MCP that follows. However, we omit a step-by-step mathematical derivation here, it is given in Kim & Ferris (2017) [109].

$$\begin{aligned} F(z) &= [(F_i(z)^T)_{i=1}^N, F_h(z)^T]^T, & z &= [(z_i^T)_{i=1}^N, z_h^T]^T \\ F_i(z) &= \begin{bmatrix} \bigtriangledown_{x_i} f_i(x,y) - (\bigtriangledown_{y_i} f_i(x,y))\Lambda_i \\ \bigtriangledown_y H(y,x)\Lambda_i - (\bigtriangledown_{x_i} H(y,x)) \end{bmatrix}, & z_i &= \begin{bmatrix} x_i \\ \Lambda_i \end{bmatrix} \\ F_h(z) &= \begin{bmatrix} H(y,x) \end{bmatrix}, & z_h &= \begin{bmatrix} y \end{bmatrix}. \end{aligned} \tag{21}$$

The size of this MCP is $(n + mn + m)$. It can be significantly reduced if the shared variable is *explicitly* defined, for example, $H(y, x) = y - h(x)$. In this case, $(\bigtriangledown_y H)^{-1}$ is the identity matrix, therefore it is not neccessary to introduce the variables Λ_i and the MCP takes the following form:

$$\begin{array}{ll} F(z) = [(F_i(z)^T)_{i=1}^N, F_h(z)^T]^T, & z = [(z_i^T)_{i=1}^N, z_h^T]^T \\ F_i(z) = \left[\bigtriangledown_{x_i} f_i(x,y) - \bigtriangledown_{x_i} H(y,x) \bigtriangledown_y f_i(x,y)\right], & z_i = \left[x_i\right] \\ F_h(z) = \left[H(y,x)\right], & z_h = \left[y\right]. \end{array} \tag{22}$$

The size of this MCP is $(n + m)$, which is a huge decrease compared to the other formulations. Note that the EMP framework detects automatically if the shared variable is explicitly defined and exploits this fact. In the following table an overview of the size of the MCP for the different reformulation techniques is given.

Strategy	Size of the MCP
Replication	$(n + 2mN)$
Switching	$(n + mN + m)$
Substitution (implicit)	$(n + mn + m)$
Substitution (explicit)	$(n + m)$

Table 2: Size of the Reformulated MCP For Different Reformulation Techniques

4.51.8.2 Equilibrium Problems with Shared Variables: A Simple Example

The following simple example computes a saddle point of the Lagrangian associated with a minimization over x subject to a single equality constraint with dual multiplier y. The GAMS EMP implementation is from Youngdae Kim. The primal (minimizing) and dual (maximizing) agents share the variable `L` containing the value of the Lagrangian, as well as its defining equation `defL`:

```
set i / 1*2 /;
variables
  L    'Lagrangian function: f(x) - y * h(x)'
  x(i) 'primal variables'
  y    'dual variable'
  ;
equation defL;
defL..
   L =e= sum{i, sqr(x(i)-1)} - y*(sum{i, x(i)} - 4);

model m / defL /;
file empinfo / '%emp.info%' /; putclose empinfo
  'equilibrium' /
  'implicit L defL' /
  '  min L x' /
  '  max L y' /
  ;

solve m using emp;
```

Note that the variable `L` is not indexed and it appears in the optimization problem of each agent. It is declared to be an implicit variable (using the keyword `implicit`) in the first line following the `equilibrium` keyword. Its defining constraint `defL` is listed only once, in this same line: it does not appear in the problem specification of each agent. By modeling the Lagrangian as a shared variable, its defining equation does not have to be replicated for each agent.

For other, more complex examples, both with shared decision variables and with a shared objective variable, see Kim & Ferris 2017 [109] .

4.51.9 Bilevel Programs

Bilevel programs are mathematical programs with optimization problems in their constraints. The main problem is called the *upper-level* problem or the *leader* and the nested problem is called the *lower-level* problem or the *follower*. A simple example is the bilevel programming problem that optimizes an upper-level objective over constraints that include a lower-level optimization problem. A famous example from economics is the Stackelberg game, where there is one leader and many followers. Bilevel programming is used in many areas, for example the design of optimal tax instruments: the tax instrument is modeled in the upper level and the clearing market is modeled in the lower level.

In this section we first present a mathematical formulation of a bilevel program with one follower and introduce the respective EMP annotations. Then we present three examples: a simple first example, an example with a variational inequality as the lower-level problem and an example with multiple followers that form a Nash equilibrium. We conclude the section with a short discussion of the general EMP syntax for bilevel programs.

4.51.9.1 Bilevel Programs: Mathematical Formulation

A bilevel program with one leader and one follower can be expressed as follows:

$$\begin{array}{lllll} \text{Min}_{x\in X,y} & f(x,y) & & & \\ \text{s.t.} & h(x,y) \leq 0 & & & \\ & y \quad \text{solves} & \text{Min}_y & g(x,y) & \\ & & \text{s.t} & k(x,y) \leq 0, & \end{array} \tag{23}$$

where

- $x \in \mathbb{R}^n$ are the upper-level variables,
- $y \in \mathbb{R}^m$ are the lower-level variables,
- $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the upper-level objective function,
- $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is the lower-level objective function,
- $h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^u$ are the upper-level constraints, and
- $k : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^l$ are the lower-level constraints.

Note that the upper-level constraints do not bind the lower-level decision maker.

This problem can be implemented with EMP as follows:

```
Sets  i, j ;
Variables x(i), objout, y(i), objin;
Equations deff(x,y), defg(x,y), defh(x,y), defk(x,y);

* Definitions of equations are omitted

Model bilevel / deff, defg, defH, defw /;

$onecho > "%emp.info%"
bilevel x
min objin y defg defk
$offecho

solve bilevel using EMP minimizing objout ;
```

Note that the variables and equations of the program are defined in GAMS in the usual way. The special bilevel structure of the model is specified in the EMP annotations. The EMP keyword `bilevel` indicates that this is a bilevel problem. The name and direction of the leader's objective variable is taken from the solve statement, but the other variables owned by the leader are listed after the `bilevel` keyword. The lower-level problem is specified next via the EMP keyword `min`, followed by the follower's objective variable `objin`. The other variables and the equations of the lower-level problem are listed next. Observe that this is exactly the same syntax that we introduced above for specifying optimization problems in the context of equilibrium problems.

The EMP tool reformulates the bilevel problem as a Mathematical Program with Equilibrium Constraints (MPEC) and passes this on to a subsolver (e.g. NLPEC or KNITRO). The reformulation is obtained by replacing the lower-level optimization problem by its KKT conditions, resulting in the following problem:

$$\begin{array}{lll} \text{Min}_{x\in X,y,\lambda} & f(x,y) & \\ \text{s.t.} & h(x,y) \leq 0 & \\ & k(x,y) \leq 0 & \\ & \lambda_i \leq 0, & i=1,\ldots,l \\ & \lambda_i k_i(x,y) = 0, & i=1,\ldots,l \\ & \bigtriangledown_y \mathcal{L}(x,y,\lambda) = 0, & \end{array} \tag{24}$$

where

$$\mathcal{L}(x,y,\lambda) = g(x,y) + \sum_{i=1}^{l} \lambda_i k_i(x,y) \tag{4.25}$$

is the Lagrangian function associated with the lower-level problem.

Note, however, that this reformulation is potentially problematic. KKT conditions require theoretical assumptions (like convexity) to be necessary and sufficient for *local optimality*. There may be cases where the lower lovel problem has multiple local solutions, but the modeler is interested in the *global* solution. Hence the reformulation above may not lead to the global solution, even if a global subsolver is used within the solver NLPEC.

Observe that there are two variations to problem (23) that the EMP framework is equipped to handle: the lower-level problem may be a variational inequality instead of an optimization problem, and there may be multiple lower-level problems (optimizing and/or VI) each behaving in a Nash manner. Examples follow in sections Bilevel Programs with EMP: A VI as Follower and Bilevel Programs with EMP: Multiple Followers respectively.

4.51.9.2 Bilevel Programs with EMP: A Simple Example

To illustrate, we use model **[BARD851]** from the GAMS EMP Library. Mathematically, the problem is

$$\begin{array}{lll} \text{Min}_{x,y_1,y_2} & (x-1)^2 + 2y_1^2 - 2x & \\ & x \geq 0 & \\ \text{s.t.} & (y_1,y_2) \quad \text{solve} & \begin{array}{ll} \text{Min}_{y_1,y_2} & (2y_1-4)^2 + (2y_2-1)^2 + xy_1 \\ \text{s.t} & 4x + 5y_1 + 4x_2 \leq 12 \\ & -4x + 5y_1 + 4x_2 \leq -4 \\ & 4x - 4y_1 + 5x_2 \leq 4 \\ & -4x + 4y_1 + 4x_2 \leq -4 \\ & y_1, y_2 \geq 0 \end{array} \end{array} \tag{25}$$

Note that this problem does not have any upper-level constraints. The GAMS code follows.

```
Positive Variables x, y1, y2;
Variables          objout, objin;

Equations defout, defin, e1, e2, e3, e4;

defout..  objout =e= sqr(x-1) + 2*sqr(y1) - 2*x;
defin..   objin  =e= sqr(2*y1-4) + sqr(2*y2-1) + x*y1;

e1..      4*x + 5*y1 + 4*y2 =l= 12;
e2..    - 4*x - 5*y1 + 4*y2 =l= -4;
e3..      4*x - 4*y1 + 5*y2 =l=  4;
e4..    - 4*x + 4*y1 + 5*y2 =l=  4;

Model bard / all /;

$echo bilevel x min objin y1 y2 defin e1 e2 e3 e4 > "%emp.info%"

solve bard use emp min objout;
```

The leader minimizes the variable `objout`, as specified in the solve statement. The EMP annotations specify the rest of the bilevel structure: the variable `x` belongs to the upper-level problem, and there is one lower-level agent or problem with `objin` minimized over the variables `y1` and `y2` subject to the constraints or equations `defin`, `e1`, `e2`, `e3` and `e4`.

Alternatively, we could write the EMP annotations as:

```
$echo bilevel x min objin * defin e1 e2 e3 e4 > "%emp.info%"
```

Here the lower-level variables are not listed: instead, the '$*$' indicates that all variables in the GAMS model not explicitly assigned to any agent are to be assigned to the follower.

The EMP Summary produced by JAMS gives the number of followers in the bilevel program:

```
--- EMP Summary
    Bilevel Followers   = 1
```

The GAMS EMP Library contains other bilevel examples, including several models `bard`$*$, the engineering models [**CCMG74**] and [**CCMG153**], and the simple nonconvex model [**MIRRLESS**]. The models [**JOINTC1**] and [**JOINTC2**] illustrate the interplay of the decision variables of the upper-level and lower-level problems.

4.51.9.3 Bilevel Programs with EMP: A VI as Follower

If the lower-level problem is a variational inequality, problem (23) will take the following form:

$$\begin{array}{ll} \text{Min}_{x,y} & f(x,y) \\ \text{s.t.} & h(x,y) \leq 0 \\ & y \text{ solves } \quad VI(F,K(x)), \end{array} \tag{26}$$

where

- $x \in \mathbb{R}^n$ are upper-level variables and $y \in \mathbb{R}^m$ are lower-level variables, and
- $K(x) \subseteq \mathbb{R}^m$ is a closed convex set, parametrized by x.

Consider the following simple example adapted from **[MULTMPEC]**:

$$\begin{array}{ll} \text{Min}_{w,z} & z \\ \text{s.t.} & e^z + w = 2 \\ & z \geq 1 \\ & w \text{ solves the } VI(F,K), \text{ where } K = \mathbb{R} \text{ and } F = w + z + 3 \end{array} \tag{27}$$

This bilevel problem can be modeled with EMP as follows:

```
variables w, z;
equations h, F;

h..  exp(z) + w =e= 2;
F..  w + z =n= -3;

z.lo = 1;

model bpvi / h, F /;

$onecho > %emp.info%
bilevel z
vi F w
$offecho

solve bpvi using emp min z;
```

Note that in the EMP annotations the lower-level problem is specified as a VI with the VI function `F` and the corresponding variable `w:` for details see section EMP Syntax for Variational Inequalities. The count of VI functions is indicated in the EMP Summary:

```
--- EMP Summary
    ...
    VI Functions        = 1
    ...
    Bilevel Followers   = 1
```

4.51.9.4 Bilevel Programs with EMP: Multiple Followers

The EMP framework allows multiple followers in a bilevel program. Taking the leader decisions as fixed, these followers behave as the agents in a Nash equilibrium. Consider the engineering example **[CCMG71]** from the GAMS EMP Library. In this bilevel program there are two followers, both solving minimization problems, specified in the EMP annotations as follows:

```
...
$onecho > %emp.info%
bilevel x1 x2 x3 x4
 min h1 u1 u2 u3 u4 defh1 e1
 min h2 v1 v2 v3 v4 defh2 e2
$offecho
...
```

The variables x1, x2, x3 and x4 are owned or controlled by the leader. The first follower minimizes the variable h1 over the variables u1, u2, u3 and u4 subject to the constraint/equation e1. The second follower is defined in a similar way. More details on how to specify optimization subproblems in the EMP annotations are given in section EMP Syntax for Equilibrium Problems.

Observe that in the EMP annotations of the actual model [**CCMG71**] the following shorthand notation is used:

```
...
$onecho > %emp.info%
bilevel x1 x2 x3 x4
min h1 * defh1 e1
min h2 * defh2 e2
$offecho
...
```

The symbol '∗' in the problem of the first follower indicates that this agent will optimize over all variables appearing in equations defh1 and e1 (i.e. the equations it controls) that are not claimed by another agent. In this case, the respective variables are u1, u2, u3 and u4. N.B.: in this example, if a variable appears in the equations for both followers the problem structure is not well defined and an error will result. To avoid confusion and promote clarity, we recommend that modelers explicitly list all the variables in each agent's problem, as shown in the first version of the EMP annotations above.

The number of followers is indicated in the EMP Summary:

```
--- EMP Summary
    ...
    Bilevel Followers   = 2
```

The **GAMS EMP Library** contains several bilevel problems with several followers, e.g. the well-known transportation model with a variable demand function cast as a bilevel problem with one optimization follower and one VI follower [**TRANSBP**], a simple example with two VI followers [**MULTMPEC**], and the spatial equilibrium Stackelberg model [**HARK-STACK**].

4.51.9.5 EMP Syntax for Bilevel Programs

The general syntax that EMP provides to specify bilevel programs in the EMP annotations file emp.info is as follows:

```
Bilevel {var}
{MAX|MIN obj {var|*} {[-] equ}}
{VI {var|*} {[-] equ var} {[-] equ}}
{Dualvar {var [-] equ}}
```

A bilevel program is declared with the EMP keyword bilevel, followed by the decision variables of the upper-level problem. The other specifications refer to the lower-level problems. These lower-level followers are optional, but at least one follower has to be specified. Optimization problems are defined using the EMP keyword max or min followed by the objective variable obj and the decision variables and the equations that implement the constraints of the problem. Variational inequalities are introduced with the EMP keyword vi: details of the specification are given in section EMP Syntax for Variational Inequalities. The dualvar keyword is used in the same way as it is for equilibrium problems.

Note

> While the EMP framework allows the symbol '∗' to be used to specify an automatically-created list of variables, we recommend against using the '∗' in the annotations file. Instead, list all variables explicitly to make the structure of the model clear and unambiguous to both human and machine readers of the EMP info file.

4.51.10 Disjunctive Programming

Disjunctive programming is an alternative modeling approach to mixed integer programming. Both mathematical programs model optimization problems that involve discrete and continuous variables. The advantage of disjunctive programming is that it retains and exploits the inherent logic structure of problems and thus reduces the combinatorics and improves the relaxations by using Boolean variables and disjunction definitions for modeling discrete choices.

Disjunctive programs have many applications, including ordering of tasks in a production process, organizing complex projects in a time saving manner and choosing the optimal route in a circuit.

In this section we first present the mathematical formulation of Generalized Disjunctive Programs (GDPs) and then demonstrate how disjunctive programs are implemented with GAMS EMP using two simple examples. Note that both examples are adapted from the `LogMIP 2.0 User's Manual`. At the end of the section we introduce and discuss the syntax for EMP annotations for disjunctive programs.

For information on the development of disjunctive programming in GAMS EMP and its connection to the solver LogMIP, see the respective section in the solver manual of the solver JAMS.

4.51.10.1 Generalized Disjunctive Programs (GDPs)

A GDP has Boolean and continuous variables, algebraic constraints that need to be satisfied regardless of the discrete choices, disjunctions that represent the discrete choices, and logic propositions that contain the logic relationships between the Boolean variables. Mathematically, the general structure of a GDP may be expressed as follows:

$$\begin{array}{lll} \text{Min} & f(x) & \text{Objective Function} \\ \text{s.t.} & g(x) \leq 0 & \text{Algebraic Constraints} \\ & \bigvee_{i \in D_k} \begin{bmatrix} Y_{ik} \\ r_{ik}(x) \leq 0 \end{bmatrix}, k \in K & \text{Disjunctions} \\ & \Omega(Y) = \text{True} & \text{Logic Propositions} \\ & L \leq x \leq U,\ x \in \mathbb{R}^n, & \text{Continuous Variables} \\ & Y_{ik} \in \{\text{True, False}\} & \text{Boolean Variables} \end{array} \tag{28}$$

where:

- $f : \mathbb{R}^n \to \mathbb{R}$ is a function, x is a vector of continuous variables with bounds L and U.
- $g : \mathbb{R}^n \to \mathbb{R}^l$ represents the set of global constraints.
- Each disjunction $k \in K$ is composed of a number of terms $i \in D_k$ that are connected by the Boolean operator `OR` ($\bigvee$).
- Each term $i \in D_k$ consists of a Boolean variable Y_{ik} and a set of inequalities $r_{ik}(x) \leq 0$, $r_{ik} : \mathbb{R}^n \to \mathbb{R}^j$. If Y_{ik} is true, then $r_{ik}(x) \leq 0$ is enforced, otherwise these constraints are ignored.
- $\Omega(Y) = \text{True}$ are logic propositions for the Boolean variables Y_{ik} expressed in the Conjunctive Normal Form $\Omega(Y) = \bigwedge_{t=1,2,\ldots,T}[\bigvee_{Y_{jk} \in R_t}(Y_{jk}) \bigvee_{Y_{jk} \in Q_t}(\neg Y_{jk})]$, where for each clause $t \in 1, \ldots, T$, R_t is the subset of Boolean variables that are non-negated and Q_t is the subset of Boolean variables that are negated.

N.B.: we assume that each disjunction is an exclusive-or, so that for each k exactly one variable Y_{ik} is true. Put another way, we assume the logic constraints $\underline{\bigvee}_{i \in D_k} Y_{ik}$ are contained in $\Omega(Y) = \text{True}$.

There are three cases of disjunctive programs: the functions f, g and r are linear, some of them are nonlinear, but convex, and some of them are nonlinear and nonconvex. Note that currently GAMS EMP facilitates modeling only the first two cases.

4.51.10.2 Disjunctive Programming with EMP: Example with No Algebraic Constraints

Consider the following simple example, that has no algebraic constraints which must be satisfied regardless of the disjunctive choices:

$$
\begin{array}{lll}
\text{Minimize} & c + 2x_1 + x_2 & \text{Objective Function} \\
\text{subject to} & & \\
& \begin{bmatrix} Y_1 \\ -x_1 + x_2 + 2 \leq 0 \\ c \leq 5 \end{bmatrix} \vee \begin{bmatrix} Y_2 \\ 2 - x_2 \leq 0 \\ c \leq 7 \end{bmatrix} & \text{Disjunctions} \\
& \begin{bmatrix} Y_3 \\ x_1 - x_2 \leq 0 \end{bmatrix} \vee \begin{bmatrix} \neg Y_3 \\ x_1 \leq 1 \end{bmatrix} & \\
& Y_1 \wedge \neg Y_2 \Rightarrow \neg Y_3 & \text{Logic Propositions} \\
& Y_2 \Rightarrow \neg Y_3 & \\
& Y_3 \Rightarrow \neg Y_2 & \\
& 0 \leq x_1 \leq 5 & \text{Continuous Variables} \\
& 0 \leq x_2 \leq 5 & \\
& c \geq 0 & \\
& Y_j \in \{\text{True, False}\},\ j = 1, 2, 3 & \text{Boolean Variables}
\end{array}
\tag{29}
$$

Observe that there are two disjunctions, each with two terms. In the first disjunction, each term is governed by a different Boolean variable: the first term is active if Y_1 is true and the second term is active if Y_2 is true. In the second disjunction, both terms are governed by the Boolean variable Y_3: the first term applies if Y_3 is true and the second term applies if Y_3 is false.

Note that the logic propositions imply that if Y_1 is true and Y_2 is false , then Y_3 must be false, and that Y_2 and Y_3 cannot both be true.

This example can be implemented in GAMS EMP as follows:

```
Set i / 1*2 /
    j / 1*3 /;

Positive Variables x(i), c;
Variable           z;
Binary Variables   y(j);

x.up(i) = 5;
c.up    = 7;

Equations Obj, Eq1, Eq2, Eq3, Eq4, Eq5, Eq6;

Obj..   z =e= c + 2*x('1') + x('2');

* Equations for Disjunctions
Eq1..   x('2') - x('1') =l=  - 2;
Eq2..   c =l= 5;
Eq3..   x('2') =g= 2;
Eq4..   c =l= 7;
Eq5..   x('1') - x('2') =l= 1 ;
Eq6..   x('1') =l= 1;
```

```
* Equations for Logic Propositions
Logic Equations  LEq1, LEq2, LEq3;

LEq1..   y('1') and not y('2') -> not y('3');
LEq2..   y('2') -> not y('3');
LEq3..   y('3') -> not y('2');

Model small1 / all /;

File emp / '%emp.info%' /;
put emp;
$onput
disjunction y('1') Eq1 Eq2 elseif y('2') Eq3 Eq4
disjunction y('3') Eq5 else Eq6
$offput
putclose;

Option optcr = 0.0;

solve small1 using EMP minimize z;
```

Note that in this model the Boolean variables Y_j are implemented as GAMS binary variables, the inequalities in the terms of the disjunctions are formulated as GAMS equations, and the logic propositions are expressed as GAMS logic equations. The disjunctive structure of the model is specified in the EMP annotations file `emp.info`. This file contains two lines (one per disjunction), each starting with the EMP keyword `disjunction` and specifying the structure and content of its disjunction. In the first line or disjunction, the binary variable `y('1')` that governs the first term is followed by the two equations contained in this term. The EMP keyword `elseif` denotes the start of a new term, here governed by the binary variable `y('2')` listed next and containing the two equations `Eq3` and `Eq4`. Similarly, in the second line, the binary variable `y('3')` that governs the first term of the second disjunction is followed by the equation `Eq5` contained in that term. As the binary variable governing the second term is just the negation of `y('3')`, the keyword `else` is enough to specify this and is followed by the equation `Eq6` of the second term.

Note

> Much more complex logical constructs for disjunctions are possible. For details, see section EMP Syntax for Disjunctive Programming below.

Finally, note that the model type in the solve statement is `EMP`.

Given the annotations in the file `emp.info`, the solver JAMS reformulates the model as a MIP (Mixed Integer Programming) model and passes it to a subsolver. By default, the convex hull relaxation is used for the reformulation, but users may choose reformulations that use big M or indicator constraints: see section EMP Syntax for Disjunctive Programming for details.

Observe that the listing file will contain some additional information if the model type EMP is used. The *EMP Summary* and the *Disjunction Summary* may be particularly useful. The respective listings for our example model follow:

```
--- EMP Summary
    Logical Constraints = 3
    Disjunctions        = 2
...
...
--- Disjunction Summary
    Disjunction 1 Term 2 is active
    Disjunction 2 Term 2 is active
```

Note that the EMP summary lists the number of logic constraints and disjunctions and the disjunction summary reports which terms of the disjunctions are active in the optimal solution.

4.51.10.3 Disjunctive Programming with EMP: Example with No Logic Propositions

Consider the following simple example:

$$
\begin{array}{lll}
\text{Min} & t & \text{Objective Function} \\
\text{s.t.} & t \geq x_A + 8 & \text{Algebraic Constraints} \\
 & t \geq x_B + 5 & \\
 & t \geq x_C + 6 & \\
 & \begin{bmatrix} Y_1 \\ x_A - x_C + 5 \leq 0 \end{bmatrix} \vee \begin{bmatrix} \neg Y_1 \\ x_C - x_A + 2 \leq 0 \end{bmatrix} & \text{Disjunctions} \\
 & \begin{bmatrix} Y_2 \\ x_B - x_C + 1 \leq 0 \end{bmatrix} \vee \begin{bmatrix} \neg Y_2 \\ x_C - x_B + 6 \leq 0 \end{bmatrix} & \\
 & \begin{bmatrix} Y_3 \\ x_A - x_B + 5 \leq 0 \end{bmatrix} \vee \begin{bmatrix} \neg Y_3 \\ x_B - x_A \leq 0 \end{bmatrix} & \\
 & t, x_A, x_B, x_C \geq 0 & \text{Continuous Variables} \\
 & Y_j \in \{\text{True, False}\},\ j = 1,2,3 & \text{Boolean Variables}
\end{array}
\tag{30}
$$

As this example has no logic propositions, the binary variables would not appear in any equations of a GAMS formulation like the one above. As a result, the binary variables would not be part of the model and any EMP annotations file that mentions these variables would be rejected by the EMP solver. To avoid this problem, we can introduce a dummy equation and include it in the EMP model: this ensures that the binary variables will be part of the GAMS model. The respective code follows:

```
set i / A, B, C /
    j / 1*3 /
    ;
positive variables x(i), t;
binary variables   y(j);
variable           z;

equations obj,  alg1, alg2, alg3,
          d1t1, d1t2, d2t1, d2t2, d3t1, d3t2
          dummy;

obj..  z =e= t;

* common algebraic equations
alg1..  t =g= x('A') + 8;
alg2..  t =g= x('B') + 5;
alg3..  t =g= x('C') + 6;

* equations for disjunctions
d1t1..  x('A') - x('C') + 5 =l= 0;
d1t2..  x('C') - x('A') + 2 =l= 0;
d2t1..  x('B') - x('C') + 1 =l= 0;
d2t2..  x('C') - x('B') + 6 =l= 0;
d3t1..  x('A') - x('C') + 5 =l= 0;
d3t2..  x('B') - x('C')     =l= 0;

* dummy equation
dummy..   sum(j, y(j)) =g= 0;
```

```
model small2 / all /;
file emp / '%emp.info%' /;

putclose emp
"disjunction y('1') d1t1 else d1t2" /
"disjunction y('2') d2t1 else d2t2" /
"disjunction y('3') d3t1 else d3t2" /
;
option optcr = 0.0;
solve small2 using EMP minimize z;
```

Apart from the dummy equation, this formulation is very similar to the formulation in the first example above.

EMP also supports an alternative formulation for models that have no logic equations, a formulation using *implicit default binary variables*. These variables are denoted in the EMP annonations with the star symbol '∗', which is internally replaced by a default binary variable. Note that this alternative model does not contain any explicit binary variables `Y(j)` and hence the dummy equation may be omitted, as in the following GAMS code which can be added to the model above:

```
model small3  'no dummy equation needed' / small2 - dummy /;

putclose emp
"disjunction * d1t1 else d1t2" /
"disjunction * d2t1 else d2t2" /
"disjunction * d3t1 else d3t2" /
;
solve small3 using EMP minimize z;
```

4.51.10.4 EMP Syntax for Disjunctive Programming

The general syntax that EMP provides to write disjunctions to the EMP annotations file `emp.info` is as follows:

```
Disjunction [chull [chull eps] | bigM [big M value] | indic]
            [NOT] var|* [NOT] {equ} {ELSEIF [NOT] var|* [NOT] {equ}} [ELSE [NOT] {equ}]
```

The EMP keyword `Disjunction` is mandatory, it indicates that what follows is a disjunction. The three constructs that follow are optional and relate to the three possible reformulations: convex hull (`chull`), big M method (`bigM`) or indicator constraints (`indic`). Note that in the the sequencing model **[SEQUENCE]** all three options are implemented. Note further, that currently, indicator constraints can only be handled by the solvers CPLEX, SCIP and XPRESS.

Note

- By default, the convex hull reformulation method is used.
- A different reformulation method may be used for each disjunction.

Observe that for the convex hull reformulation, the value of the parameter `epsilon` may optionally be specified. This parameter is an upper bound to check for constraint satisfaction, the default value is 0.0001.

Note that for the big M method, the value of `M` may optionally be specified. The value of `M` should be large enough to relax the constraint, but it should not be too large, to avoid infeasible solutions. The default value is 10000.

Following the EMP keyword `disjunction`, the first mandatory entry is a specification of the variable that governs the disjunction: `[NOT] var|*`. This may be either a binary variable, a negated binary variable or the symbol '*', that is internally replaced by default binary variables. For an example and more details on the symbol '*' in this context, see section Disjunctive Programming with EMP: Example with No Logic Propositions. Further, {`equ`} denotes a set of GAMS equation names that must be satisfied if the first disjunction term is selected. The remainder of the syntax is self-explanatory.

Alternatively, the following syntax may be used:

```
Default [chull [chull eps] | bigM [big M value] | indic]
Disjunction  [NOT] var|* [NOT] {equ} {ELSEIF [NOT] var|* [NOT] {equ}} [ELSE [NOT] {equ}]
```

Note that the first line is optional and serves to specify the reformulation method. The second line is identical to the first formulation of the general syntax where the specification of the reformulation method is omitted. Some users find this alternative syntax clearer, since the specification of the reformulation method and the disjunction are separated.

In addition to the sequencing model [**SEQUENCE**] that was already mentioned, the GAMS EMP Library has two other models for disjunctive programming: the manufacturing problem [**FOODEMP**] and the job scheduling problem [**MAKESPAN**]

4.51.11 Stochastic Programming

Stochastic programs are mathematical programs that involve data that is not known with certainty. Deterministic programs are formulated with fixed parameters, whereas real world problems frequently include some uncertain parameters. Often these uncertain parameters follow a probability distribution that is known or can be estimated. Thus stochastic programs approximate unknown data by probability distributions. The goal is to find some policy that is feasible for all (or almost all) the possible data instances and that maximizes the expectation of some function of the decision variables and the random variables. The EMP framework includes an extension for stochastic programming that allows users to model various stochastic problems as deterministic models, while information about the stochastic structure of the problem, like probability distributions for some data, is specified in the EMP annotations. Thus formulating stochastic programs becomes straightforward.

In the remainder of this chapter we discuss the stochastic programming extension of GAMS EMP. We introduce the basics of stochastic programming with EMP using a two-stage stochastic model and then show how the logic can be extended to multi-stage stochastic problems. In most stochastic problems the expected value of the objective is optimized. The EMP framework facilitates optimizing two additional risk measures: Value at Risk (VaR) and Conditional Value at Risk (CVaR). These alternative risk measures are discussed next. Another type of stochastic programming includes constraints that hold only with certain probabilities. These constrains are called chance constraints (or *probabilistic constraints*), they are the topic of the last section on stochastic programming. At the end of each section we give an overview of the EMP annotations that are specific to each topic: stochastic programming with recourse, additional risk measures, and chance constraints. A summary of all EMP keywords for stochastic programming is given in section GAMS EMP Keywords for Stochastic Programming.

4.51.11.1 Stochastic Programming with Recourse

One way to think about stochastic problems is to require the decision maker to make a decision now and then to minimize the expected costs of the consequences of that decision. This paradigm is called the *recourse* model. The simplest form of the recourse model has two stages: a decision is made in the first stage, then the realization of the uncertain parameters is revealed at the start of the second stage and recourse actions can be taken given this new information. This simple model can be extended to include more stages. In a *multistage* problem, a decision is made in the first stage, then some uncertainty is resolved in the second stage and another decision is made based on this new knowledge, then some other uncertainty is resolved and so on. The objective is to minimize the expected costs of the decisions in all stages.

This section is organized as follows. We start with a mathematical formulation of the two-stage stochastic problem with recourse, then show how such problems can be modeled with EMP using a simple example. The uncertain data in this first example follows a discrete distribution, there are just three different scenarios. Continuous distributions are more complex to model. Mostly, they are approximated to discrete distributions by sampling, which is discussed next. The second example illustrates how the modeling approach for a simple two-stage problem can be extended to solve a multi-stage problem. We conclude this section with a description of the EMP annotations for stochastic programs with recourse.

Two-Stage Stochastic Programs: Mathematical Formulation

The simplest form of a stochastic program is the two-stage stochastic linear program with recourse. In mathematical terms it is defined as follows.

Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ be two variables and let the set of all realizations of the unknown data be given by Ω, $\Omega = \{\omega_1, \ldots, \omega_S\} \subseteq \mathbb{R}^r$, where r is the number of the random variables representing the uncertain parameters. Then the stochastic program is given by

$$\begin{array}{lllllllll} & \text{Min}_x & z = & c^T x & + & \mathbb{E}[Q(x,\omega)] & & & \\ & \text{s.t.} & & Ax & & & = & b, & x \geq 0, \\ \text{where} \quad Q(x,\omega) = & \text{Min}_y & & & & q_\omega^T y(\omega) & & & \\ & \text{s.t.} & & T_\omega x & + & W_\omega y(\omega) & = & h_\omega, & y(\omega) \geq 0, \quad \forall \omega \in \Omega. \end{array} \tag{31}$$

The first two lines define the *first-stage* problem and the last two lines define the *second-stage* problem. In the first stage, x is the decision variable, c^T represents the cost coefficients of the objective function and $\mathbb{E}[Q(x,\omega)]$ denotes the expected value of the optimal solution of the second stage problem. In addition, A denotes the coefficients and b the right-hand side of the first stage constraints. In the second stage, y is the decision variable, T represents the transition matrix, W the recourse matrix (cost of recourse) and h the right-hand side of the second stage constraints. Note that all parameters and the decision variable of the second stage are dependent on the specific realization of the stochastic data ω. The objective variable z is also a random variable, since it is a function of ω. As a random variable cannot be optimized, stochastic solvers automatically optimize the *expected value* of the objective variable z. Note that the EMP framework allows other risk measures to be optimized in addition to the expected value. This is discussed below.

In the first stage, a decision has to be made before the realization of the uncertain data is clear. The optimal solution of the first stage is fixed and only then it will become known which values the uncertain parameters will take. Given the fixed solution of the first stage and the new data, recourse action can be taken in the second stage and the optimal solution determined. Each possible realization of the uncertain data is represented by $\omega_s \in \Omega$ and is called a *scenario*. The objective is to find a feasible solution x that minimizes the total cost, namely the sum of the first-stage costs and the expected second-stage costs.

One of the most common methods to solve a two-stage stochastic LP is to build and solve the *deterministic equivalent*. Assume that the uncertain parameters follow a (finite) discrete distribution and that each

scenario ω_s occurs with probability $P(\omega_s) = p_s$ for all $s = 1, \ldots, S$ and $\sum_s p_s = 1$. Thus $\mathbb{E}[Q(x,\omega)] = \sum_s p_s q^T y_s$, where y_s denotes the optimal second-stage decision for the scenario ω_s. Then the deterministic equivalent can be expressed as follows:

$$\begin{array}{lllllllllll} c^T x & + & p_1 q^T y_1 & + & p_2 q^T y_2 & + & \ldots & + & p_S q^T y_S & & \\ \text{s.t.} & & & & & & & & & & \\ Ax & & & & & & & & & = & b \\ T_1 x & + & W_1 y_1 & & & & & & & = & h_1 \\ T_2 x & & & + & W_2 y_2 & & & & & = & h_2 \\ \vdots & & & + & & & \ddots & & & & \vdots \\ T_S x & & & & & & & + & W_S y_S & = & h_S \\ x \in \mathbb{R}^n & & y_1 \in \mathbb{R}^m & & y_2 \in \mathbb{R}^m & & & & y_S \in \mathbb{R}^m & & \end{array} \tag{32}$$

Note that for stochastic linear programs the deterministic equivalent is just another (potentially very large) linear program.

A GAMS EMP stochastic model has three parts: the *core model*, EMP annotations and the *dictionary*, which contains output-handling information. The *core model* is a deterministic model, where the uncertain data is given as fixed parameters. Often, the expected value of the probability distribution is chosen.

The *annotations* contain information about the stochastic features of the model: a specification of the random variables and their distributions, details about the stages and possibly directions concerning sampling.

Given the probability distributions of the random variables, the solvers of stochastic programming models create various scenarios and evaluate them. In the *dictionary* users specify which results from each scenario are to be stored in standard parameters in order to be accessed later. These three parts of a GAMS EMP stochastic model will become much clearer as they are illustrated in the next subsection.

A Simple Example: The News Vendor Problem

Consider the following simple example adapted from the news vendor model **[NBSIMPLE]**. A news vendor has to decide early in the morning how many newspapers to buy from a distributor on a particular day in order to sell them to his customers. He knows from experience that the demand will be 45 in 70% of all cases, 40 with a probability of 20% and 50 with a probability of 10%. If the demand is less than the number of newspapers he bought in the morning, the left-over newspapers will be stored in an inventory at a cost per unit. If the demand exceeds his expectations and there are more customers than newspapers, then he will have to pay a penalty. He aims to maximize his profit by selling the newspapers at a higher price than he has bought them. Mathematically, the problem can be expressed as follows:

$$\begin{array}{lllll} & \text{Max}_x & Z(x,D) = -cx + \mathbb{E}[Q(x,D)], & x \geq 0, & \\ \text{where} \quad Q(x,D) = & \text{Max}_{s,i,l} & vs_D - hi_D - pl_D & & \\ & \text{s.t.} & x - s_D - i_D = 0 & & \\ & & s_D + l_D = D, & s_D, i_D, l_D \geq 0 & \end{array} \tag{33}$$

where

- the variable x denotes the number of newspaper bought in the morning,
- c is the cost per newspaper,
- D is a random variable that denotes the uncertain demand; the set of all realizations of D is given by $\Omega = \{d_1, d_2, d_3\}$, with $d_1 = 45$, $d_2 = 40$ and $d_3 = 50$, and the respective probabilities $p_1 = 0.7$, $p_2 = 0.2$ and $p_3 = 0.1$,

- the variable s denotes the number of newspapers that are sold,
- v is the selling price,
- the variable i denotes the number of newspapers that could not be sold, in case the demand turned out to be less than expected,
- h is the holding cost in the inventory per unit,
- the variable l denotes the unsatisfied demand, in case the demand turned out to be higher than expected,
- p is the penalty per unit of unsatisfied demand.

Note that x is the first-stage decision variable and the variables s, i and l are the second-stage decision variables. The second-stage decision variables are dependent on D. An overview of the stages in this problem is given in the following table:

1st Stage Decision Variable	**2nd Stage Decision Variable;** $\Omega = \{d_1, d_2, d_3\}$	**Probabilities**
x	$y_{d_1} = (s_{d_1}, l_{d_1}, i_{d_1})$	Scenario 1: $p_1 = 0.7$
x	$y_{d_2} = (s_{d_2}, l_{d_2}, i_{d_2})$	Scenario 2: $p_2 = 0.2$
x	$y_{d_3} = (s_{d_3}, l_{d_3}, i_{d_3})$	Scenario 3: $p_3 = 0.1$

Table 3: Two stages in the News Vendor Problem

Observe that the decision x has to be made before the realization of the demand D is known. Given the uncertainty of the demand, we aim to maximize the *expected value* of the profit, denoted by $\mathbb{E}[Z(x, D)]$. The expected value of the profit is the profit *on average*. Note that as we have a finite number of scenarios and their probabilities are known, the expected value of the profit $\mathbb{E}[Z(x, D)]$ can be expressed as a weighted sum:

$$\begin{aligned} \mathbb{E}[Z(x,D)] = \quad & -cx + \mathbb{E}[Q(x,D)] \\ & -cx + \textstyle\sum_{k=1}^{3} p_k Q(x, d_k). \end{aligned} \tag{34}$$

Note that in this example there are no first stage equations $Ax = b$.

The *core model* is the first part of the respective GAMS EMP model:

```
Variable  z      "profit";
Positive Variables
          x      "units bought"
          i      "inventory"
          l      "lost sales"
          s      "units sold";

Scalars   c      "purchase costs per unit"          / 30 /
          p      "penalty shortage cost per unit"   /  5 /
          h      "holding cost per leftover unit"   / 10 /
          v      "revenue per unit sold"            / 60 /
          d      "demand, random parameter"         / 45 /;

Equations profit "profit to be maximized"
          row1   "demand = UnitsSold + LostSales"
          row2   "inventory = UnitsBought - UnitsSold";
```

```
profit..  z =e= v*s - c*x - h*i - p*l;
row1..    d =e= s + l;
row2..    i =e= x - s;

Model nv / all /;
```

Observe that the model is defined in the usual way. In particular, the demand `d` is modeled as a fixed parameter.

The EMP annotations are the second part. Here we specify the stochastic structure:

```
File emp / '%emp.info%' /;
put emp '* problem %gams.i%'/;
$onput
randvar d discrete 0.7 45 0.2 40 0.1 50
stage 2 i l s d
stage 2 Row1 Row2
$offput
putclose emp;
```

First, we define the parameter `d` to be a random variable using the EMP keyword `randvar` and we specify the probability distribution. The EMP keyword `discrete` indicates that `d` follows a discrete distribution: with probability 0.7 it takes a value of 45, with probability 0.2 it takes a value of 40, and with probability 0.1 it takes a value of 50.

Note

> If the sum of the probabilities of a discrete distribution is smaller or larger than 1, the EMP framework will automatically normalize the probabilities so that the sum equals 1 and a corresponding remark will appear in the log and listing file.

Observe that continuous distributions are also possible, see section Random Variables with Continuous Distributions below for details.

Secondly, we specify which variables and equations belong to stage 2 using the EMP keyword `stage`. Note that the variables and equations that are not listed in the annotations are automatically assigned to a stage, by default stage 1 (this could be changed using the EMP keyword stageDefault). Only the objective variable (in this case `Z`) and the objective equation (`profit`) are assigned to the highest stage specified instead (stage 2 in this example). Observe that Z is in fact a random variable since it is a function of the random variable D. As such it cannot be optimized directly, EMP implicitly maximizes the *expected value* of Z. This might lead to some confusion since the expected value of Z belongs to `stage 1`. We show in section Expected Value Revisited how to specify more clearly the fact that we are maximizing $\mathbb{E}(Z)$.

All keywords that can be used in EMP annotations in the context of stochastic programming are introduced in subsequent examples and they are summarized in section GAMS EMP Keywords for Stochastic Programming.

The dictionary with output-handling information and the solve statement are the third part of the model. After solving a stochastic programming model, only the solution of the expected value problem may be accessed via the regular `.l` and `.m` fields. The results of the scenarios that were created and evaluated by the stochastic solvers may be stored in standard parameters in the following way:

```
Set scen           "scenarios" / s1*s3 /;
Parameter
     s_d(scen)     "demand realization by scenario"
     s_x(scen)     "units bought by scenario"
     s_s(scen)     "units sold by scenario"
     s_rep(scen,*) "scenario probability"   / #scen.prob 0/;

Set dict / scen .scenario.''
           d    .randvar .s_d
           s    .level   .s_s
           x    .level   .s_x
           ''   .opt     .s_rep /;

solve nv max z use EMP scenario dict;
display s_d, s_x, s_s, s_rep;
```

The size of the set `scen` defines the maximal number of scenarios we are willing to store results for. It does not have to match the number of scenarios that are actually generated in the solution process. Assume that the size of the set `scen` is n and n is smaller than the number of generated scenarios. In this case only the results for the first n scenarios will be stored. On the other hand, if the size of `scen` is larger than the number of generated scenarios, then the positions of the surplus elements of `scen` will be empty in the parameters (e.g. `s_d`).

The elements of the three-dimensional set `dict` determine which scenario-dependent values will be stored. The set contains mapping information between the symbols in the model in the first position and the symbols in which scenario solution information is stored in the third position. The type of storing is specified in the second position. The following entries are allowed:

Label	Description
`level`	Stores the level values of a scenario solution of a variable or equation.
`marginal`	Stores the marginal values of a scenario solution of a variable or equation.
`randvar`	Stores the realization of a random variable.
`opt`	Stores the probability of each scenario.
`scenario`	The symbol in the first position of the tuple is used as the scenario index.

Table 4 Entries in the second position of the tuples in the scenario dictionary

Attention

> The first two tuples in the set `dict` are mandatory, all other elements of the set are optional. If one of the first two tuples is missing, the stochastic model cannot be solved and an error will be generated.

In the example above, we store the realization for each scenario of the random variable `d` in the parameter `s_d`, the level values of the variables `s` and `x` in the parameters `s_s` and `s_x` respectively and the actual probability of each scenario in the parameter `s_rep`.

Note

> Storing and retrieving the actual probabilities of the scenarios may be particularly relevant if the probabilities have been normalized such that their sum equals 1.

Finally, the solve statement needs to be adjusted: we use the model type `EMP` and add `scenario dict` to indicate that a stochastic problem is to be solved.

Random Variables with Continuous Distributions

Now we assume that the random variable D in the example above has a continuous distribution, say a Normal distribution with mean 45 and standard deviation 10. The structure of the problem remains unchanged, the only difference is that the set Ω, that is the set of all realizations of D, contains an infinite number of scenarios. There are various ways of modeling this. For example, a sampling procedure implemented in the solver can be used: a finite number of scenarios is generated to approximate Ω and thus the problem is converted to a problem with a finite discrete distribution.

We can model a continuous distribution of the random variable in GAMS EMP by changing the annotations in the following way:

```
randvar d normal 45 10
```

Note that currently only the solver LINDO has implemented a sampling procedure for parametric distributions so that it can solve such a problem directly. If the solver DE is used, the sampling can be performed by the LINDO system behind the scenes. More details about sampling are given in the next section. All parametric distributions that can be modeled are listed in Table 5 .

Distribution	Parameter 1	Parameter 2	Parameter 3
Beta	shape 1	shape 2	
Cauchy	location	scale	
Chi_Square	deg. of freedom		
Exponential	lambda		
F	deg. of freedom 1	deg. of freedom 2	
Gamma	shape	scale	
Gumbel	location	scale	
Laplace	mean	scale	
Logistic	location	scale	
LogNormal	mean	std dev	
Normal	mean	std dev	
Pareto	scale	shape	
StudentT	deg. of freedom		
Triangular	low	mid	high
Uniform	low	high	
Weibull	shape	scale	
Binomial	n	p	
Geometric	p		
Hyper_Geometric	total	good	trials
Logarithmic	p-factor		
Negative_Binomial	failures	p	
Poisson	lambda		

Table 5: Parametric distributions supported by LINDO

Recall that there are two ways to specify that a specific solver should be used (that is not the default solver for the problem type): with a command line parameter (e.g. `emp=lindo`) and with an option statement (e.g. `option emp = lindo;`).

Examples with random variables that follow parametric distributions include the multistage example below (three random variables with Gamma distributions) and the production model [**PRODSP3**] in

the GAMS EMP Model Library (one random variable that follows a Normal distribution and one random variable that is uniformly distributed).

Sampling

Currently, only the solver LINDO has the ability to perform sampling for parametric distributions directly. Note that LINDO generates 6 samples by default and users need a LINDO license for sample sizes larger than 10 (smaller sample sizes are included in the demo version, but just for certain distributions). Also, changing the default variance reduction method requires a valid LINDO license. There are three ways to customize sampling: including additional information in the EMP annotations, generating a sample with the LINDO library `lsadclib` (these two options allow to pass on the samples to other solvers with EMP SP capabilities) and setting various options in the solver LINDO. We discuss each method in more detail in the next three subsections.

Customizing Sampling in the EMP Annotations

The EMP framework provides two keywords to facilitate sampling: `sample` and `setSeed`. The keyword `sample` allows users to specify the size of the sample in the EMP annotations. Consider the following example:

```
randvar d normal 45 10
sample d 9
```

The second line specifies that the size of the sample of the distribution of the random variable D is 9. Note that currently the LINDO Sampling Library is used for this sampling. If the solver DE is used and a parametric distribution for the random variable is specified, the second line with the keyword `sample` is *mandatory*. Otherwise the following error message will appear:

```
*** Only random variables with sampled continuous distributions supported.
```

In addition to specifying the size of the sample, information about a mathematical variance reduction method may be added in the line that starts with the keyword `sample`. Variance reduction is a procedure used to increase the precision of the estimated values from the distribution. LINDO provides three methods for reducing the variance: Monte Carlo sampling, Latin Square sampling and Antithetic sampling. We illustrate with an example.

Consider a stochastic model with four random variables: E, F, G and H. Assume that E follows a Normal distribution with mean 23 and standard deviation 5, F follows a Normal distribution with mean 37 and standard deviation 8, G is uniformly distributed on the interval $[0, 1]$ and H is binomially distributed with $n = 100$ and $p = 0.55$. We wish that three variance reduction methods are applied: Antithetic sampling is to be used for E and F, Monte Carlo sampling for G and Latin Square sampling for H. We insert the following lines in the EMP annotations:

```
randvar e normal 23 5
randvar f normal 37 8
randvar g uniform 0 1
randvar h binomial 100 0.55
sample e f 10 method1
sample g 12 method2
sample h 8 method3
```

First, the random variables and their distributions are defined. Next, details about the sampling procedures are given. Note that the keyword `sample` can take more than one random variable if the sample size and the variance reduction method for these random variables are identical. We need to add the following lines before the solve statement to specify the content of `method1`, `method2` and `method3` (we assume that the name of the model is `nv`):

```
$onecho > lindo.opt
SVR_LS_ANTITHETIC=method1
SVR_LS_MONTECARLO=method2
SVR_LS_LATINSQUARE=method3
$offecho
nv.optfile=1;
```

If Latin Square sampling should also be used for E and F, we would simply change the EMP annotations to replace the label `method1` with the label `method3`. For more details on variance reduction methods, please consult the LINDO manual.

In addition, the EMP keyword `setSeed` may be used to further customize the sampling procedure:

```
setSeed <seed>
```

This line sets the seed (an integer number) for the random number generator of the sampling routines. If `setSeed` is used in the EMP annotations, the seed will be set once before all samples are generated. Please note that `setSeed` only works with a valid LINDO license.

Separating Sampling and Solving

Users may want to sample from a distribution with the LINDO system and solve the model with another solver, say DE. This is possible with the sampling routines from the LINDO Sampling Library `lsadclib`. We could solve the news vendor model by first drawing a sample from a Normal distribution with mean 45 and standard deviation 10 and then using the sample in the EMP annotations. The GAMS code for sampling from a Normal distribution where the sample size is 9 follows:

```
$funclibin msllib lsadclib

Function setSeed         / msllib.setSeed /
         sampleNormal    / msllib.sampleLSNormal /
         getSampleValues / msllib.getSampleValues /;

Scalar k;
k = sampleNormal(45,10,9);

Set g /1*9/; parameter sv1(g);
loop(g,
   sv1(g) = getSampleValues(k);
);
display sv1;
```

The directive in the first line makes the LINDO sampling library available, `msllib` is the internal library name (see also $funcLibIn for more information). For further details and a list of the available probability distributions, see section The LINDO Sampling Library.

In the following lines we demonstrate how the sample is used in the EMP annotations:

```
File emp / '%emp.info%' /;
put emp;
put 'randvar d discrete '; loop(g, put (1/card(g)) ' ' sv1(g) ' ');
$onput
stage 2 I L S d
stage 2 Row1 Row2
$offput
putclose emp;
```

Note that the third line states that the random variable D follows a discrete distribution and the probabilities and values are taken from the previously generated sample. The other lines of the annotations remain unchanged.

Sampling Options for the Solver LINDO

There are some customizable sampling options in LINDO. Users could control the number of sampled scenarios by setting any of the following LINDO/SP options in the file LINDO options file `lindo.opt`:

```
STOC_NSAMPLE_PER_STAGE      - list of sample sizes per stage (starting at stage 2)
STOC_NSAMPLE_SPAR           - common sample size per stochastic parameter
STOC_NSAMPLE_STAGE          - common sample size per stage
```

For example, we could insert the following three lines before the solve statement:

```
option emp = lindo;
$echo STOC_NSAMPLE_STAGE = 100 > lindo.opt
nv.optfile = 1;
```

The first line directs GAMS to solve models of the type `emp` with the LINDO solver. The second line writes `STOC_NSAMPLE_STAGE = 100` to the file `lindo.opt`, it communicates to the solver to generate 100 samples per stage. The third line informs GAMS to use the solver option file (i.e. `lindo.opt`).

The following example shows how to use large samples and instruct LINDO to use a Benders decomposition algorithm to tackle the problem:

```
$onecho > lindo.opt
STOC_MAX_NUMSCENS = 1000000
STOC_NSAMPLE_STAGE = 40
STOC_METHOD = 1
$offecho
```

In the second line we ensure that the maximum number of scenarios is large enough. The options in the following lines state the sample size and determine the stochastic method to be used (1 means Nested Benders Decomposition). For further details on LINDO options, please consult the LINDO user manual.

Multistage Stochastic Programming Example

The modeling principles for two-stage stochastic models can be easily extended to multistage stochastic models. At the beginning of each stage some uncertainty is resolved and recourse decisions or adjustments are made after this information has become available. At the point where decisions are made only outcomes of the current stage and previous stages are available. This logic can be pictured schematically as follows:

$$\begin{aligned}\underbrace{\text{Make decision}}_{\text{Stage 1}} \rightarrow \underbrace{\text{Random Variable is realized} \rightarrow \text{Make decision}}_{\text{Stage 2}} \\ \rightarrow \cdots \rightarrow \underbrace{\text{Random Variable is realized} \rightarrow \text{Make decision}}_{\text{Stage n}}\end{aligned} \tag{4.26}$$

Observe that random variables which are realized in stage k are fixed parameters in stage $k+1$ and following; stage 1 random variables are in fact simply given deterministic values and will create a warning or an error dependent on the solver selected.

Consider an inventory problem. At the end of each week the decision how many hats should be bought in order to satisfy the stochastic demand in the following week must be made. The aim is to maximize the profit. We assume that the stochastic demand can be modeled using a Gamma distribution. The planning horizon is 3 weeks. *Before* the first week starts an initial purchase decision has to be made and the goods are stored in the inventory for use in week 1. At this point only the *distribution* of the demand of the first week is known. During the first week the *actual* demand is revealed and some items that were stored in the inventory are sold. Some items may be left over; they are stored as the inventory for the second week. In addition, a purchase decision for the second week has to be made given the size of the inventory and the *distribution* of the demand in the second week. Again, the *actual* demand is revealed in the course of the second week. The same holds for the third week.

We will model the problem with 4 stages, where the first stage corresponds to the preparation time before the first week, the second stage corresponds to decisions made in the first week, the third stage corresponds to decisions made in the second week and the fourth stage corresponds to decisions made in the third week. Let t denote the stages. Note that while the stages range from $t = 1$ to $t = 4$, demand variables are realized only at $t = 2$ to $t = 4$. Let y_t be the amount bought and i_t the amount stored at the end of each stage and let D_t denote the demand in each week and s_t the amount sold each week. Note that we denote the demand with a capital D since it is a random variable. Let $\alpha = 10$ be the cost per hat bought, $\beta = 20$ the revenue per hat sold and $\delta = 4$ the storage cost per unit. Further, in the storage facility a maximum of $\kappa = 5000$ hats can be stored.

Mathematically, this problem can be expressed as follows:

$$\begin{array}{ll}\text{Max}_{s_t,y_t,i_t} & z = -\alpha y_1 - \gamma i_1 + \mathbb{E}[\text{Max} \quad \beta s_2(D_2) - \alpha y_2(D_2) - \delta i_2(D_2) + \cdots + \\ & \mathbb{E}[\text{Max} \quad \beta s_4(D_4) + -\alpha y_4(D_4) - \delta i_4(D_4)] \dots] \\ \text{s.t.} & i_1 = y_1 \\ & i_{t-1} + y_t = s_t + i_t \\ & s_t \leq i_{t-1} \\ & s_t \leq D_t \\ & i_t \leq \kappa \\ & y_1, i_1, s_t, y_t, i_t \geq 0,\end{array} \tag{35}$$

where $t = 2, \dots, 4$ and D_t follows a Gamma distribution. Note that s_t, y_t and i_t depend on the realization of D_t.

As discussed above, the solvers use a sampling procedure to approximate a problem with a continuous random variable by a problem with a discrete distribution. Figure 1 illustrates the stages assuming a sample size of 6 (per stage). Note that this results in a total of $6^3 = 216$ scenarios.

The problem can be modeled with GAMS EMP as follows:

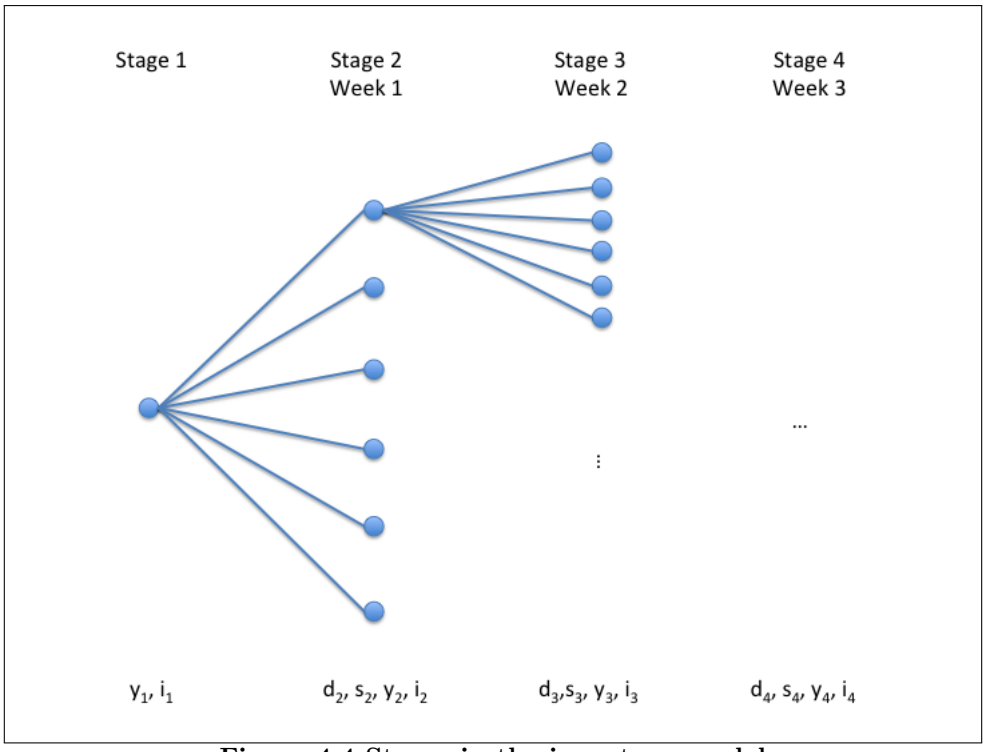


Figure 4.4 Stages in the inventory model

```
* Core Model
Set t             "stages" / 1*4 /;
Set st(t)         "stages where sales occur" / 2*4 / ;

Positive Variables
     y(t)         "units to be bought in time period t"
     i(t)         "ending inventory in period t"
     s(t)         "units sold in time period t" ;
Free Variable
     profit ;

Scalars
    kappa         "capacity of storage building"                    / 5000 /
    alpha         "cost per unit bought"                            /   10 /
    beta          "revenue per unit sold"                           /   20 /
    delta         "cost per unit stored at the end of time period" /    4 / ;

Parameters
     k            "shape of demand (1st parameter of gamma distribution)" / 16         /
     d_theta(st)  "scale of demand (2nd parameter of gamma distribution)" / 2 208.3125
                                                                            3 312.5
                                                                            4 125      /
     d(st)        "demand" ;

d(st) = k * d_theta(st);

Equations
     defprofit    "definition of profit"
     balance(t)   "balance equation"
     sales1(st)   "sales cannot exceed demand"
     sales2(t)    "sales cannot exceed inventory of previous time period" ;
```

```
defprofit..        profit =e= sum(t, beta * s(t)$st(t) - alpha * y(t) - delta * i(t));
balance(t)..       i(t-1) + y(t) =e= s(t)$st(t) + i(t) ;
sales1(st)..       s(st) =l= d(st);
sales2(t)$st(t)..  s(t) =l= i(t-1);

i.up(t) = kappa;
Model inventory /all/;

* EMP Annotations
File emp / '%emp.info%' /;
put emp; emp.nd=6;
put "randvar d('2') gamma ", k d_theta('2') /;
put "randvar d('3') gamma ", k d_theta('3') /;
put "randvar d('4') gamma ", k d_theta('4') /;
$onput
stage 1 y('1') i('1') balance('1')
stage 2 y('2') d('2') s('2') i('2') balance('2') sales1('2') sales2('2')
stage 3 y('3') d('3') s('3') i('3') balance('3') sales1('3') sales2('3')
stage 4 y('4') d('4') s('4') i('4') balance('4') sales1('4') sales2('4')
$offput
putclose emp;

* Dictionary
Set scen          "scenarios" / s1*s216 /;
Parameters
     s_d(scen,st) "demand realization by scenario"
     s_y(scen,t)  "units bought by scenario"
     s_s(scen,t)  "units sold by scenario"
     s_i(scen,t)  "units stored by scenario" ;

Set dict /
   scen .scenario.''
   d    .randvar .s_d
   s    .level   .s_s
   y    .level   .s_y
   i    .level   .s_i  /;

option emp=lindo;
solve inventory max profit using emp scenario dict;
display s_d, s_s, s_y, s_i;
```

Observe that in the core problem the values of the demand `d(t)` are replaced by the expected values of the random variable D_t which follows a Gamma distribution. Note that as expected, in stage 1 we have only the variables `y` and `i`, but no variables `s` and `d`. Recall that currently only the solver LINDO can solve models with parametric distributions, see sections Random Variables with Continuous Distributions and Sampling for more information.

EMP Syntax for Stochastic Programs with Recourse

The general syntax of the EMP annotations used to specify stochastic problems with recourse is as follows:

```
[randvar rv discrete prob val {prob val}]
[randvar rv distr par {par}]
[jrandvar rv rv {rv} prob val val {val} {prob val val {val}}]
```

```
[setSeed number]
[sample  rv1 [rv2 ... rvn] sampleSize [varRedMethod]]
stage stageNo rv | equ | var {rv | equ | var}
{stage stageNo rv | equ | var {rv | equ | var}}
[stageDefault stageNo]
```

The first three lines present three ways to specify random variables: a single random variable with a discrete distribution, a single random variable with a parametric distribution and joint random variables with discrete distributions. Note that `randvar`, `jrandvar` and `discrete` are EMP keywords. The distribution of single discrete random variables is defined by pairs of the probability `prob` of an outcome and the corresponding realization `val`, see example above. The distribution of parametric random variables is defined by the name of the distribution `dist` and the respective parameter(s) `par`. An overview of all supported parametric distributions can be found in Table 5. All possible values for `distr` and the related parameters `par` are listed there. The keyword `jrandvar` is used to define discrete random variables that are *jointly* distributed. At least two random variables must be named. For an example, see the news vendor model [**NBDISCJOINT**].

Note

At least one random variable must be defined and all three ways to define random variables may appear in the EMP annotations. See the scheduling model [**AIRLIFT**] for an example with both, a discrete random variable and a continuous random variable.

The keywords `setSeed` and `sample` are optional. The seed for the random number generator may be set with `setSeed`. The random number generator is used for the sampling routines that are called with the keyword `sample`. If `setSeed` is used, the seed will be set once before all samples will be generated. The keyword `sample` is followed by the name of the respective random variable, the sample size (a number) and - optionally - a variance reduction method. Note that the random variable must have been previously declared to follow a parametric distribution. Note further, that the sample size of more than one random variable may be customized simultaneously. Observe that the default sample size is 6. For details on available variance reduction methods, see section Customizing Sampling in the EMP Annotations.

Note

- A valid LINDO license is required to use the keywords `setSeed` and `sample`. The keyword `sample` may be used with a demo or community version of the LINDO license, but it is limited to the Normal and Binomial distributions with a maximum sample size of 10 and the variance reduction method cannot be changed.
- If a parametric distribution is used with any solver but LINDO, the keyword `sample` is mandatory, see above for details.

With the keyword `stage` random variables, variables and equations are assigned to their respective stages, where `StageNo` defines the stage number. Note that the default stage for all random variables, equations and variables is 1, except for the objective equation and variable. The default for those is the highest stage in the problem.

With the keyword `stageDefault` one can change the default stage to which symbols get assigned if they are not listed with a `stage` keyword explicitly.

4.51.11.2 Risk Measures with EMP

The literature on stochastic programming usually assumes that the expected value of the objective is optimized. The EMP tool follows this trend and implicitly optimizes the expected value. However, there are other risk measures that could be taken into consideration and that are frequently used in practice, particularly in finance. For example, a fund manager might be more interested in the expected value of the 10% worst cases of the projected wins than in the expected value of the overall distribution. The expected value of a partition of the distribution at one tail is called *Conditional Value at Risk*. The EMP framework provides special keywords to facilitate optimizing this risk measure. On a more abstract level, risk measures can be understood as mechanisms to evaluate the effects of uncertainty in the underlying system on the outcomes of interest. They can be used to modify the distribution of outcomes.

Using the example of an investor who wishes to balance expected rewards and the risk of loss when she decides how to allocate assets in a portfolio, we explore how stochastic optimization problems involving risk measures can be modeled with EMP. Note that this example is adapted from the stochastic portfolio model **[PORTFOLIO]**. For simplicity of exposition, we only describe two-stage models here. In the examples that follow, the period $(0, T)$ is the period between investing in a portfolio of assets and return from this portfolio.

We first present the example problem and introduce a new EMP keyword to model the expected value explicitly. Then we introduce and discuss Value at Risk (VaR) and Conditional Value at Risk (CVaR). At the end of this section, we give a summary of the EMP annotations that are specific to risk measures.

Note

The stochastic extension of EMP facilitates the optimization of a single risk measure or a combination of risk measures (for example, the weighted sum of Expected Value and CVaR). In addition, the modeler can choose to trade off risk measures.

Expected Value Revisited

Suppose an investor has the opportunity to invest a certain amount in three assets. She is given the probability distribution in Table 6 that links each asset with a possible return at time T. The question arises how she should allocate her funds between the three assets at time 0 in order to maximize her *expected* return at time T.

Scenario	Probability	ATT	GMC	USX
s1	1/12	1.300	1.225	1.149
s2	1/12	1.103	1.290	1.260
s3	1/12	1.216	1.216	1.419
s4	1/12	0.954	0.728	0.922
s5	1/12	0.929	1.144	1.169
s6	1/12	1.056	1.107	0.965
s7	1/12	1.038	1.321	1.133
s8	1/12	1.089	1.305	1.732
s9	1/12	1.090	1.195	1.021
s10	1/12	1.083	1.390	1.131
s11	1/12	1.035	0.928	1.006
s12	1/12	1.176	1.715	1.908

Table 6: Return by scenario

Mathematically, the problem can be expressed as follows:

$$\begin{array}{ll} \text{Max} & \mathbb{E}[R] \\ \text{s.t} & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{array} \tag{36}$$

where the variable R denotes the return and is a function of the random variable v, $\mathbb{E}[R]$ is the expected return, w_j is the weight associated with each asset j, and v_j is the return of each asset j. The weights can also be interpreted as proportions of the amount to be invested, their sum must equal 1. Note that w_j is the decision variable in this problem. Note further, that v_j is a random variable that depends on which scenario is realized.

We present two different ways to model this problem in GAMS EMP: in the first version the expected value of the return is modeled *implicitly* and in the second version it is modeled *explicitly*. Both models have two stages, in the first stage the weights are chosen without knowing which scenario will be realized, in the second stage the 12 scenarios are taken into account. We start with the part of the code where the data is given. It is named `data.gms` and it will be included in all models in this section.

```
Sets   j       "assets"     / att, gmc, usx /
       s       "scenarios"  / s1*s12 / ;

Table vs(s,j) "scenario returns from assets"
     att     gmc     usx
 s1  1.300   1.225   1.149
 s2  1.103   1.290   1.260
 s3  1.216   1.216   1.419
 s4  0.954   0.728   0.922
 s5  0.929   1.144   1.169
 s6  1.056   1.107   0.965
 s7  1.038   1.321   1.133
 s8  1.089   1.305   1.732
 s9  1.090   1.195   1.021
s10  1.083   1.390   1.131
s11  1.035   0.928   1.006
s12  1.176   1.715   1.908 ;

Parameters
      v(j)    "return from assets "
      p(s)    "probability" / #s [1/card(s)] / ;

v(j) =  sum(s, vs(s,j))/card(s);
```

The first model is similar to the two-stage example model above, the fact that the *expected* return is being maximized is not stated explicitly but is only implied:

```
* Core Model
$include data.gms
Variables
      r       "value of portfolio under each scenario"
      w(j)    "portfolio selection" ;
Positive Variables
      w ;

Equations
```

```
      defr    "return of portfolio"
      budget  "budget constraint" ;

defr..     r =e= sum(j, v(j)*w(j));
budget..   sum(j, w(j)) =e= 1;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /;
emp.nd=4;
put emp '* problem %gams.i%'
    / 'stage 2 v r defr'
    / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
      s_v(s,j) "return from assets by scenario"
      s_r(s)   "return from portfolio by scenario" ;

Set   dict / s  .scenario.''
             v  .randvar .s_v
             r  .level   .s_r /;

solve portfolio using emp max r scenario dict;
display s_v, s_r;
```

As usual, the core model is defined first and the specifications relating to the stochastic structure of the model are written to the file `emp.info`. Note that the statement `emp.nd=4` ensures that 4 decimal places are used for the values of the parameter `vs`. In the first line of the EMP annotations, the variables `v` and `r` and the equation `defr` are assigned to the second stage. Note that the other variables and equations (in this case, the variable `w` and the equation `budget`) are automatically assigned to the first stage. In the second line of the annotations, the EMP keyword `jrandvar` is used to declare that `v('att')`, `v('gmc')` and `v('usx')` are *joint* random variables, i.e. they are jointly distributed (thus we have 12 scenarios and not 12∗12∗12=1728 scenarios). The respective probabilities and values are specified with a loop statement. Observe that the syntax of the solve statement suggests that the return `r` is maximized. However, `r` is in fact a random variable, since it is a function of random variables, so actually the *expected* return is maximized.

In the second model we introduce a new variable for the expected return, `evr`. This new variable is linked to the EMP keyword `ExpectedValue` in the annotations, thus it is made explicit that the expected value of the return is optimized.

```
* Core Model
$include data.gms
Variables
      r       "value of portfolio under each scenario"
      w(j)    "portfolio selection"
      evr     "expected value of r"
      obj     "objective variable" ;
Positive Variables
      w ;

Equations
```

```
      defr    "return of portfolio"
      budget  "budget constraint"
      defobj  "objective equation" ;

defr..     r =e= sum(j, v(j)*w(j));
budget..   sum(j, w(j)) =e= 1;
defobj..   obj =e= evr;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /;
emp.nd=4;
put emp '* problem %gams.i%'
    / 'ExpectedValue r evr'
    / 'stage 1 obj defobj evr'
    / 'stage 2 v defr r'
    / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
      s_v(s,j) "return from assets by scenario"
      s_r(s)   "return from portfolio by scenario" ;

Set   dict / s  .scenario.''
             v  .randvar .s_v
             r  .level   .s_r /;

solve portfolio using emp max obj scenario dict;
```

In the EMP annotations, the EMP keyword `ExpectedValue` is followed by the variable `r` and the variable `evr`. In this way the variable `evr` is declared to be the expected value of the (random) variable `r`. Note that the new variables `evr` and `obj` belong to stage 1 and the variable `evr` is maximized (via `obj`). The remainder of the annotations and the code referring to the output-handling information in the dictionary are the same like in the first model. Both models have the same solution. We prefer the second model since the syntax is more explicit and clearer.

Observe that the keyword `ExpectedValue` is particularly useful if users wish to model an objective that is the weighted sum of several risk measures, thus trading-off different risk measures. An example with the weighted sum of the expected value and Value at Risk as objective and an example with the weighted sum of the expected value and Conditional Value at Risk as objective are given below.

Value at Risk (VaR)

The Value at Risk (VaR) is the value of the distribution at a given cut-off point; for example, the value of the standard Normal distribution at the point x such that 95% of the distribution is to the left. In this subsection we introduce the notion of VaR in more detail by developing a mathematical formulation, demonstrating how VaR can be modeled with GAMS EMP using a simple example and showing how to optimize the weighted sum of the expected value and VaR.

VaR: Mathematical Formulation

Suppose $G(x,\xi)$ is a real valued function of the decision vector x and a random data vector ξ and that it denotes the *loss* function of a portfolio of assets. We aim to restrict potential losses and so we choose a portfolio composition such that the loss only exceeds a certain threshold γ ($\gamma \in \mathbb{R}$) with a probability smaller or equal to α, $\alpha \in (0,1)$, where α is small. This condition can be modeled as a chance constraint and has the following form:

$$P(G(x,\xi) > \gamma) \leq \alpha \tag{37}$$

It is easy to see that this can be written in the following way:

$$P(G(x,\xi) - \gamma \leq 0) \geq 1 - \alpha. \tag{38}$$

Consider the random variable $Z_x := G(x,\xi) - \gamma$. For a given value of x, let $F_Z(z) := P(Z \leq z)$ be the cumulative distribution function of Z. Now, the point x satisfies the constraint (37) if and only if $F_Z(0) \geq 1-\alpha$. This is equivalent to saying that x satisfies the constraint (39) if and only if $F_Z^{-1}(1-\alpha) \leq 0$.

The (left-side) quantile $F_Z^{-1}(\theta)$ is called *Value at Risk*. It is denoted by $VaR_\theta(Z)$, i.e.

$$VaR_\theta(Z) = \inf\{t : F_Z(t) \geq \theta\}. \tag{39}$$

Hence constraint (38) can be written in the following equivalent form:

$$VaR_{1-\alpha}(G(x,\xi)) \leq \gamma. \tag{40}$$

Value at Risk as introduced in equation (39) refers to a percentile on the *left* tail of a distribution. From now on it will be denoted by $\underline{VaR}_\theta(Z)$. When considering the Value at Risk at the *right* tail of the distribution, θ typically equals 0.9 or 0.95, it is denoted by $\overline{VaR}_\theta(Z)$.

Figure 2 illustrates $\underline{VaR}_{0.05}(Z)$, where Z is normally distributed with mean $\mu = 0.645$ and standard deviation $\sigma = 1$.

The EMP framework provides the EMP keywords `varlo` and `varup` as a convenient alternative to chance constraints to model Value at Risk.

VaR with EMP: A Simple Example

We continue to illustrate with the stochastic portfolio model that we have already used above. Consider that the investor might be interested in a strategy that maximizes the threshold at a certain cutoff, say at 10% on the left tail of the return curve. Mathematically, the problem can be expressed as follows:

$$\begin{array}{ll} \text{Max} & \underline{VaR}_\theta[R] \\ \text{s.t} & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{array} \tag{41}$$

where $\underline{VaR}_\theta$ is the Value at Risk at the lower θth percentile.

In the EMP annotations of the corresponding GAMS model we introduce a new variable for VaR (`varr`), a scalar to specify the percentile we are interested in (`theta`) and the EMP keyword `varlo`:

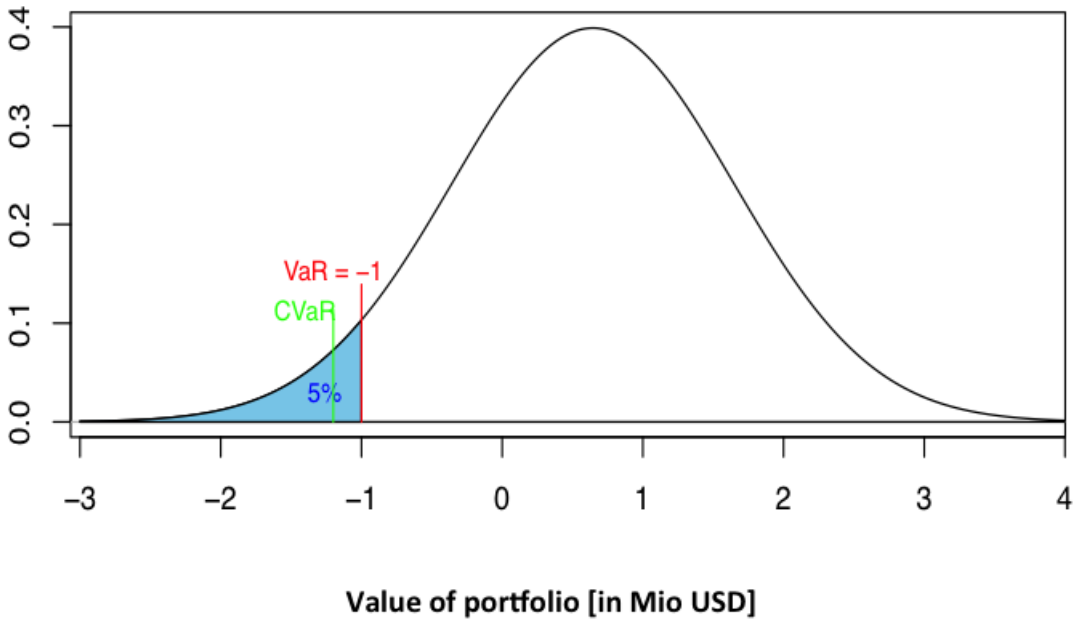


Figure 4.5 VaR and CVaR

```
* Core Model
$include data.gms
Variables
      r       "value of portfolio under each scenario"
      w(j)    "portfolio selection"
      varr    "value at risk of r"
      obj     "objective variable" ;
Positive Variables
      w ;

Scalar
      theta   "relative volume" / 0.1 /;

Equations
      defr    "return of portfolio"
      budget  "budget constraint"
      defobj  "objective equation" ;

defr..     r =e= sum(j, v(j)*w(j));
budget..   sum(j, w(j)) =e= 1;
defobj..   obj =e= varr;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
```

```
      / 'varlo r varr ' theta
      / 'stage 1 obj defobj varr'
      / 'stage 2 r defr v'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
      s_v(s,j) "return from assets by scenario"
      s_r(s)   "return from portfolio by scenario" ;

Set   dict / s  .scenario.''
             v  .randvar .s_v
             r  .level   .s_r /;

solve portfolio using emp max obj scenario dict;
```

The first line in the EMP annotations begins with the EMP keyword `varlo`. This line specifies that the variable `varr` is the Value at Risk relating to the random variable `r` and the scalar `theta` is the percentile (in range 0 to 1) we consider. The other lines in the annotations are similar to the second model above. Note that the objective equals the Value at Risk at the left tail, denoted by $\underline{VaR}$.

Observe that the EMP keyword `varlo` specifies VaR at the *left* tail of the probability distribution. For the `right` tail of the distribution, the EMP keyword to be used is `varup`.

Note

- The keyword `var` is identical to `varup` which refers to the right tail of the distribution.
- Currently only the solver DE supports the keywords for VaR.

Note further, that it is only appropriate to maximize $\underline{VaR}_\alpha$ and minimize $\overline{VaR_\alpha}$.

The implementation of the keywords `varlo` and `varup` is based on a mixed integer program similar to that described in section Example with a Single Chance Constraint. Observe that these implementations are likely to be hard and/or time consuming to solve. There is an option that allows users to customize the big M value (default is 1000) in the same manner that we outline below:

```
$onecho > de.opt
VaRBigM = 500
$offecho
portfolio.optfile=1;
```

Observe that the EMP framework offers an alternative, shorter way to model VaR. There is no additional variable for VaR, hence in the EMP annotations, the EMP keyword `varlo` is only followed by `theta` and the objective variable in the solve statement is `r`. As `varlo` is specified in the annotations, the risk measure is applied to the objective variable implicitly. The modified EMP annotations and the new solve statement follow.

```
File emp / '%emp.info%' /;
put emp '* problem %gams.i%'
      / 'varlo ' theta
      / 'stage 2 r defr v'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s,   put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;
...
solve portfolio using emp max r scenario dict;
```

Note that this alternative notation is shorter, but it is also opaque, therefore we recommend the first way.

Combining VaR and Expected Value

In a variation of the example above, consider an investor who aims to take into account both, the expected return and the Value at Risk of the return at a certain threshold θ. She combines the two risk measures and uses a scalar (λ) as a weight. A mathematical formulation of the problem reads as follows:

$$\begin{array}{ll} \text{Max} & \lambda\mathbb{E}[R] + (1-\lambda)\underline{VaR}_\theta[R] \\ \text{s.t} & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{array} \tag{42}$$

where $\mathbb{E}[R]$ is the expected value of the return and $\underline{VaR}_\theta$ is the Value at Risk at the θth percentile.

This problem can be modeled in GAMS EMP as follows:

```
* Core Model
$include data.gms
Variables
      r       "value of portfolio under each scenario"
      w(j)    "portfolio selection"
      varr    "value at risk of r"
      evr     "expected value of r"
      obj     "objective variable" ;
Positive Variables
      w ;

Scalars
      theta   "relative volume" / 0.1 /
      lambda  "weight EV versus VaR"  / 0.2 /;

Equations
      defr    "return of portfolio"
      budget  "budget constraint"
      defobj  "convex combination of both risk measures" ;

defr..      r =e= sum(j, v(j)*w(j));
budget..    sum(j, w(j)) =e= 1;
defobj..    obj =e= lambda*evr + (1-lambda)*varr;

Model portfolio_ext / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
      / 'ExpectedValue r evr'
      / 'varlo r varr ' theta
      / 'stage 2 r defr v'
      / 'stage 1 defobj obj'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
      s_v(s,j) "return from assets by scenario /s1.att 1/"
```

```
      s_r(s)   "return from portfolio by scenario" ;

Set   dict / s  .scenario.''
             v  .randvar .s_v
             r  .level   .s_r /;

solve portfolio_ext using EMP max obj scenario dict;
```

Note that we introduced a variable for the expected value of the return like in the example above where we modeled the expected return explicitly. Note further, that we added the parameter `lambda` and we defined the variable `obj` to be the weighted sum of the expected value of the return `evr` and the lower VaR `varr`. Observe that in the EMP annotations both keywords `ExpectedValue` and `varlo` are used. The other parts of the model remain unchanged.

Conditional Value at Risk (CVaR)

The Conditional Value at Risk (CVaR) is the expected value of the left or right tail of the distribution; for example, the expected value of the left 5% of the standard Normal distribution. Note that CVaR is also called *expected shortfall*. In this subsection we introduce and discuss CVaR in more detail by presenting a mathematical formulation, demonstrating how CVaR can be modeled with GAMS EMP using a simple example and showing how to optimize the weighted sum of the expected value and CVaR.

CVaR: Mathematical Formulation

$\underline{CVaR}_\alpha$ is the expected average return (in a given time period) given that we are in the $(\alpha \times 100)\%$ *left tail* of the return distribution, where $\alpha \in (0,1)$. In other words, $\underline{CVaR}_\alpha$ is a mean of the left tail. For example, if we are interested in the 5% worst cases, i.e. $\alpha = 0.05$, $\underline{CVaR}_\alpha$ is the conditional expectation of the return, given the return is no greater than VaR.

Let ξ be a random variable with probability density function $p(\xi)$, let $G(\xi)$ be a function of the random variable ξ denoting the *return* of a portfolio of assets and let $\alpha \in (0,1)$ be a probability. Then the conditional value at risk of $G(\xi)$ is defined as

$$\underline{CVaR}_\alpha(G(\xi)) = \frac{1}{\alpha}\int_{-\infty}^{VaR_\alpha} G(\xi)\cdot p(\xi)d\xi, \tag{43}$$

where VaR_α is the value at risk.

CVaR with EMP: A Simple Example

In the stochastic portfolio example that we have used throughout this section, the investor might be interested to make sure that in the worst cases she loses as little as possible. Thus she could consider only the worst 10% of possible cases and allocate her funds such that the expected mean return in these cases is maximized. Mathematically, the problem can be expressed as follows:

$$\begin{array}{ll} \text{Max} & \underline{CVaR}_\theta[R] \\ \text{s.t} & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{array} \tag{44}$$

where $\underline{CVaR}_\theta$ is the CVaR at the confidence level θ at the *left* tail of the distribution.

Like in the case of Value at Risk above, we introduce a new variable for CVaR (`cvarr`), a scalar to specify the percentile we are interested in (`theta`) and the EMP keyword `cvarlo` in the EMP annotations:

```
* Core Model
$include data.gms
Variables
      r       "value of portfolio under each scenario"
      w(j)    "portfolio selection"
      cvarr   "conditional value at risk of r"
      obj     "objective variable" ;
Positive Variables
      w ;

Scalar
      theta   "relative volume" / 0.1 /;

Equations
      defr    "return of portfolio"
      budget  "budget constraint"
      defobj  "objective equation" ;

defr..     r =e= sum(j, v(j)*w(j));
budget..   sum(j, w(j)) =e= 1;
defobj..   obj =e= cvarr;

Model portfolio / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
      / 'cvarlo r cvarr' theta
      / 'stage 1 obj defobj cvarr'
      / 'stage 2 r defr v'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
      s_v(s,j) "return from assets by scenario"
      s_r(s)   "return from portfolio by scenario" ;

Set   dict / s  .scenario.''
             v  .randvar .s_v
             r  .level   .s_r /;

solve portfolio using emp max obj scenario dict;
```

In the EMP annotations, the first line starts with the EMP keyword `cvarlo`. This line specifies that the variable cvarr is the left-tail CVaR relating to the random variable `r` and the scalar `theta` is the fraction (in range 0 to 1) we consider. Like in the previous section, the objective variable, the objective equation and the variable in the objective equation belong to the first stage while the equation that handles the random data and all its variables belong to the second stage. Observe that the objective equals the Conditional Value at Risk.

Note that the keyword `cvarlo` specifies CVaR at the *left* tail of the probability distribution. For the *right* tail of the distribution the keyword to be used is `cvarup`.

Note

- The EMP keyword `cvar` is identical to the keyword `cvarup`.
- Currently only the solver DE supports the keywords for CVaR.

The conditional value at risk denoting the mean of the *right* tail of the distribution can be denoted by $\overline{CVaR}$ and is defined as:

$$\overline{CVaR}_\alpha(G(\xi)) = \frac{1}{1-\alpha}\int_{VaR_\alpha}^{\infty} G(\xi)\cdot p(\xi)d\xi. \tag{45}$$

Observe that it is only appropriate to maximize $\underline{CVaR}_\alpha$ and minimize $\overline{CVaR_\alpha}$. Furthermore, $\underline{CVaR}_\alpha$ is a concave function and so should only be constrained using e.g.

$$\underline{CVaR}_\alpha \geq \gamma$$

and $\overline{CVaR_\alpha}$ is convex, so should only appear in constraints like

$$\overline{CVaR_\alpha} \leq \gamma.$$

The EMP framework offers an alternative way to model CVaR. In this alternative formulation, there is no additional variable for CVaR, hence in the EMP annotations, the EMP keyword `cvarlo` is only followed by `theta` and the objective variable in the solve statement is `r`. As `cvarlo` is specified in the annotations, the risk measure is applied to the objective variable implicitly. The modified EMP annotations and the new solve statement follow.

```
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
      / 'cvarlo ' theta
      / 'stage 2 r defr v'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s,   put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;
...
solve portfolio using emp max r scenario dict;
```

Note that this alternative notation is shorter, but it is considerably less clear, therefore we recommend the explicit notation of the first model.

Combining CVaR and Expected Value

In a final variation on the portfolio example, we consider an investor who aims to take into account both, the expected return and the Conditional Value at Risk of the return at a certain threshold θ. She combines the two risk measures and uses a scalar (λ) to weigh the two summands. A mathematical formulation of the problem reads as follows:

$$\begin{array}{ll} \text{Max} & \lambda\mathbb{E}[R] + (1-\lambda)\underline{CVaR}_\theta[R] \\ \text{s.t} & R = \sum_j w_j v_j \\ & \sum_j w_j = 1 \\ & w_j \geq 0, \end{array} \tag{46}$$

where $\mathbb{E}[R]$ is the expected value of the return and $\underline{CVaR}_\theta$ is the CVaR at the confidence level θ.

This problem can be modeled with GAMS EMP as follows:

```
* Core Model
$include data.gms
Variables
      r       "value of portfolio under each scenario"
      w(j)    "portfolio selection"
      cvarr   "conditional value at risk of r"
      evr     "expected value of r"
      obj     "objective variable" ;
Positive Variables
      w ;

Scalars
      theta   "relative volume" / 0.1 /
      lambda  "weight EV versus CVaR"  / 0.2 /;

Equations
      defr    "return of portfolio"
      budget  "budget constraint"
      defobj  "convex combination of both risk measures" ;

defr..     r =e= sum(j, v(j)*w(j));
budget..   sum(j, w(j)) =e= 1;
defobj..   obj =e= lambda*evr + (1-lambda)*cvarr;

Model portfolio_ext / all /;

* EMP Annotations
File emp / '%emp.info%' /
put emp '* problem %gams.i%'
      / 'ExpectedValue r evr'
      / 'cvarlo r cvarr ' theta
      / 'stage 2 r defr v'
      / 'stage 1 defobj obj'
      / "jrandvar v('att') v('gmc') v('usx')"
loop(s,
  put / p(s) vs(s,"att") vs(s,"gmc") vs(s,"usx"));
putclose emp;

* Dictionary
Parameters
      s_v(s,j) "return from assets by scenario"
      s_r(s)   "return from portfolio by scenario" ;

Set   dict / s  .scenario.''
             v  .randvar .s_v
             r  .level   .s_r /;

solve portfolio_ext using emp max obj scenario dict;
```

Note that apart from the EMP keyword `varlo` being replaced by `cvarlo`, the code is identical to the code above, where we maximized the weigthed sum of the expected value and VaR. Observe that by decreasing the value of λ from 1 to 0 we can explore the results of an increasingly risk averse behavior.

EMP Syntax for Risk Measures

The EMP annotations where risk measures are used have the same general syntax like stochastic models with recourse. In addition, the following keywords may be used:

```
ExpectedValue rv var
varlo [rv var] scalar
varup [rv var] scalar
cvarlo [rv var] scalar
cvarup [rv var] scalar
```

The EMP keyword `Expected Value` is used to state that the variable `var` is the expected value of the random variable `rv`. Note that the random variable `rv` must be defined to be a random variable in the EMP annotations.

The EMP keywords `varlo`, `varup`, `cvarlo` and `cvarup` refer to VaR at the left tail of the distribution ($\underline{VaR}_\alpha$), VaR at the right tail of the distribution ($\overline{VaR}_\alpha$), CVaR at the left tail of the distribution ($\underline{CVaR}_\alpha$) and CVaR at the right tail of the distribution ($\overline{CVaR}_\alpha$) respectively.

Note

The EMP keyword `var` is a synonym to `varup` and `cvar` is a synonym to `cvarup`.

The specifications that follow all four keywords have a long and a short version. In the long version, the random variable `rv` and the variable that is assigned to be the respective risk measure (VaR or CVaR) are named explicitly. The scalar is a number in the interval $(0, 1)$, it defines the percentile in VaR and confidence level in CVaR. In the short version, only the scalar is specified. In this case the risk measure will applied to the objective variable implicitly.

Note

These five keywords are only supported by the solver DE.

4.51.11.3 Chance Constraints with EMP

In stochastic programs with chance constraints the goal is to make an optimal decision prior to the realization of random data while allowing the constraints (or some of them) to be violated with a certain probability. Note that chance constraints are also called *probabilistic constraints*.

This section is organized as follows. After introducing a mathematical formulation of chance constraints, we show how such problems can be modeled with GAMS EMP using a simple example with only one chance constraint. Then we extend the model to include multiple chance constraints. We discuss the two ways to model problems with multiple chance constraints: using *joint chance constraints* and using *individual chance constraints*. Joint chance constraints require that all constraints are satisfied simultaneously with a given probability. Individual chance constraints require each constraint to be satisfied with a given probability independently of other constraints. Further, the EMP framework offers the option to penalize the violation of chance constraints. This is the topic of subsection Penalizing Violations of Chance Constraints. We conclude this section with an overview of the EMP annotations that are specific to chance constraints.

Chance Constraints: Mathematical Formulation

Mathematically, a stochastic linear program with chance constraints can be expressed as follows:

$$\begin{array}{ll} \text{Min}_x & c^T x \\ \text{s.t.} & P(Ax \leq b) \geq p \\ & x \geq 0, \end{array} \tag{47}$$

where $x \in \mathbb{R}^n$ is the decision variable and c^T denotes the coefficients of the objective function, $A \in \mathbb{R}^{m \times n}$ is a random matrix and represents the coefficients and $b \in \mathbb{R}^m$ is a random vector and denotes the right-hand side of the constraints. The distinctive feature of stochastic programs with chance constraints is that the constraints (or some of them) may be violated with probability $\epsilon = 1 - p$, where $0 < p \leq 1$. Note that ϵ is sometimes called the *risk tolerance*.

One way to solve a stochastic problem with chance constraints is to convert it to a mixed-integer problem (MIP) first and then solve the MIP equivalent. The idea is to introduce a set of scenarios S and a vector with binary variables, say $y_k \in \mathbb{R}^m$, for each scenario $k \in S$. The binary variables take value 1 if the corresponding constraint is satisfied in a scenario and 0 otherwise. A scenario-based formulation of the chance-constrained stochastic linear program above can be written as:

$$\begin{array}{ll} \text{Min}_x & c^T x \\ \text{s.t.} & A^k x \leq b^k + M^k(1 - y_k) \\ & \sum_{k \in S} y_k \geq p \times |S| \\ & x \geq 0, \quad y \in (0,1)^{|S|}, \end{array} \tag{48}$$

where $M^k \in \mathbb{R}^m$ is a chosen big-M vector. The entries of the vector M^k should be chosen such that it does not cut off any feasible solution if $y_k = 0$.

Example with a Single Chance Constraint

We start with the simplest case, where the random matrix A consists of just one random vector a, resulting in a problem with a single chance constraint. The following example is adapted from model [**SIMPLECHANCE**].

$$\begin{array}{lll} \text{Min} & x_1 + x_2 & \\ \text{s.t.} & P(\omega x_1 + x_2 \geq 7) \geq 0.75, & \omega \in \Omega = \{1,2,3,4\} \\ & x_1, x_2 \geq 0. & \end{array} \tag{49}$$

Here the random parameter ω has four possible realizations. Thus we have four scenarios and we assume that each scenario is equally likely to be realized, i.e. $\pi^k = \frac{1}{4}$, where π^k denotes the probability that scenario k is realized. Note that in this example b is fixed at 7. As $p = 0.75$ and each scenario is equally likely to be realized, we need to choose x_1 and x_2 such that the inequality is satisfied in at least 3 scenarios. For clarity, we spell out the inequalities for the scenarios:

$$\begin{array}{llll} k = 1: & \omega_1 = 1 & \omega_1 x_1 + x_2 \geq 7 \\ k = 2: & \omega_2 = 2 & \omega_2 x_1 + x_2 \geq 7 \\ k = 3: & \omega_3 = 3 & \omega_3 x_1 + x_2 \geq 7 \\ k = 4: & \omega_4 = 4 & \omega_4 x_1 + x_2 \geq 7. \end{array} \tag{50}$$

The MIP equivalent of problem (49) is given below:

$$
\begin{array}{ll}
\text{Min} & x_1 + x_2 \\
\text{s.t.} & 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
& 2x_1 + x_2 \geq 7 - M(1 - y_2) \\
& 3x_1 + x_2 \geq 7 - M(1 - y_3) \\
& 4x_1 + x_2 \geq 7 - M(1 - y_4) \\
& \\
& cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 4,\ \pi^k = \frac{1}{4} \\
& \\
& x_1, x_2 \geq 0 \\
& 0 \leq cc_1 \leq (1 - 0.75) \\
& y_k \in (0, 1).
\end{array}
\tag{51}
$$

Observe that the first four constraints cover the four possible scenarios with ω taking the values 1, 2, 3 and 4 respectively. On the right-hand side we introduce a big-M factor and y_k, a binary indicator variable. y_k takes the value 1 if the constraint is satisfied and zero otherwise. The new variable cc_1 represents the probability that the constraint is violated. If cc_1 equals zero, the sum will equal 1, indicating that the constraint is satisfied in all four scenarios. If $cc_1 = 0.25$, the constraint will be unsatisfied in one scenario out of four (for this scenario $y_k = 0$).

The problem can be modeled in GAMS EMP as follows:

```
* Core Model
Positive Variables x1, x2;
Variables          z;

Scalar om  / 1 /;

Equations obj, e1;

obj..   z =e= x1 + x2;
e1..    om*x1 + x2 =g=  7;

Model sc / all /;

* EMP Annotations
File emp / '%emp.info%' /;
put emp;
$onput
randvar om discrete 0.25 1  0.25 2   0.25 3   0.25 4
chance e1 0.75
$offput
putclose emp;

* Dictionary
Set scen "scenarios" / s1*s4 /;
Parameter  s_om(scen)
       x1_l (scen)
       x2_l (scen)
       e1_l (scen);

Set dict / scen .scenario.''
            om  .randvar .s_om
            x1  .level   .x1_l
            x2  .level   .x2_l
```

```
            e1  .level   .e1_l/;

solve sc min z use EMP scenario dict;
display s_om, x1_l, x2_l, e1_l;
```

Like other EMP stochastic programming models, the model consists of three parts: the core model, the EMP annotations and the dictionary with output-handling information. As usual, the core model is defined as a deterministic model and the specifications relating to the stochastic structure of the problem are written to the file `emp.info`. In the EMP annotations, the EMP keyword `randvar` is used to declare a parameter from the core model as a random variable. The keyword is followed by `om`, the random variable, and the distribution. The EMP keyword `discrete` indicates that we have a discrete distribution. The discrete distribution is specified via probability-numerical value pairs. Note that continuous distributions are also possible. See section Random Variables with Continuous Distributions for more information. The second line in the EMP annotations starts with the EMP keyword `chance` followed by the equation `e1` and a probability. The specification means that the constraints `e1` must hold for at least 75% of all scenarios. We can verify that this requirement has been enforced by checking in the listing file the level value of the constraint `e1_l`. We will see that in the first scenario the constraint was violated, but is was satisfied in all other scenarios.

Observe that the EMP keyword `chance` allows for an optional specification that is related to a penalization factor for constraints that are violated. This topic is discussed in section Penalizing Violations of Chance Constraints below.

Note

> Currently, there are no stages in stochastic problems with chance constraints, unlike in stochastic problems with recourse.

As the problem is solved as a MIP, it is important that the options OptCA and OptCR are assigned appropriate values.

Note that the default value of M is 1e7 for the solver Lindo. To customize this and set it to, e.g., 1000 instead, one could insert the following five lines before the solve statement:

```
option emp = lindo;
$onecho > lindo.opt
STOC_BIGM 1e3
$offecho
sc.optfile = 1;
```

For DE the default value of M is 10000 and it could be changed to 1000 in a similar way as seen above for Lindo:

```
option emp = de;
$onecho > de.opt
ccreform bigM 1e3
$offecho
sc.optfile = 1;
```

Note that `ccreform` is a DE option. This option allows to specify two alternative solution strategies: a reformulation using a convex hull and a reformulation using indicator variables and indicator constraints. The following line indicates that a convex hull with $M = 1000$ and $\epsilon = 0.00001$ is to be used:

```
ccreform cHull 1e3 1e-6
```

For indicator variables and constraints, the following line is used:

```
ccreform indic
```

Note that currently only the solver CPLEX supports indicator variables, so the resulting reformulated problem has to be solved with CPLEX.

Examples with Multiple Chance Constraints

Stochastic problems with multiple chance constraints have two different forms: problems with *joint* chance constraints and problems with *multiple individual* chance constraints. In problems with joint chance constraints the constraints all have to be satisfied simultaneously with probability p. In problems with multiple individual chance constraints each constraint carries its own risk tolerance. We illustrate with two examples.

Joint Chance Constraints

We illustrate joint chance constraints by extending the example above by one constraint resulting in a problem with two chance constraints:

$$\begin{array}{ll} \text{Min} & x_1 + x_2 \\ \text{s.t.} & P(\omega_1 x_1 + x_2 \geq 7;\ \omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, \qquad (\omega_1, \omega_2) \in \Omega \\ & x_1, x_2 \geq 0, \end{array} \tag{52}$$

where

$$\Omega = \{(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3),(4,1),(4,2),(4,3)\} \tag{4.27}$$

and

$$\pi^k = \pi(\omega_1, \omega_2) = \frac{1}{12} \quad \text{for all} \quad (\omega_1, \omega_2) \in \Omega. \tag{4.28}$$

Note that both random variables follow discrete uniform distributions, ω_1 has four realizations and ω_2 has three realizations. Thus we have 12 scenarios that are all equally likely. The MIP equivalent takes the following form:

$$\begin{array}{ll} \text{Min} & x_1 + x_2 \\ \\ \text{s.t.} & 1x_1 + x_2 \geq 7 - M(1 - y_1) \\ & 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\ & 2x_1 + x_2 \geq 7 - M(1 - y_2) \\ & 1x_1 + 3x_2 \geq 12 - M(1 - y_2) \\ & \vdots \\ & 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\ & 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\ \\ & cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 12,\ \pi^k = \frac{1}{12} \\ & x_1, x_2 \geq 0 \\ & 0 \leq cc_1 \leq (1 - 0.6) \\ & y_k \in (0,1). \end{array} \tag{53}$$

Note that the first set of constraints covers the 12 scenarios, where each scenario has two constraints. The other constraints are similar to those introduced in the previous model.

The problem can be modeled with GAMS EMP as follows:

```
* Core Model
Positive Variables x1, x2;
Variables          z;

Scalars om1, om2;
om1 = 1;
om2 = 1;

Equations obj, e1, e2;

obj..   z =e= x1 + x2;
e1..    om1*x1 +   x2 =g=  7;
e2..    om2*x1 + 3*x2 =g= 12;

Model sc / all /;

* EMP Annotations
File emp / '%emp.info%' /;
put emp '* problem %gams.i%'/;
$onput
randvar om1 discrete 0.25 1  0.25 2  0.25 3  0.25 4
randvar om2 discrete 0.3333 1  0.3334 2  0.3333 3
chance e1 e2 0.6
$offput
putclose emp;

* Dictionary
Set scen "scenarios" / s1*s12 /;
Parameters  s_om1(scen)
            s_om2(scen)
            x1_l (scen)
            x2_l (scen)
            e1_l (scen)
            e2_l (scen);

Set dict / scen .scenario.''
           om1  .randvar .s_om1
           om2  .randvar .s_om2
           x1   .level   .x1_l
           x2   .level   .x2_l
           e1   .level   .e1_l
           e2   .level   .e2_l/;

solve sc min z use emp scenario dict;
display s_om1, s_om2, x1_l, x2_l, e1_l, e2_l;
```

In the EMP annotations, the line `chance e1 e2 0.6` specifies that in at least 60% of all scenarios both constraints `e1` and `e2` must be satisfied at the same time. There are a total of 12 scenarios, hence both constraints must be satisfied in at least 8 scenarios (i.e. $12 \times 0.6 = 7.2 \leq 8$). We can verify that this requirement has been enforced by checking in the listing file the level values of the constraints, `e1_l` and `e2_l`. Indeed, in the optimal solution both constraints hold in scenarios 4 to 12, meaning that there are 9 scenarios that satisfy both inequalities.

Individual Chance Constraints

In stochastic problems with individual chance constraints there is no correlation between the probabilities of the rows of the matrix A. Hence problem (47) takes the following form:

$$\begin{array}{ll} \text{Min}_x & c^T x \\ \text{s.t.} & P(A_i x \leq b_i) \geq p_i, \quad i = 1, \ldots, m \\ & x \geq 0. \end{array} \tag{54}$$

To illustrate, we use an extended version of the the example from the previous section and replace the joint chance constraints by individual chance constraints:

$$\begin{array}{lll} \text{Min} & x_1 + x_2 & \\ \text{s.t.} & P(\omega_1 x_1 + x_2 \geq 7) \geq 0.75, & \omega_1 \in \Omega_1 = \{1, 2, 3, 4\} \\ & P(\omega_2 x_1 + 3x_2 \geq 12) \geq 0.6, & \omega_2 \in \Omega_2 = \{1, 2, 3\} \\ & P(\omega_1 x_1 + \omega_2 x_2 \geq 10) \geq 0.5, & (\omega_1, \omega_2) \in \Omega_1 \times \Omega_2 = \Omega \\ & x_1, x_2 \geq 0. & \end{array} \tag{55}$$

Note that Ω is defined as in (52) above, we have again 12 scenarios, each with probability $\pi^k = \frac{1}{12}$. However, in this example, the first inequality must hold in 9 out of 12 scenarios ($0.75 \times 12 = 9$), the second inequality must hold in 8 out of 12 inequalities ($0.6 \times 12 = 7.2 \leq 8$) and the third inequality must hold in 6 out of 12 scenarios. Note further, that we may have four types of scenarios: scenarios where all constraints are violated, scenarios where two constraints are violated, scenarios where one constraint is violated and scenarios where all three constraints are satisfied. The only condition is that for each constraint there is the respective number of scenarios where the constraint is satisfied. Observe that the random data in the third inequality is a combination of the random data of the first two inequalities. The inequalities for the scenarios are given below:

$$\begin{array}{lll} k = 1: & \omega_1^1 = 1,\ \omega_2^1 = 1 & \omega_1^1 x_1 + x_2 \geq 7 \\ & & \omega_2^1 x_1 + 3x_2 \geq 12 \\ & & \omega_1^1 x_1 + \omega_2^1 x_2 \geq 10 \\ k = 2: & \omega_1^2 = 1,\ \omega_2^2 = 2 & \omega_1^2 x_1 + x_2 \geq 7 \\ & & \omega_2^2 x_1 + 3x_2 \geq 12 \\ & & \omega_1^2 x_1 + \omega_2^2 x_2 \geq 10 \\ \vdots & & \\ k = 12: & \omega_1^{12} = 4,\ \omega_2^{12} = 3 & \omega_1^{12} x_1 + x_2 \geq 7 \\ & & \omega_2^{12} x_1 + 3x_2 \geq 12 \\ & & \omega_1^{12} x_1 + \omega_2^{12} x_2 \geq 10 \end{array} \tag{56}$$

The MIP equivalent follows:

$$
\begin{array}{ll}
\text{Min} & x_1 + x_2 \\
\\
\text{s.t.} & 1x_1 + x_2 \geq 7 - M(1 - y_1^1) \\
& 1x_1 + 3x_2 \geq 12 - M(1 - y_1^2) \\
& 1x_1 + 1x_2 \geq 10 - M(1 - y_1^3) \\
& 2x_1 + x_2 \geq 7 - M(1 - y_2^1) \\
& 1x_1 + 3x_2 \geq 12 - M(1 - y_2^2) \\
& 2x_1 + 1x_2 \geq 10 - M(1 - y_2^3) \\
& \vdots \\
& 4x_1 + x_2 \geq 7 - M(1 - y_{12}^1) \\
& 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}^2) \\
& 4x_1 + 3x_2 \geq 10 - M(1 - y_{12}^3) \\
\\
& cc_1 = 1 - \sum_k \pi^k y_k^1, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& cc_2 = 1 - \sum_k \pi^k y_k^2, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& cc_3 = 1 - \sum_k \pi^k y_k^3, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
\\
& x_1, x_2 \geq 0 \\
& 0 \leq cc_1 \leq (1 - 0.75) \\
& 0 \leq cc_2 \leq (1 - 0.6) \\
& 0 \leq cc_3 \leq (1 - 0.5) \\
& y_k^j \in (0, 1).
\end{array}
\tag{57}
$$

As expected, there are three constraints for every scenario. Note that we introduced three new variables, cc_1, cc_2 and cc_3 and three corresponding constraints. Each of the variables has a different range reflecting the different probabilities with which a constraint may be violated.

The core model of the GAMS code is very similar to the core model with joint chance constraints. We just add the third inequality:

```
Equations obj, e1, e2, e3;

obj..   z =e= x1 + x2;
e1..    om1*x1 +   x2 =g=  7;
e2..    om2*x1 + 3*x2 =g= 12;
e3..    om1*x1 + om2*x2 =g= 10;
```

There is a slight modification in the annotations:

```
File emp / '%emp.info%' /;
put emp '* problem %gams.i%'/;
$onput
randvar om1 discrete 0.25 1   0.25 2  0.25 3  0.25 4
randvar om2 discrete 0.3333 1  0.3334 2  0.3333 3
chance e1 0.75
chance e2 0.6
chance e3 0.5
$offput
putclose emp;
```

Observe that we have three lines that start with the EMP keyword `chance:` every constraint is listed separately with its respective probability.

Note

In case one constraint has to be satisfied in all scenarios (so it is strictly speaking not a chance constraint), then it has to be listed with probability 1.0.

An overview of which constraints are satisfied in which scenarios in the optimal solution is given in the following table:

Scenarios	1	2	3	4	5	6	7	8	9	10	11	12
Scenarios where `e1` is satisfied				x	x	x	x	x	x	x	x	x
Scenarios where `e1` is satisfied		x	x		x	x		x	x		x	x
Scenarios where `e1` is satisfied						x	x	x	x	x	x	x

Table 7: Overview of constraint satisfaction per scenario

Observe that all constraints are satisfied in as many scenarios as required and there are scenarios where all three constraints are satisfied ($k = 6, 8, 9, 11, 12$), scenarios where only two constraints are satisfied ($k = 5, 7, 10$), scenarios where only one constraint is satisfied ($k = 2, 3, 4$) and one scenario where all constraints are violated ($k = 1$).

The choice whether joint or individual chance constraints should be used depends on the system being modeled. Both approaches have their own advantages. Individual chance constraints are weaker since not all constraints have to be satisfied at the same time. This can be clearly observed in the optimal solution for example (55). The objective value is 5.20 in the model with joint chance constraints and 4.75 in the model with individual chance constraints (assuming that each constraint is satisfied in 60% of all scenarios). As it is a minimizing problem, the model with individual chance constraints yields the better result. However, in this solution we have only 6 scenarios where both constraints are simultaneously satisfied while each constraint is satisfied in eight scenarios in total for the joint chance constraint formulation, as required.

Penalizing Violations of Chance Constraints

The EMP framework provides the syntax for a penalty factor for each scenario that violates one or more constraints. In the example with joint constraints above, the EMP annotations could be modified in the following way:

```
$onput
randvar om1 discrete 0.25 1  0.25 2  0.25 3  0.25 4
randvar om2 discrete 0.3333 1  0.3334 2  0.3333 3
chance e1 e2 0.6 3
$offput
```

Note that we added the number 3 at the end of the last line. This new entry represents a penalty factor. Recall that the MIP equivalent (53) takes the following form:

$$
\begin{array}{ll}
\text{Min} & z = x_1 + x_2 \\
\text{s.t.} & 1x_1 + x_2 \geq 7 - M(1 - y_1) \\
& 1x_1 + 3x_2 \geq 12 - M(1 - y_1) \\
& \vdots \\
& 4x_1 + x_2 \geq 7 - M(1 - y_{12}) \\
& 3x_1 + 3x_2 \geq 12 - M(1 - y_{12}) \\
& cc_1 = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12} \\
& x_1, x_2 \geq 0 \\
& 0 \leq cc_1 \leq (1 - 0.6) \\
& y_k \in (0, 1).
\end{array}
\tag{58}
$$

Observe that the probability with which the constraints are allowed to be violated is stored in the variable cc_1. Introducing a penalty factor or weight w has the effect that the term $(w \times cc_1)$ is added to the objective function:

$$z = x_1 + x_2 + w \times cc_1 = x_1 + x_2 + 3\,cc_1 \tag{4.29}$$

Similarly, we could add weights in the example with individual chance constraints:

```
chance E1 0.75 5
chance E2 0.6 6
chance E3 0.5 7
```

Given these annotations, we obtain the following objective function in the MIP equivalent:

$$z = x_1 + x_2 + 5\,cc_1 + 6\,cc_2 + 7\,cc_3. \tag{4.30}$$

Penalty terms can be useful to explore sensitivities to slight changes.

Note

> Like in the context of soft constraints, the penalty term is *added* in *minimization* problems and *subtracted* in *maximization* problems.

Moreover, the probability expression cc_1 in the MIP equivalent may be used as a variable in the original model. For example, in the joint chance constraints problem above, we could introduce a new variable, *viol*, in the objective function:

$$\text{Min} \quad z = x_1 + x_2 + 3 * viol \tag{4.31}$$

Then we write the chance constraint specification in the EMP annotations as follows:

```
chance E1 E2 0.6 viol
```

The addition of the variable `viol` as penalty factor causes cc_1 to be replaced by *viol* in the MIP equivalent. Thus we have:

$$viol = 1 - \sum_k \pi^k y_k, \quad k = 1, \ldots, 12, \quad \pi^k = \frac{1}{12}, \quad viol \in [0, 0.4]. \tag{4.32}$$

Note that the corresponding model is equivalent to the joint chance constraints model with penalty factor 3 with which we started this subsection.

EMP Syntax for Chance Constraints

The general syntax of the EMP annotations used to specify stochastic problems with chance constraints is as follows:

```
[randvar rv discrete prob val {prob val}]
[randvar rv distr par {par}]
[jrandvar rv rv {rv} prob val val {val} {prob val val {val}}]
chance equ {equ} [holds] minRatio [weight|var]
{chance equ {equ} [holds] minRatio [weight|var]}
```

The first three lines present three ways to specify random variables. The syntax and logic is the same as in stochastic problems with recourse, see above for details. Note that at least one random variable has to be specified. The EMP keyword `chance` is used to define the constraint(s) that only have to hold for a certain ratio of the possible outcomes. This ratio is given with the number `minRatio` ($0 \leq$ minRatio ≤ 1). The keyword `holds` is optional and does not affect the solver. The remaining specifications are optional and relate to the penalization of chance constraints that are violated. If `weight` is defined, the violation of a chance constraint will be penalized in the objective function (weight $\times$ violationRatio). Alternatively, if the name of a variable of the model (`var`) is specified, the violation will be multiplied by the value of this variable instead of the fixed multiplier `weight`.

In this section we have introduced various versions of problems with chance constraints with simple examples. For a more complex example, see the farming model [**KILOSAFARM**] in the GAMS EMP Model Library.

4.51.12 EMP Keywords

Some words act as keywords in the context of the EMP annotations in the file `emp.info`, but they are *not* GAMS reserved words, i.e. they are not keywords in the GAMS code apart from the EMP annotations. In this section we present an overview of all GAMS EMP keywords, ordered according to the type of programming where they are used.

4.51.12.1 GAMS EMP Keywords for Soft Constraints

EMP Keyword	Description
`abs`	Penalty function: absolute value of the equation.
`adjustequ`	Indicates that the specification(s) that follow relate to equations that are converted from constraints to penalty terms in the objective function.
`maxz`	Penalty function: the maximum of the equation and zero. An example is given above.
`sqr`	Penalty function: least squares applied to the equation. An example is given above.

4.51.12.2 GAMS EMP Keywords for Variational Inequalities

EMP Keyword	Description
`VI`	Indicates that the specifications that follow are a variational inequality. See the discussion of EMP specifications for VI and the example above.
`QVI`	Indicates that the specifications that follow are a quasi-variational inequality. See the discussion of EMP specifications for QVI and the example above.

4.51.12.3 GAMS EMP Keywords for Equilibrium Problems

EMP Keyword	Description
`dualvar`	See below.
`Equilibrium`	Indicates that the specifications that follow define the structure of an equilibrium problem. For examples, see sections Equilibrium Problems with EMP: A Simple Example and Equilibrium Problems with EMP: Example with Dual Variables. The general syntax of EMP annotations for equilibrium problems is introduced in section EMP Syntax for Equilibrium Problems.
`implicit`	Specifies a shared variable and its defining constraint. For details, see section Equilibrium Problems with Shared Variables.
`max`	This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the *maximization* problem of one agent.
`min`	This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the *minimization* problem of one agent.
`VI`	Indicates that the specifications that follow define a VI, as in this example.
`VIsol`	Specifies that the equation that follows is a shared constraint and prompts the EMP framework to use the MCP reformulation where a variational inequality is associated with the shared constraint. For more information and an example, see section Equilibrium Problems with Shared Constraints.

4.51.12.4 GAMS EMP Keywords for Embedded Complementarity Systems

EMP Keyword	Description
`DualEqu`	This keyword is followed by an equation-variable pair. It establishes a complementarity relationship between an external equation and the named variable of the model. An example is given above. See also the spatial price equilibrium model [**HARK-MONOP**].
`DualVar`	This keyword is followed by a variable-equation pair. It specifies that the variable is the dual of the equation. Examples are discussed in sections Equilibrium Problems with EMP: Example with Dual Variables and Embedded Complementarity Systems.

Note that problems with embedded complementarity systems can be recast as equilibrium problems. Details are given in section Embedded Complementarity Systems.

4.51.12.5 GAMS EMP Keywords for Bilevel Programming

EMP Keyword	Description
`bilevel`	Indicates that the specifications that follow relate to a bilevel programming problem. The keyword is follwed by the decision variable(s) of the upper-level problem and the definitins(s) of the lower-level problem(s). For examples and more details, see section Bilevel Programs.
`dualvar`	See above.
`max`	This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the *maximization* problem of one agent.
`min`	This keyword is followed by the objective variable, the decision variable(s) and the equation(s) of the *minimization* problem of one agent, as in this example.
`VI`	Indicates that the specifications that follow define a VI, as in this example.

4.51.12.6 GAMS EMP Keywords for Disjunctive Programming

EMP Keyword	Description
`bigM`	Indicates that the big M reformulation method shall be used.
`chull`	Indicates that a convex hull shall be used for the reformulation. Note that this is the default method.
`default`	Indicates that the specification that follows is the reformulation method.
`disjunction`	Indicates that the specification that follows is a disjunction. For a discussion of the general syntax following this keyword, see above.
`indic`	Indicates that indicator constraints shall be used as reformulation method.
`star (*)`	The symbol $*$ will be replaced by internal default binary variables, thus explicit binary variables that model the Boolean variables are not needed. Note that default binary variables may only be used if there are no logic equations in the model. For an example and more details, see above.

4.51.12.7 GAMS EMP Keywords for Stochastic Programming

EMP Keyword	Description
`chance`	Defines individual or joint chance constraints.
`cvar`	Synonym to cvarup.
`cvarlo`	This keyword assigns a variable to have the value $\underline{CVaR}_\alpha$, where α is a scalar that represents the confidence level for the Conditional Value at Risk. Note that `cvarlo` refers to the left tail of the distribution. For more details and examples, see section Conditional Value at Risk (CVaR).
`cvarup`	This keyword assigns a variable to have the value $\overline{CVaR}_\alpha$. α is a scalar that represents the confidence level for the Conditional Value at Risk. Note that `cvarup` refers to the right tail of the distribution.
`discrete`	Indicates that the specification that follows is the discrete distribution of one or more random variables.
`ExpectedValue`	This keyword is used to state that a variable is the expected value of a random variable.
`jrandvar`	This keyword is used to define discrete random variables that are *jointly* distributed. At least two random variables must be named. For an example, see the news vendor model [**NBDISCJOINT**].
`randvar`	This keyword declares that a parameter of the model is in fact a stochastic random variable. The keyword is followed by the name of the parameter and details about the probability distribution (discrete or parameteric). A list of all supported parametric distributions is given in Table 4.
`sample`	This keyword allows users to customize the size of the sample of one or more random variables from a continuous distribution and - optionally - to determine the variance reduction method to be used. An example and further details are given above. Note that without a valid LINDO license this is limited to the Normal and Binomial distributions with a maximum sample size of 10.
`setSeed`	This keyword sets the seed for the random number generator of the sampling routines that are called via the keyword sample. If `setSeed` is used in EMP annotations, the seed will be set once before all samples will be generated. Note that a valid LINDO license is required to use this keyword.
`stage`	This keyword is followed by a number and the names of the random variables, variables and equations that are assigned to the respective stage. Stage 1 is the default stage for all random variables, variables and equations that are not assigned a stage explicitly except for objective equation and variable. Their default stage is the highest stage in the model.

EMP Keyword	Description
`stageDefault`	This keyword is followed by a number. This specifies the default stage for all random variables, variables and equations that are not assigned a stage explicitly (except for objective equation and variable). If this keyword is not used explicitly, the default is 1.
`var`	Synonym to varup.
`varlo`	This keyword assigns a variable to have the value $\underline{VaR}_\alpha$, where α is a scalar that represents the percentile of the Value at Risk. Note that `varlo` refers to the left tail of the distribution. For more details and examples, see section Value at Risk (VaR).
`varup`	This keyword assigns a variable to have the value $\overline{VaR}_\alpha$, where α is a scalar that represents the percentile of the Value at Risk. Note that `varup` refers to the right tail of the distribution and α typically equals 0.95 or 0.9. For further information, see section Value at Risk (VaR).

Currently, two GAMS solvers can be used to solve stochastic programming models with EMP: DE and LINDO. Not all keywords mentioned above are supported by both solvers. The following table specifies which keywords can be used with which solvers. The keywords not mentioned in the table are supported by all solvers mentioned.

	DE	LINDO
`chance`	√	√
`jrandvar`	√	√
`randvar`(discrete)	√	√
`randvar`(parametric)	√	√
`sample`	√	√
`setSeed`		√
`var`	√	
`cvar`	√	
`ExpectedValue`	√	

Table 8: Solver capabilities

Note

In general, JAMS is the default solver for models of type `EMP`. However, since `JAMS` cannot handle stochastic EMP models, GAMS switches the solver to DE automatically when, a stochastic EMP model is tried to be solved with `JAMS`.

Further information about these solvers can be found in the corresponding solver manuals. The stochastic programming options for the solver LINDO might be of particular interest.

4.52 Accessing Model Libraries

Professor Paul Samuelson is fond of saying that he hopes each generation of economists will be able to `"stand on the shoulders"` of the previous generation. The library of models included with the GAMS system is a reflection of this desire. We believe that the quality of modeling will be greatly improved and the productivity of modelers enhanced if each generation can stand on the shoulders of the previous generation by beginning with the previous models and enhancing and improving them. The Model Libraries includes a large number of models, collectively organized as

- **GAMS Model Library** - includes GAMS models representing interesting and sometimes classic problems, ranged from production and shipment by firms, investment planning, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and many more.
- **GAMS Test Library** - includes GAMS models developed for testing and quality control, both for the GAMS base module and the many solvers distributed with the GAMS system.
- **GAMS Data Utilities Library** - includes GAMS models demonstrating various utilities to interface GAMS with other tools and applications such as spreadsheets and database interface.
- **GAMS EMP Library** - includes GAMS Extended Mathematical Programming (EMP) models that illustrate and test the capabilities of GAMS/EMP.
- **GAMS API Library** - includes GAMS Models used as scripts to compile and execute application programs in various programming languages interfacing to GAMS.
- **FIN Library** - includes GAMS practical financial optimization models described in the book `Practical Financial Optimization: Decision Making for Financial Engineers` by Consiglio, Nielsen and Zenios,
- **NOA Library** - includes GAMS nonlinear optimization applications models based on the book `Nonlinear Optimization Applications Using the GAMS Technology` by Neculai Andrei.
- **PSOPT Library** - includes GAMS optimization models based on the book `Power System Optimization Modelling in GAMS` by Alireza Soroudi.

The models included have been selected not only because they collectively provide strong shoulders for new users to stand on, but also because they represent interesting and sometimes classic problems. For example the trade-off between consumption and investment is richly illustrated in the Ramsey problem, which can be solved using nonlinear programming methods. Examples of other problems included in the library are production and shipment by firms, investment planning in time and space, cropping patterns in agriculture, operation of oil refineries and petrochemical plants, macroeconomics stabilization, applied general equilibrium, international trade in aluminum and in copper, water distribution networks, and relational databases.

Another criterion for including models in the library is that they illustrate the modeling capabilities GAMS offers. For example, the mathematical specification of cropping patterns can be represented handily in GAMS. Another example of the system's capability is the style for specifying initial solutions as staring points in the search for the optimal solution of dynamic nonlinear optimization problems.

Finally, some models have been selected for inclusion because they have been used in other modeling systems. Examples are network problems and production planning models. These models permit the user to compare how problems are set up and solved in different modeling systems.

Most of the models have been contributed by GAMS users. The submission of new models is encouraged. If you would like to see your model in a future release of the library, please send the model and associated documents and reports to GAMS Development Corporation.

4.52.1 Usage

4.52.1.1 Command Line Approach

One way to access the library is through command line. The following commands copy a model from the library directory into the current directory.

Command	Library to access
gamslib	**GAMS Model Library**
testlib	**GAMS Test Library**
datalib	**GAMS Data Library**
emplib	**GAMS EMP Library**
apilib	**GAMS API Library**
finlib	**FIN Library**
noalib	**NOA Library**
psoptlib	**PSOPT Library**

Take the command gamslib for an example, if you enter gamslib without any parameters, the command syntax will be displayed as shown below:

```
> gamslib modelname [target]
```

or

```
> gamslib modelnum [target]
```

where

- modelname is the modelname
- modelnum is the model sequence number, and
- target is the target file name.

If the target file name is not provided, the default is modelname.gms. The file will be automatically copied into the current working directory. For example, the [**TRNSPORT**] model from the **GAMS Model library** has sequence number 1 and could be copied in any of the following ways.

To copy [**TRNSPORT**] model file to target file trnsport.gms under the current directory:

```
> gamslib trnsport
```

```
> gamslib 1
```

To copy [**TRNSPORT**] model file to target file myname.gms under the current directory:

```
> gamslib trnsport myname
```

```
> gamslib 1 myname
```

The other commands have similar usage to gamslib command.

4.52.1.2 IDE Approach

A convenient way to access the model library on the major platforms is the "Model Library Explorer" from GAMS Studio. It is described in detail in the GAMS Studio documentation.

Users may define their own library by using a GLB file. These libraries can also be accessed through Studio.

4.53 Mathematical Programming System for General Equilibrium analysis (MPSGE)

- Introduction to MPSGE
- MPSGE Models in GAMS
- Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE
- Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae
- `A Library of Small Examples for Self-Study`
- `Linking Implan Social Accounts to MPSGE`
- The MPSGE guide is also available as `PDF`.

4.53.1 Introduction to MPSGE

MPSGE is a language used for formulating and solving Arrow–Debreu economic equilibrium models and exists as a subsystem within GAMS. The name stands for 'mathematical programming system for general equilibrium'. MPSGE provides a short-hand non-algebraic representation for the systems of nonlinear inequalities which underly general equilibrium models. The MPSGE framework is based on nested constant elasticity of substitution utility and production functions. The data requirements for a model include share and elasticity parameters, endowments, and tax rates for all the consumers and production sectors in the model. These may or may not be calibrated from a consistent benchmark equilibrium dataset.

The main benefit of using MPSGE is that modelers are released from having to write the equations of the model and the calibrated demand and supply functions. Experience shows that even computing the parameters of the functions is error-prone. The tabular input format of MPSGE facilitates a compact, non-algebraic representation of the equations of a model. Thus algebraic tedium and the scope for programming errors are reduced, and more attention can be paid to economic interpretation and testing of alternative formulations.

This chapter offers an introduction to MPSGE for users who are familiar with GAMS. For more information on MPSGE and details on more advanced features, see [156] and

[130]. In addition, a library of small examples for self-study is available `here` .

This chapter is organized as follows. We start with an overview of some basic economic ideas and general remarks on specifying an MPSGE model. Next, we introduce a first simple example. Then we demonstrate how intermediate demand and joint production are modeled with MPSGE. We follow with a third model that features a small open economy and also includes taxes on inputs and outputs, labor-leisure choice, classical unemployment, and nested demand functions. We conclude the chapter with an overview of MPSGE keywords and the syntax for specifying functions, some other features like domain restrictions on variable declarations, and a few brief comments on MPSGE-specific output in the GAMS listing file. Note that a full version of all MPSGE models that are discussed, including standard GAMS versions, are given in the Appendix.

4.53.1.1 Some Basic Economic Ideas

MPSGE models are based on the following economic ideas. In each model there are multiple interacting agents, where each agent solves an optimization problem. For example, consumers maximize utility subject to their budget constraint, and producers minimize their cost given available technology. Agent interactions are mediated by markets and prices and the optimal solution is an equilibrium where prices and activity levels have adjusted so that markets for produced goods and input factors clear. In equilibrium, each agent cannot do better by altering their behavior (given the constraints they face). In this section we give more details on these basic economic ideas. In particular, we offer brief introductions to consumer demand theory, producer supply theory and general economic equilibrium.

Consumer Demand Theory

Consumer choice is modeled using an optimization paradigm: consumers form a consumption bundle which maximizes welfare subject to a budget constraint. As an abstract notion, this seems plausible, but certain details must be dealt with to provide an operational application of the idea.

Consider the following example. Thomas lives in Ann Arbor where he spends 30% on housing and the rest of his income on other goods. This information is essentially an observation of a *benchmark equilibrium*, consisting of the prevailing prices and quantities of goods demanded. Now, Thomas has an employment offer in Berlin which pays 50% more than he currently earns, but he is hesitant to take the job since rental rates in Berlin are three times higher than in Ann Arbor. The question is: On purely economic grounds, should he move?

Thomas' choices are illustrated below. Point a represents the benchmark point, where he spends the amount H_a on housing in Ann Arbor. Point b represents a potential trade-off between housing and other goods in Berlin, where he spends a lower percentage of his income on housing and more on other goods.

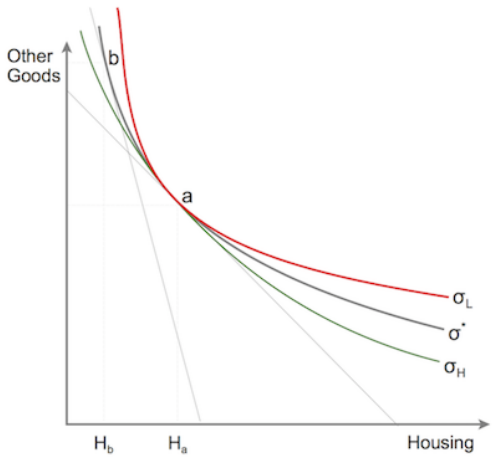


Figure 4.6 Thomas' Choices

Whether Thomas derives the same welfare from the choices represented by point b like those represented by point a depends on the *elasticity of substitution*. The elasticity of substitution is the willingness to exchange some amount of one good for more of the other good. The three curves in the figure above represent three elasticities of substitution. Note that the less convex (flatter) the curve is, the higher is the elasticity, i.e. the more willing is Thomas to substitue housing for other goods.

If the elasticity of substitution equals $\sigma^\star$, Thomas is *indifferent* between choices a and b, since both points lie on the respective *indifference curve*. However, if the elasticity of substitution is lower, say is equals σ_L, or higher, say it equals σ_H, then b will be worse than a and Thomas will not choose it. Hence, the

elasticity of substitution is crucial in determining whether Thomas should move on economic grounds. Note that the elasticity of substitution is a measure for Thomas' preferences.

The utility function can be deduced if prices and the associated choices are observed and the elasticity of substitition is given. Thus, in MPSGE, a utility function is represented by benchmark (or observed) quantities, benchmark prices and the associated elasticity of substitution. Benchmark quantities determine an anchor point for the set of indifference curves. Benchmark prices fix the slope of the indifference curve at that point. The elasticity of substitution is a measure of the curvature of the indifference curve and specifies the utility function unambiguously.

Producer Supply Theory

While consumers aim to maximize their utility subject to a budget, producers aim to reduce their cost of production given available technology. The activity of a firm is modeled as a *production function* that maps inputs into outputs. Inputs are primary factors like labor, capital, land, raw materials etc. and outputs are goods and services. Similarly to utility functions, in MPSGE, a production function is represented by reference quantities of reference prices of inputs and outputs, elasticities of substitution and elasticities of transformation.

Summarizing, the main principle for both, consumers and producers, is choice. Consumers choose quantities of goods given their preferences, prices and budgets. Producers choose quantities of primary factors and output levels given available technology. Available technology is modeled via elasticities of substitution between inputs and elasticities of transformation between outputs.

General Economic Equilibrium

As mentioned above, MPSGE is a system for modeling general economic equilibria. General economic equilibrium analysis studies the working of an economic system as a whole. The basic assumption is that there are two types of agents in the economy: producers and consumers. Producers are assumed to operate with constant return to scale and to be perfectly competitive price-taking firms. Further, they are assumed to select factor inputs and output levels in order to maximize their profits given available technology. Consumers are assumed to maximize their welfare subject to budget constraints (incomes and endowments). An economy is in economic equilibrium if all consumers spend their given incomes such that they gain maximum satisfaction, all firms in each sector minimize their costs given availabe technology, and prices (for primary factors and commodities) and activity levels have adjusted such that all agents cannot do better by altering their behavior given the constraints they face. In equilibrium, all markets have cleared: in both, goods and factors markets, total demand equals total supply.

Mathematically, such an economy is modeled as a system of weak inequalities, where each inequality is associated with a non-negative variable denoting a price or a quantity. If a particular weak inequality holds as an equation, then the associated variable is strictly positive. If it holds as a strict inequality, then the associated variable equals zero. This is a mixed complemantarity problem that can be formulated and solved with standard GAMS syntax. For details see section Mixed Complementarity Problems (MCPs). As we will demonstrate, MPSGE models can be translated to standard GAMS MCP models. However, the framework that MPSGE offers is much more user-friendly, since modelers do not have to input the complex equations manually.

4.53.1.2 Specifying an MPSGE Model

In this section we give an overview of the MPSGE model specification, a simple introductory example follows in the next section. Like any standard GAMS program, a program with an embedded MPSGE model begins with set and parameter definitions that will later be used in the model. The MPSGE model is specified within an $ontext / $offtext block.

First, the name of the model must be defined:

```
$MODEL:mymodel
```

Here `$MODEL` is an MPSGE keyword and the model name `mymodel` is a GAMS identifier.

In a GAMS program lines within $on/offtext are are treated as a block comment. However, if the preprocessor encounters the MPSGE model definition statement, it will recognize that an MPSGE model specification follows and will process the code accordingly.

Note

Like GAMS, MPSGE is not case sensitive.

Secondly, the variables for an MPSGE model are declared. In every model, there are three classes *central variables:*

- `$SECTORS:` variables for activity levels associated with constant returns to scale production sectors in the economy, which are non-negative,
- `$COMMODITIES:` variables for commodity prices including all final goods, intermediate goods and primary factors of production, which are also non-negative, and
- `$CONSUMERS:` variables for income levels, one for each "household" in the model, including any government entities.

In addition, models may include *auxiliary* variables (`$AUXILIARY`) that are used to introduce endogenous tax instruments or endogenous endowment quantities to a model.

Note

Unlike GAMS models, variables in an MPSGE model must be declared over an explicit domain. Only variables that have been declared as one of the four MPSGE variable types may be used in the MPSGE model. Error messages are generated for variables which are declared but not referenced.

Thirdly, the parameters for functions are specified in a structured manner. The general syntax for all possible entries is given is section Syntax for Production and Demand Functions below. The MPSGE framework uses these specifications to automatically generate the respective market-clearing and income-balance equations. There are two types of functions:

- `$PROD:` This block defines a production function, specifying inputs, outputs and elasticities of substitution and transformation. A production function must be given for each sector in the model. Note that most of the power and subtleties of the MPSGE framework center on the `$PROD` tables.
- `$DEMAND:` This block defines a demand function. This function represents preferences (using reference demands), initial factor endowments and elasticities of substitution. A demand function must be specified for each consumer in the model.

Finally, any auxiliary variable has an associated equilibrium condition that is defined by the user with conventional GAMS algebraic syntax and placed in a block called `$CONSTRAINT`.

These four parts - model definition, variable declarations, function specifications and possibly side constraints for auxiliary variables - are the core of the MPSGE model specification. The model specification is followed by the dollar control option `$offtext` and the following compiler directive:

```
$sysinclude mpsgeset mymodel
```

This compiler directive instructs MPSGE to compile the functions and generates an external file called MYMODEL.GEN. This external file is then loaded to the GAMS file with the following directive:

```
$include MYMODEL.GEN
```

Finally, a standard GAMS solve statement follows and the model will be handed for solution to one of the MCP solvers:

```
solve mymodel using mcp;
```

Observe that the appropriate GAMS model type is MCP.

Most often the model is first evaluated at the benchmark point to test whether it is peoperly calibrated and everything works as expected. This is achieved by setting the option iterlim to zero. Then one or more counterfactuals are solved: first some parameter is modified using standard GAMS syntax, then the $include statement is repeated and the model is solved again.

Note

"$sysInclude mpsgeset" allows for an optional argument -mt=0 or 1 after the model name. The default value for the argument mt can be controlled via the double dash option --MPSGEMT=0 or 1. If the mt option is set to 1 the MODEL.GEN file is created in the GAMS scratch directory. Hence the $include before the solve needs to read $include "%gams.scrdir%MODEL.GEN". This allows to run multiple MPSGE jobs with the same model in the same working directory. The default of this option is 0. The model **hansmge** demonstrates the use.

4.53.1.3 An Introductory Example: Two Goods and Two Factors in a Closed Economy

In the canonical 2x2 model, two final goods, X and Y, are produced with two primary factors, labor L and capital K, and there is a single representative agent (consumer) RA. The model is defined by technology, preferences and endowments. We can describe prefrences by a utility function which provides an ordinal ranking of consumption levels for X and Y. We assume that both factors are in fixed supply, thus in equilibrium, factor endowments equal factor demands, and the supply of the two goods equal their demand.

The problem can be cast as consisting of three production activities, X, Y and U, and five markets X, Y, U, K and L. The initial data for the model are given by the following matrix.

	Production Sectors			Consumers	
Markets	X	Y	U	RA	Row Sums
PX	100		-100		0
PY		50	-50		0
PU			150	-150	0
PK	-50	-20		70	0
PL	-50	-30		80	0
Column Sums	0	0	0	0	

The data describes a representative equilibrium. Rows correspond to markets, where PX, PY, PU, PK and PL are the prices for the commodity X, the commodity Y, the utility U, and capital and labor respectively. Positive entries denote the value of commodity flows *into* the economy (sales or factor supplies) and negative entries represent the value of commodity flows *out* of the economy (factor demands and good demands). Observe that the sum of a row will equal zero if the market clears, i.e. the total amount of commodity that flows into the economy equals the total amount of this commodity that flows out of the economy. The matrix is called *balanced* if all rows and columns sum to zero.

Note that the layout of the matrix ensures that a complete list of the transactions associated with an activity is given in each production column. The sum of a production sector column is zero if the value of the outputs equals the costs of the inputs. A consumer column is balanced if the sum of primary factor sales equals the value of final demands. Hence columns that sum to zero indicate zero profits. This type of matrix is related to the concept of a *social accounting matrix* (or short: *SAM*).

The problem can be modeled with GAMS MPSGE as follows:

```
Parameter   endow     index of labour endowment     / 1.0 /;

*       MPSGE model declaration follows

$ontext
$MODEL:twobytwo

$SECTORS:
    X   ! Activity level for sector X -- benchmark=1
    Y   ! Activity level for sector Y -- benchmark=1
    U   ! Activity level for sector U -- benchmark=1

$COMMODITIES:
    PX  ! Relative price index for commodity X -- benchmark=1
    PY  ! Relative price index for commodity Y -- benchmark=1
    PU  ! Relative price index for commodity U -- benchmark=1
    PL  ! Relative price index for labor -- benchmark=1
    PK  ! Relative price index for capital -- benchmark=1

$CONSUMERS:
    RA  ! Income level for representative agent -- benchmark=150;

$PROD:X s:1
        O:PX   Q:100
        I:PL   Q: 50    ! Variable LX in the algebraic model
        I:PK   Q: 50    ! Variable KX in the algebraic model

$PROD:Y s:1
        O:PY   Q: 50
        I:PL   Q: 20    ! Variable LY in the algebraic model
        I:PK   Q: 30    ! Variable KY in the algebraic model

$PROD:U s:1
        O:PU   Q:150
        I:PX   Q:100    ! Variable DX in the algebraic model
        I:PY   Q: 50    ! Variable DY in the algebraic model

$DEMAND:RA
        D:PU
        E:PL   Q: (70*endow)
        E:PK   Q: 80
```

```
$offtext

* Compiler directive instructing MPSGE to compile the functions

$sysinclude mpsgeset twobytwo

* Benchmark replication

twobytwo.iterlim = 0;
$include TWOBYTWO.GEN
solve twobytwo using mcp;
abort$(abs(twobytwo.objval) gt 1e-7) "*** twobytwo does not calibrate ! ***";
twobytwo.iterlim = 1000;

*   Counterfactual : 10% increase in labor endowment

endow = 1.1;

*   Solve the model with the default normalization of prices which
*   fixes the income level of the representative agent.  The RA
*   income level at the initial prices equals 80 + 1.1*70 = 157.

$include TWOBYTWO.GEN
solve twobytwo using mcp;
```

Note that an algebraic version of this model using standard GAMS syntax is given in the Appendix.

After the model statement where the model is named, variables are declared in each of the three central variable classes. Unlike conventional GAMS syntax, in MPSGE, trailing comments in variable declarations (starting with the symbol `!`) are interpreted as variable descriptors that will appear in the listing file.

Then three production blocks `$PROD` - one for each sector - follow. Output parameters are specified in the line starting with the label `O` and input parameters are given in the line starting with the label `I`, the respective quantities are listed in the field `Q`. Note that the first input in the production block for sector `X` represents labor demand in sector X and the second input represents capital demand in sector `X`. Similarly, the inputs in the production block for sector `Y` represent labor and capital demand for sector `Y`. In the production block for sector `U`, the inputs represent demand for `X` and `Y` in sector `U`. In the field `s:` next to the name of the sector the elasticity of substitution between the inpiuts is specified.

A demand block `$DEMAND` for the consumer `RA` completes the model specification. Demand parameters are specified in the line starting with the label `D` and endowment parameters are given in the line starting with the label `E` . Note that all possible entries for production and demand blocks are discussed in section Syntax for Production and Demand Functions. Observe that there is no algebraic formulation of equations, only parameter specifications. Given this input, the MPGSGE frameworks generates the respective equations automatically.

Note that it is not necessary to fix a numeraire. If a numeraire is not specified, the normalization of prices is arbitrary. For example, in the model above, `RA` is used as a numeraire and the following remark appears in the listing file:

```
Default price normalization using income for RA
```

Note that the version of this model in the Appendix includes two additional counterfactual scenarios, where first sector `X` is used as numeraire commodity and then the wage rate `PL` is fixed as numeraire. In the Appendix, there are also two alternative versions of the model above: a model in standard GAMS syntax where the equations that are automatically generated by the MPSGE framework are given explicitly, and an MPSGE model that uses vector notation instead of scalars.

4.53.1.4 Modeling Intermediate Demand and Joint Production

In this section we extend the two by two model above in the following three ways:

1. We distinguish between production sectors and produced goods.
2. Each sector produces both goods, so we have *joint production*.
3. In addition to primary factors like labor and capital, goods enter the production process as inputs. Thus we have *intermediate demand*.

The full MPSGE model and MCP and NLP versions of the model are given in the Appendix. Here we reproduce and discuss only selected code snippets. The main difference to the simple model above is the specification of the production block:

```
$PROD:X(j)   s:1    t:1
      O:P(i)    Q:make0(i,j)    ! S(i,j) in the MCP and NLP models
      I:P(i)    Q:use0(i,j)     ! D(i,j) in the MCP and NLP models
      I:PF(f)   Q:fd0(f,j)      ! FD(f,j) in the MCP and NLP models
```

Note that the supply is specified in the output line, the intermediate demand is given in the first input line and the factor demand is listed in the second input line. Thus each sector `j` produces the goods `i` using the intermediate goods `i` and the primary factors `f`. Consider the fully spelled-out production block for sector `s1`:

```
$PROD:X("s1")   s:1    t:1
      O:P("g1")        Q:6.0
      O:P("g2")        Q:2.0
      I:P("g1")        Q:4.0
      I:P("g2")        Q:2.0
      I:PF("labor")    Q:1.0
      I:PF("capital")  Q:1.0
```

At benchmark, sector `s1` produces 6 units of good `g1` and 2 units of good `g2`. The input for this output is 4 units of the good `g1`, 2 units of the good `g2` and labor (one unit) and capital (one unit).

Note

> *Double* quotes have to be used if a singleton set element is referenced in an MPSGE model. Single quotes are not recognized.

Observe that the technology used to combine these four production factors is based on a Cobb-Douglas production function, i.e. the elasticity of substitution equals 1, as indicated by the value in field `s`. In addition, the curvature of the production possibility frontier (the constant elasticity of transformation function) is given in field `t`.

4.53.1.5 Modeling a Small Open Economy

An open economy is one in which goods are traded on international markets, typically at fixed international prices, i.e. there are imports and exports. One class of models for small open economies are known as "123 models" [42]. In the original 123 model, there was one small country, two producing sectors and three goods (one domestic good, one import good and one export good). The 123 model we present and discuss in this section has exogenously given world prices for imports and exports, taxes, labor-leisure choice, classical unemployment and joint production. The programming language keeps track of tax revenue flows, equations which are otherwise tedious to program. The model is a tiny version of the open economy GTAP model [115]. It follows the canonical microecomic optimization framework: (i) consumers maximize welfare subject to a budget constraint with fixed levels of investment and public expenditure, and (ii) producers combine intermediate inputs and primary factors at least cost for given technology.

Note that the full MPSGE model `mge123` and an algebraic version are given in the Appendix. In addition, we introduce and discuss a version of the model that includes nesting. Note further, that the data for the model and its variants are included in a separate GAMS file. Observe that the input-output matrix in the data file provides data on value flows for primary production factors, intermediate goods and final consumption products. In the following, we comment on selected code snippets and thereby discuss the most important features of the model.

Consider the production block for the entire production of the small economy:

```
$PROD:Y  t:etadx s:esubkl
        O:PD    Q:d0    P:1                              ! YD
        O:PX    Q:x0    P:px0   A:GOVT  T:tx             ! YX
        I:RK    Q:kd0   P:rr0   A:GOVT  T:tk             ! KD
        I:PL    Q:ly0   P:pl0   A:GOVT  T:tl N:TAU_TL    ! LY
```

Note that the first output line represents production for the domestic market, which is denoted by the variable `YD` in the algebraic version of the model. The reference price at benchmark is given in the new field `P`.

Note

The default value for both P and Q fields is 1.0

The second output line represents production for the export market (variable `YX` in the algebraic version). Observe that taxes are specified in the two new fields `A` and `T:` the tax recipient is specified in field `A` and the tax rate is listed in field `T`. Note that the tax recipient must reference a `$CONSUMER` variable and the tax rate `tx` (tax rate on exports) is given *exogenously*.

Attention

In MPSGE, the way exogneous taxes are computed depends on whether they are imposed on outputs or inputs. Taxes on *outputs* are specified on a *gross* basis. Hence, if a tax on outputs has proportional rate t, the producer price will be $p(1-t)$, where p is the market price. However, taxes on *inputs* are specified on a *net* basis. Hence, if a tax on inputs has ad valorem rate t , the user cost will be $p(1+t)$, where p is the market price.

The first input line represents capital demand (variable `KD` in the algebraic version). The model includes investment by the private (domestic) households, thus the variable `RK` denotes a rental price index. Observe that capital is taxed exogenously at the capital tax rate `tk`.

The second input line represents labor demand (variable `LY` in the algebraic version). In addition to an exogenous labor tax `tl`, an *endogenous* tax is imposed on this input. The endogenous tax rate is determined by the auxiliary variable `TAU_TL`, which is the entry in the field `N` at the end of the line. Note that an equation associated with the auxiliary variable is specified in the following block:

```
$CONSTRAINT:TAU_TL
        GOVT =e= PA * g0;
```

Note

When an auxiliary variable is fixed no constraint is required. When the lower bound of the auxiliary variable is less than the upper bound, a constraint is required.

Note further, that the variable TAU_TL is a wage replacement tax and at benchmark, it is fixed at zero. It is positive in two of the four counterfactual scenarios.

Domestic demand is modeled using the Armington assumption: consumers demand one single good which is an aggregation of *composite goods* consisting of the goods produced domestically and the imported versions, where imports and domesctic goods in the same sector are imperfect substitutes. The aggregated composite good is given by a constant elasticity of substitution aggregation function of the domestic goods and the imported goods with elasticity of substitution sigmadm. In our model, sigmadm equals 4 (domestic versus imported goods). The production of the Armington good is specified in the following block:

```
$PROD:A  s:sigmadm
        O:PA    Q:a0    A:GOVT  t:ta
        I:PD    Q:d0                        ! DA
        I:PM    Q:m0    p:pm0 A:GOVT t:tm   ! MA
```

Note that both inputs are intermediate goods. The first input line represents domestic absorption (variable DA in the algebraic version of the model) and the second input line represents imports (variable MA in the algebraic version). Imports are taxed with the import tariff rate tm and the composite good is taxed with the excise and sales tax rate ta. Both taxes are exogenous.

In addition, the model contains production blocks for imports and exports. Foreign exchange (denoted by the MPSGE variable PFX) is used in these two production blocks. It is the input in the production block for imports and the output in the production block for exports.

The two consumers in the model are the government (MPSGE variable GOVT) and aggregated private households (MPSGE variable HH). The demand block for the government agent is as follows:

```
$DEMAND:GOVT
        E:PFX   Q:bopdef
        E:PA    Q:dtax
        E:PA    Q:g0            R:TAU_LS
        D:PA
```

Note that the first endowment line represents the balance of payment in the current account (denoted by the $COMMODITIES variable PFX). In our model, exports exceed imports resulting in the deficit bopdef. If imports are larger than exports there will be a surplus. The other two endowment lines represent taxes that are paid by the households: a direct tax dtax and a lumpsum labor replacement tax that is specified as an auxiliary variable and determined by the respective complementarity equation. In this model the labor replacement tax is either a lumpsum tax (TAU_LS) or it is levied on labor at a tax rate of TAU_TL (see above). Note that the direct tax and the lumpsum tax feature in the demand block for HH as *negative* endowments. Negative endowments represent payments to be made. The demand block for HH follows:

```
$DEMAND:HH  s:sigma
        E:PA    Q:(-g0)         R:TAU_LS
        E:PA    Q:(-dtax)
        E:RK    Q:kd0
        E:PA    Q:(-i0)
        E:PL    Q:(ly0+l0)      ! Labor endowment = ly0+l0 - UR * (ly0+l0)
        E:PL    Q:(-(ly0+l0))   R:UR
        D:PA    Q:c0
        D:PL    Q:l0
```

Households demand two goods: the Armington composite good PA (first demand line) and leisure (second demand line). Note that the first two endowment lines represent taxes as discussed above. The third and and fourth endowment line relate to capital: the third line represents the income generated by capital investment and the fourth line represents capital investments. The last two endowment lines refer to labor. Labor endowment for the aggregated households equals the sum of labor and leisure minus the percentage of unemployment. Note that at benchmark, the unemployment rate UG is fixed at zero, other scenarios are explored in the counterfactuals.

In model mge123, the rate of substitution for domestic and imported goods that constitute the Armington good PA is fixed at sigmadm = 4. Suppose that we want more control over substitution possibilities. For example, we may wish to model a substitution rate in final demand that is different than for other uses of the Armington good. MPSGE facilitates modeling this additional degree of freedom through *nesting*. Consider the following demand block for HH from model mgenested, a variant of model mge123:

```
$DEMAND:HH  s:sigma    c:sigmac
        E:PA    Q:(-g0)          R:TAU_LS
        E:PA    Q:(-dtax)
        E:RK    Q:kd0
        E:PA    Q:(-i0)
        E:PL    Q:(ly0+l0)           ! Labor endowment = ly0+l0 - UR * (ly0+l0)
        E:PL    Q:(-(ly0+l0))   R:UR
        D:PD    Q:cd0    c:
        D:PM    Q:cm0    c:
        D:PL    Q:l0
```

Note that in this formulation the endowment lines are unchanged and we have three demand lines. The first two demand lines have an additional field (c:). This tag indicates that these two goods are part of a nest: they are combined to an intermediate good called C. The rate of substitution for the goods in the nest is given in the first line of the demand block: c:sigmac. Observe that except for the substitution rate, C equals PA. The structure of the final demand can be presented graphically using the followong nesting diagram:

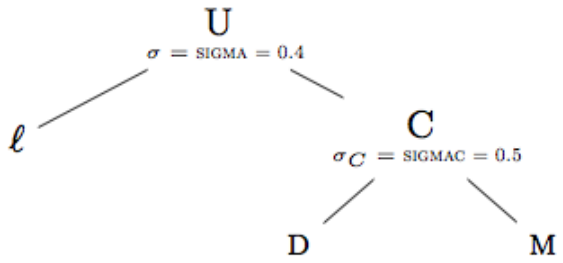


Figure 4.7 Structure of nested demand for HH

The nested utility function, expressed as a *unit function* is:

$$U = \left[\theta\left(\frac{\ell}{\bar{\ell}}\right)^{1-1/\sigma} + (1-\theta)\left(\alpha\left(\frac{D}{\bar{D}}\right)^{1-1/\sigma_C} + (1-\alpha)\left(\frac{M}{\bar{M}}\right)^{1-1/\sigma_C}\right)^{\frac{1-1/\sigma}{1-1/\sigma_C}}\right]^{1/1-1/\sigma}$$

in which θ is the budget share of leisure (ℓ) in aggregate expenditure, and α is the budget share of domestic goods (D) in commodity expenditure, both evaluated at the benchmark point:

$$\theta = \frac{\bar{\ell}}{\bar{\ell} + \bar{D} + \bar{M}}$$

and

$$\alpha = \frac{\bar{D}}{\bar{D} + \bar{M}}$$

The demand for leisure is a function of extended income (HH):

$$\ell = \bar{\ell} \left(\frac{p_U}{p_\ell} \right)^{\sigma} \frac{HH}{p_U \left(\bar{\ell} + \bar{D} + \bar{M} \right)}$$

The demand for domestic and imported goods is a function of both the top-level elasticity of substitution, σ, as well as the elasticity of substitution between consumption goods, σ_C:

$$D = \bar{D} \left(\frac{p_U}{p_C} \right)^{\sigma} \left(\frac{p_C}{p_D} \right)^{\sigma_C} \frac{HH}{p_U \left(\bar{\ell} + \bar{D} + \bar{M} \right)}$$

and

$$M = \bar{M} \left(\frac{p_U}{p_C} \right)^{\sigma} \left(\frac{p_C}{p_M} \right)^{\sigma_C} \frac{HH}{p_H (\bar{\ell} + \bar{D} + \bar{M})}$$

The graph illustrates the two-level nesting structure. Domestic goods and imports are combined in the second-level nest with an elasticity of substitution of `sigmac`. Here we use the identifier `C`.

Note

> The name space of nesting assignments is segmented from that of sets, parameters and variables in the GAMS code. Two set idenifiers are predefined: `s:` and `t:`. All other identifiers are accepted provided they have four or fewer characters.

The full model `mgenested` is given in the Appendix. Observe that as the substitution rate of the Armington good in final demand has changed, some parameters are modified at the beginning of the program in order to recalibrate the model.

A tariff reform is similar to a tax reform, it has a major influence on the structure of the budget of a government. Suppose we wish to keep government expenditure constant and experiment with different scenarios that represent different combinations of tax instruments. For example, at benchmark, in our model the unemployment rate `UR` and the endogenous wage tax `TAU_TL` are fixed at zero. Four alternative closures are explored in the counterfactuals. They depend on revenue replacement (lumpsum versus wage tax) and labor market (flexible versus fixed wages). An overview of the outcomes are given in the following table:

```
----   1914 PARAMETER report  Tariff Remove with Revenue Replacement (% impact)

          Lump Sum    Lump Sum    Wage Tax    Wage Tax
          Flexible  Rigid Wage    Flexible  Rigid Wage

PFX            4.6         4.6        13.0         9.4
PD            -2.1        -2.1         5.9         2.6
RK             0.6         0.6         7.9        -1.6
PA            -4.5        -4.5         3.3 2.22045E-14
GOVT        3299.9      3299.9      3574.4      3458.3
HH         40184.6     40184.6     42403.1     38219.6
PX             4.6         4.6        13.0         9.4
W              0.4         0.4         0.3        -7.5
Y              0.3         0.3        -0.5        -6.3
A              0.7         0.7 -4.15640E-2        -5.3
M             13.7        13.7        13.0         7.5
X             18.7        18.7        17.6        10.2
YD            -8.8        -8.8        -9.5       -14.6
YX            18.7        18.7        17.6        10.2
KD     3.800191E-8 -7.9403E-11 -2.21554E-9 2.22045E-14
LY             0.6         0.6        -0.9       -11.9
DA            -8.8        -8.8        -9.5       -14.6
MA            13.7        13.7        13.0         7.5
C              1.0         1.0 -5.89421E-2        -7.5
LD            -0.9        -0.9         1.2        -7.5
PM             4.6         4.6        13.0         9.4
TAU_LS        38.1        38.1
TAU_TL                                  9.1        11.9
UR                                                10.0
```

Note that the classical unemployment part in this model provides a good way to motivate the usefulness of the modeling framework. We easily produce a model which illustrates the importance of the labor market formulation. When the real wage is downward rigid, replacement of tariffs with wage taxes reduces welfare (denoted by `W`) by 7.9%, output by 6.6% and employment by 10%. If the wage is flexible, replacement of tariffs by wage taxes leads to a 0.1% decrease in welfare.

4.53.1.6 MPSGE Keywords and Syntax

In the presentation of the general syntax in this section, we use the usual GAMS syntax symbols: `[ ]` (the enclosed construct is optional), { } (the enclosed construct may be repeated zero or more times) and | (exclusive `OR`). `EOL` means "end-of-line" and `num_expr` denotes a number or GAMS numerical expression that may include a dollar condition for exception handling.

MPSGE Keywords

MPSGE provides nine keywords that are used to specify an MPSGE model. They are given in Table 1. Note that these keywords are *not* GAMS reserved words, i.e. they are not keywords in the GAMS code apart from the MPSGE model specification.

MPSGE Keyword and Syntax	Description
`$MODEL:model_name`	This keyword is used to assign the identifier `model_name` to the model. `Model_name` must be a valid file name, since it is used to form `model_name.GEN`. This must be the first statement within the `$ontext - $offtext` block containing the MPSGE code; all lines in an `$ontext - $offtext` block before this keyword are treated as comments.

`$SECTORS:`
`sect1 ! Description1`
`sect2 ! Description2`
...
	This keyword is used to declare one or more variables for sectors that are used in the model.

`$COMMODITIES:`
`com1 ! Description1`
`com2 ! Description2`
...
	This keyword is used to declare one or more variables for commodities that are used in the model.

`$CONSUMERS:`
`cons1 ! Description1`
`cons2 ! Description2`
...
	This keyword is used to declare one or more variables for consumers that are used in the model.

`$AUXILIARY:`
`aux1 ! Description1`
`aux2 ! Description2`
...
	This keyword is used to declare one or more auxiliary variables. It is only used in models with side constraints and endogenous taxes or rationed endowments.

`$PROD:sector`	This keyword is used to specify a production function. A production function must be specified for each sector in the model. Note that `sector` must have been previously declared with a `$SECTORS` statement.
`$DEMAND:consumer`	This keyword is used to define a demand function. A demand function must be specified for each consumer in the model. Note that `consumer` must have been previously declared with a `$CONSUMERS` statement.
`$CONSTRAINT:auxiliary`	This keyword is used to specify a side constraint to be associated with the auxiliary variable `auxiliary`. Note that `auxiliary` must have been previously declared with an `$AUXILIARY` statement.
`$REPORT:`	This keyword identifies the set of additional variables to be calculated. These variables are used for reports and include outputs and inputs by sector, and demands and welfare by individual consumers. The variables declared in `$REPORT` blocks may not be used in model equations. Note that we did not discuss report blocks in this chapter. For details see [130] .

Table 1: MPSGE Keywords

Syntax for Production and Demand Functions

In addition to the keywords above, MPSGE provides a fixed structure for specifying production and demand functions. This structure has a tabular format with pre-specified fields (or labels). The names of most fields are single letters that are reserved words in MPSGE. The exception are nest identifiers: they are arbitray names with up to four characters.

The syntax for the definition of a production function is as follows.

```
$PROD:sector    [s:num_expr]   [t:num_expr]   [a:num_expr {b:num_expr}]
O:commodity1    [Q:num_expr]   [P:num_expr]   [A:consumer]   [T:num_expr  {T:num_expr}]  [N:auxiliary
I:commodity2    [Q:num_expr]   [P:num_expr]   [A:consumer]   [T:num_expr  {T:num_expr}]  [N:auxiliary
```

Note that the numerical expression num_expr may be a parameter or a value.

Each production function block starts with the MPSGE keyword $PROD and the respective sector. Note that the variable sector must have been previously defined in the $SECTORS block. The labels that follow are optional. They include the following:

- s: Top level elasticity of substitution between inputs. The default value is zero.
- t: Elasticity of transformation between outputs in production. Can be zero, but not infinity.
- a:,b:,... Elasticities of substitution in individual input nests. Here a and b are nest identifiers.

Each production block has at least one output line O and one input line I. The value in the fields O and I is a commodity that has been previously declared in the $COMMODITIES block. Note that only this first field is mandatory, all other entries are optional and depend on the model. The valid labels for both lines include the following:

- Q: Reference quantity. Default value is 1. When specified, it must be the second entry.
- P: Reference price. Default value is 1.
- A: Tax revenue agent, the entry must be a consumer that has been previously defined in the $CONSUMERS block. This field must appear *before* the corresponding T or N field.
- T: Tax rate of the *exogenous* tax rate. Note that more that one tax rate may be listed in one line.
- N: *Endogenous* tax, the entry is an auxiliary variable that has been previously defined in the $AUXILIARY block.
- M: Endogenous tax multiplier. The ad valorem tax rate is the product of the value of the endogenous tax and this multiplier. Note that if the M field is omitted in a line with an N field, M:1 will be assumed. Observe that the M field cannot be included in the absence of an N field.
- a:,b:,.. Nesting assignments. Only one such label may appear per line.

The syntax for the specification of a demand function is as follows.

```
$DEMAND:consumer    [s:num_expr]   [a:num_expr {b:num_expr}]
D:commodity1        [Q:num_expr]   [P:num_expr]                          [a: | b:]
E:commodity2        [Q:num_expr]   [R:auxiliary]
```

Each demand function block starts with the MPSGE keyword $DEMAND and the respective consumer. Note that the variable consumer must have been previously declared in the $CONSUMERS block. The labels that follow are optional. They include the following:

- s: Top level elasticity of substitution between demands.
- a:,b:,... Elasticities of substitution in individual demand nests.

Each demand block has at least one demand line D and it may have one or more endowment lines E. The value in the fields D and E is a commodity that has been previously declared in the $COMMODITIES block. Note that only this first field is mandatory, all other entries are optional and depend on the model. The valid labels in a D line include the following:

- Q: Reference quantity. Default value is 1. When specified, it must be the second entry.
- P: Reference price. Default value is 1.

- `a:,b:,..` Nesting assignments. Only one such label may appear per line.

The valid labels in an `E` line include the following:

- `Q:` Reference quantity. Default value is 1. When specified, it must be the second entry.
- `R:` Rationing instrument, the entry is an `auxiliary` variable that has been previously defined in the `$AUXILIARY` block.

Auxiliary constraints in MPSGE models conform to standard GAMS equation syntax. They may refer to any of the four classes of variables, `$SECTORS`, `$COMMODITIES`, `$CONSUMERS` and `$AUXILIARY`, but they may not reference variables names declared within a `$REPORT` block. Complementarity conditions apply to upper and lower bounds on auxiliary variables and the associated constraints. For this reason, the orientation of the equation is important. When an auxiliary variable is designated `POSITIVE` (the default), the auxiliary constraint should be expressed as an equation of the type `=g=`. If an auxiliary variable is designated `FREE`, the associated constraint must be expressed as an equality (`=e=`).

Domain Restrictions on Variable Declarations

Domain restrictions on variable declarations are a crucial difference between models formulated in GAMS and those formulated in MPSGE. In GAMS, variables are defined over a domain, but the explicit domain used in the model is determined by CMEX at the point when the model is generated. In an MPSGE model, users are required to declare the explicit domain for every variable. If they include variables which are not referenced in the model, they will get a "No source" or "No sink" error message. If a variable is referenced which is not declared, an error message will be generated by the MPSGE function evaluator.

Consider the following example of a sector declaration in MPSGE:

```
$SECTORS:
   Y(i)$y0(i)
   M(i)$m0(i)
...
```

The corresponding production blocks need to have the corresponding exception operators:

```
$PROD:Y(i)$y0(i)
   ...

$PROD:M(i)$m0(i)
   ...
```

For more information on exception handling in GAMS, see chapter Conditional Expressions, Assignments and Equations.

The Nest Labor Operator and the Spanning Operator

The nest labor operator (`.tl`) is used to create a set of nests. The following example serves as illustraton and is self-explanatory:

```
$PROD:Y     s:0.5    m:0   i.tl(m):esubdm(i)   va:1
   O:PY     Q:y0
   I:PD(i)  Q:d0(i)        i.tl:
   I:PM(i)  Q:m0(i)        i.tl:
   I:PK     Q:k0                               va:
   I:PL     Q:l0                               va:
```

The spanning operator (`#`) is a way to provide multiple inputs of one commodity. For example, a commodity with price `PM` is a wholesale/retail trade margin. The benchmark value of margins on commodity `i` is `md0(i)`, and the net of margin (wholesale) value of commodity `i` sales is `d0(i)`. If goods trade off in a Cobb-Douglas nest at the gross of margin price, we can represent this in an MPSGE model as:

```
$PROD:A       s:1         i.tl:0
   O:PA       Q:a0
   I:P(i)     Q:d0(i)     i.tl:
   I:PM#(i)   Q:md0(i)    i.tl:
```

4.53.1.7 MPSGE-Specific Output

The output in the listing file of an MPSGE model has some distinctive features. Note that the MPSGE model specification is reproduced in the echo print. After the directive `$offtext`, the echo print is interrupted and a symbol reference map is inserted. In this map, both, the identifiers defined using standard GAMS syntax and MPSGE variables are listed. In addition to the standard GAMS data types, the following MPSGE shorthand symbols may appear in this listing:

Shorthand Symbol	MPSGE Data Type
`ACTIV`	sector variable
`AUXIL`	auxiliary variable
`CONSU`	consumer variable
`PRICE`	commodity variable

Table 2: Shorthand Symbols for MPSGE Data Types

For example, the symbol reference map for the first simple model in vector notation reads as follows:

```
Symbol Listing

      Symbol      Type       References
      ==========  =====      ==================================================
      CONS        PARAM          19   23
      DEMAND      PARAM          20
      ENDOW       PARAM          24
      F           SET             9   16   16   24   24
      FACTOR      PARAM          16
      I           SET             3   14   15   15   16   20   20
```

```
      PC          PRICE          8   15   20
      PF          PRICE          9   16   24
      PU          PRICE          7   19   23
      RA          CONSU         12   22
      SUPPLY      PARAM         15
      U           ACTIV          4   18
      Y           ACTIV          3   14
```

Note that if the normalization was done automatically, a comment will appear after the solve summary detailing which variable was used as a numeraire. For example,

```
Default price normalization using income for RA
```

Observe that the lower, level, upper and marginal values of the MPSGE variables are given in the solution listing like for any other GAMS variable. For more information on standard GAMS output, see chapter GAMS Output.

4.53.1.8 Appendix

Three Versions of Model TWOBYTWO

MPSGE Model TWOBYTWO

```
$title  A two by two general equilibrium model -- scalar GAMS/MPSGE

parameter   endow     index of labour endowment     / 1.0 /;

*       MPSGE model declaration follows

$ontext
$MODEL:twobytwo

$SECTORS:
    X   ! Activity level for sector X -- benchmark=1
    Y   ! Activity level for sector Y -- benchmark=1
    U   ! Activity level for sector U -- benchmark=1

$COMMODITIES:
    PX  ! Relative price index for commodity X -- benchmark=1
    PY  ! Relative price index for commodity Y -- benchmark=1
    PU  ! Relative price index for commodity U -- benchmark=1
    PL  ! Relative price index for labor -- benchmark=1
    PK  ! Relative price index for capital -- benchmark=1

$CONSUMERS:
    RA  ! Income level for representative agent -- benchmark=150;

$PROD:X s:1
        O:PX   Q:100
        I:PL   Q: 50    ! Variable LX in the algebraic model
        I:PK   Q: 50    ! Variable KX in the algebraic model

$PROD:Y s:1
        O:PY   Q: 50
```

```
        I:PL   Q: 20    ! Variable LY in the algebraic model
        I:PK   Q: 30    ! Variable KY in the algebraic model

$PROD:U s:1
        O:PU   Q:150
        I:PX   Q:100    ! Variable DX in the algebraic model
        I:PY   Q: 50    ! Variable DY in the algebraic model

$DEMAND:RA
        D:PU
        E:PL   Q: (70*endow)
        E:PK   Q: 80

$offtext

* Compiler directive instructing MPSGE to compile the functions

$sysinclude mpsgeset twobytwo

* Benchmark replication

twobytwo.iterlim = 0;
$include TWOBYTWO.GEN
solve twobytwo using mcp;
abort$(abs(twobytwo.objval) gt 1e-7) "*** twobytwo does not calibrate ! ***";
twobytwo.iterlim = 1000;

*   Counterfactual : 10% increase in labor endowment

endow = 1.1;

*   Solve the model with the default normalization of prices which
*   fixes the income level of the representative agent.  The RA
*   income level at the initial prices equals 80 + 1.1*70 = 157.

$include TWOBYTWO.GEN
solve twobytwo using mcp;

parameter   equilibrium Equilibrium values;

*   Save counterfactual values:

equilibrium("X.L","RA=157") = X.L;
equilibrium("Y.L","RA=157") = Y.L;
equilibrium("U.L","RA=157") = U.L;

equilibrium("PX.L","RA=157") = PX.L;
equilibrium("PY.L","RA=157") = PY.L;
equilibrium("PU.L","RA=157") = PU.L;
equilibrium("PL.L","RA=157") = PL.L;
equilibrium("PK.L","RA=157") = PK.L;

equilibrium("RA.L","RA=157") = RA.L;
equilibrium("PX.L/PX.L","RA=157") = PX.L/PX.L;
equilibrium("PY.L/PX.L","RA=157") = PY.L/PX.L;
equilibrium("PU.L/PX.L","RA=157") = PU.L/PX.L;
equilibrium("PL.L/PX.L","RA=157") = PL.L/PX.L;
equilibrium("PK.L/PX.L","RA=157") = PK.L/PX.L;
```

```
equilibrium("RA.L/PX.L","RA=157") = RA.L/PX.L;

*   Fix a numeraire price index and recalculate:

PX.FX = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

equilibrium("X.L","PX=1") = X.L;
equilibrium("Y.L","PX=1") = Y.L;
equilibrium("U.L","PX=1") = U.L;

equilibrium("PX.L","PX=1") = PX.L;
equilibrium("PY.L","PX=1") = PY.L;
equilibrium("PU.L","PX=1") = PU.L;
equilibrium("PL.L","PX=1") = PL.L;
equilibrium("PK.L","PX=1") = PK.L;

equilibrium("RA.L","PX=1") = RA.L;

equilibrium("PX.L/PX.L","PX=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PX=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PX=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PX=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PX=1") = PK.L/PX.L;

equilibrium("RA.L/PX.L","PX=1") = RA.L/PX.L;

*   Recalculate with a different numeraire.
*   "Unfix" the price of X and fix the wage rate:

PX.UP = +inf;
PX.LO = 1e-5;
PL.FX = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

equilibrium("X.L","PL=1") = X.L;
equilibrium("Y.L","PL=1") = Y.L;
equilibrium("U.L","PL=1") = U.L;

equilibrium("PX.L","PL=1") = PX.L;
equilibrium("PY.L","PL=1") = PY.L;
equilibrium("PU.L","PL=1") = PU.L;
equilibrium("PL.L","PL=1") = PL.L;
equilibrium("PK.L","PL=1") = PK.L;

equilibrium("RA.L","PL=1") = RA.L;

equilibrium("PX.L/PX.L","PL=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PL=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PL=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PL=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PL=1") = PK.L/PX.L;

equilibrium("RA.L/PX.L","PL=1") = RA.L/PX.L;

display equilibrium;
```

Model TWOBYTWO: Algebraic Version in GAMS MCP

```
$title  A two by two general equilibrium model -- scalar GAMS/MCP

parameter   endow     index of labour endowment     / 1.0 /;

*   =====================================================
*   Variables which appear explicitly in the MPSGE model:

Nonnegative Variables
    X   Activity level for sector X -- benchmark=1
    Y   Activity level for sector Y -- benchmark=1
    U   Activity level for sector U -- benchmark=1

    PU  Relative price index for commodity U -- benchmark=1,
    PX  Relative price index for commodity X -- benchmark=1,
    PY  Relative price index for commodity Y -- benchmark=1
    PL  Relative price index for labor -- benchmark=1
    PK  Relative price index for capital -- benchmark=1;

Free Variable
    RA  Income level for representative agent -- benchmark=150;

*   Assign default prices and activity levels:

X.L = 1; Y.L = 1; U.L = 1; PX.L = 1; PY.L = 1; PK.L = 1; PU.L = 1; RA.L = 150;

*   Insert lower bounds to avoid bad function calls:

PX.LO = 0.001; PY.LO = 0.001; PU.LO = 0.001; PL.LO = 0.001; PK.LO = 0.001;

*   =====================================================
*   Variables that enter the MPSGE model implicitly:

variables
   LX   'compensated labor demand in sector x'
   LY   'compensated labor demand in sector y'
   KX   'compensated capital demand in sector x'
   KY   'compensated capital demand in sector y'
   DX   'compensated demand for x in sector u'
   DY   'compensated demand for y in sector u';


*   Equations for the implicit variables:

Equations

   lxdef    'compensated labor demand in sector x'
   lydef    'compensated labor demand in sector y'
   kxdef    'compensated capital demand in sector x'
   kydef    'compensated capital demand in sector y'
   dxdef    'compensated demand for x in sector u'
   dydef    'compensated demand for y in sector u';


lxdef..     LX =e= 50 * (PL**0.5 * PK**0.5)/PL;
lydef..     LY =e= 20 * (PL**0.4 * PK**0.6)/PL;
kxdef..     KX =e= 50 * (PL**0.5 * PK**0.5)/PK;
```

```
kydef..     KY =e= 30 * (PL**0.4 * PK**0.6)/PK;
dxdef..     DX =e= 100 * (PX**(2/3) * PY**(1/3))/PX;
dydef..     DY =e= 50  * (PX**(2/3) * PY**(1/3))/PY;

*   Initial values:

LX.L = 50; LY.L = 20; KX.L = 50; KY.L = 30; DX.L = 100; DY.L = 50;

*   ===================================================

Equations
   prf_x    'zero profit for sector x'
   prf_y    'zero profit for sector y'
   prf_u    'zero profit for sector u (Hicksian welfare index)'

   mkt_x    'supply-demand balance for commodity x'
   mkt_y    'supply-demand balance for commodity y'
   mkt_l    'supply-demand balance for primary factor l'
   mkt_k    'supply-demand balance for primary factor k'
   mkt_u    'supply-demand balance for aggregate demand'

   i_ra     'income definition for consumer (ra)';

*   Zero profit:

prf_x..     PL*LX + PK*KX  =e= 100 * PX;
prf_y..     PL*LY + PK*KY  =E= 50 * PY;
prf_u..     PX*DX + PY*DY =E= 150*PU;

*   Market clearance:

mkt_x..         100 * X =e= DX*U;
mkt_y..         50 * Y =e= DY*U;
mkt_u..         150 * U =E= RA / PU;
mkt_l..         70 * endow =e= LX*X + LY*Y;
mkt_k..         80 =e= KX*X + KY*Y;

*   Income balance:

i_ra..      RA =e= (70*endow)*PL + 80*PK;

* We declare the model using the mixed complementarity syntax
* in which equation identifiers are associated with variables.

model algebraic / prf_x.X, prf_y.Y, prf_u.U, mkt_x.PX, mkt_y.PY, mkt_l.PL,
                  mkt_k.PK, mkt_u.PU, I_ra.RA,
         lxdef.LX, lydef.LY, kxdef.KX, kydef.KY, dxdef.DX, dydef.DY /;

* Use sector x as the numeraire commodtity

PX.FX = PX.L;

algebraic.iterlim = 0;
solve algebraic using MCP;
algebraic.iterlim = 1000;

*   Solve the same counterfactual:
```

```
endow = 1.1;

*   Fix the income level at the default level, i.e. the
*   income level corresponding to the counterfactual
*   endowment at benchmark price:

RA.FX = 80 + 1.1 * 70;

solve algebraic using MCP;

parameter   equilibrium Equilibrium values;

*   Save counterfactual values:

equilibrium("X.L","RA=157") = X.L;
equilibrium("Y.L","RA=157") = Y.L;
equilibrium("U.L","RA=157") = U.L;

equilibrium("PX.L","RA=157") = PX.L;
equilibrium("PY.L","RA=157") = PY.L;
equilibrium("PU.L","RA=157") = PU.L;
equilibrium("PL.L","RA=157") = PL.L;
equilibrium("PK.L","RA=157") = PK.L;

equilibrium("RA.L","RA=157") = RA.L;
equilibrium("PX.L/PX.L","RA=157") = PX.L/PX.L;
equilibrium("PY.L/PX.L","RA=157") = PY.L/PX.L;
equilibrium("PU.L/PX.L","RA=157") = PU.L/PX.L;
equilibrium("PL.L/PX.L","RA=157") = PL.L/PX.L;
equilibrium("PK.L/PX.L","RA=157") = PK.L/PX.L;
equilibrium("RA.L/PX.L","RA=157") = RA.L/PX.L;

*   Fix a numeraire price index and recalculate:

RA.LO = -inf;
RA.UP = inf;
PX.FX = 1;

solve algebraic using mcp;

equilibrium("X.L","PX=1") = X.L;
equilibrium("Y.L","PX=1") = Y.L;
equilibrium("U.L","PX=1") = U.L;

equilibrium("PX.L","PX=1") = PX.L;
equilibrium("PY.L","PX=1") = PY.L;
equilibrium("PU.L","PX=1") = PU.L;
equilibrium("PL.L","PX=1") = PL.L;
equilibrium("PK.L","PX=1") = PK.L;

equilibrium("RA.L","PX=1") = RA.L;

equilibrium("PX.L/PX.L","PX=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PX=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PX=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PX=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PX=1") = PK.L/PX.L;
```

```
equilibrium("RA.L/PX.L","PX=1") = RA.L/PX.L;

*   Recalculate with a different numeraire.
*   "Unfix" the price of X and fix the wage rate:

PX.UP = +inf;
PX.LO = 1e-5;
PL.FX = 1;
solve algebraic using mcp;

equilibrium("X.L","PL=1") = X.L;
equilibrium("Y.L","PL=1") = Y.L;
equilibrium("U.L","PL=1") = U.L;

equilibrium("PX.L","PL=1") = PX.L;
equilibrium("PY.L","PL=1") = PY.L;
equilibrium("PU.L","PL=1") = PU.L;
equilibrium("PL.L","PL=1") = PL.L;
equilibrium("PK.L","PL=1") = PK.L;

equilibrium("RA.L","PL=1") = RA.L;

equilibrium("PX.L/PX.L","PL=1") = PX.L/PX.L;
equilibrium("PY.L/PX.L","PL=1") = PY.L/PX.L;
equilibrium("PU.L/PX.L","PL=1") = PU.L/PX.L;
equilibrium("PL.L/PX.L","PL=1") = PL.L/PX.L;
equilibrium("PK.L/PX.L","PL=1") = PK.L/PX.L;

equilibrium("RA.L/PX.L","PL=1") = RA.L/PX.L;

display equilibrium;
```

Indexed MPSGE Model TWOBYTWO

```
$title  A two by two general equilibrium model -- indexed GAMS/MPSGE

Sets
    i   Produced goods          / x, y /,
    f   Factors of production   / L, K /;

table  sam(*,*)    Benchmark input-output matrix

                X       Y        U    RA
        X     100              -100
        Y              50       -50
        U                       150  -150
        K     -50     -20        70
        L     -50     -30        80;

parameters
    supply(i)   Benchmark supply of output of sectors,
    factor(f,i) Benchmark factor demand,
    demand(i)   Benchmark demand for consumption,
    endow(f)    Factor endowment,
    cons        Benchmark total consumption;

* Extract data from the original format into model-specific arrays
```

```
supply(i)   = sam(i,i);
factor(f,i) = -sam(f,i);
demand(i)   = -sam(i,'u');
cons        = sum(i, demand(i));
endow(f)    = sam(f,'ra');

display supply, factor, demand, cons, endow;

$ontext
$MODEL:twobytwo

$SECTORS:
    Y(i)    ! Activity level --  benchmark=1
    U       ! Final consumption index -- benchmark=1

$COMMODITIES:
    PU      ! Relative price of final consumption -- benchmkark=1
    PC(i)   ! Relative price of commodities -- benchmark=1
    PF(f)   ! Relative price of factors -- benchmark=1

$CONSUMERS:
    RA      ! Income level (benchmark=150)

$PROD:Y(i) s:1
        O:PC(i)     Q:supply(i)
        I:PF(f)     Q:factor(f,i)

$PROD:U s:1
        O:PU        Q:cons
        I:PC(i)     Q:demand(i)

$DEMAND:RA
        D:PU        Q:cons
        E:PF(f)     Q:endow(f)

$offtext
$sysinclude mpsgeset twobytwo

* Benchmark replication

twobytwo.iterlim = 0;
$include TWOBYTWO.GEN
solve twobytwo using mcp;
twobytwo.iterlim = 1000;

* Counterfactual : 10% increase in labor endowment

endow('l') = 1.1*endow('l');

*   Solve the model with the default normalization of prices which
*   fixes the income level of the representative agent.  The RA
*   income level at the initial prices equals 80 + 1.1*70 = 157.

$include TWOBYTWO.GEN
solve twobytwo using mcp;

parameter   equilibrium Equilibrium values;
```

```
*   Save counterfactual values:

equilibrium("Y.L",i,"RA=157") = Y.L(i);
equilibrium("U.L","_","RA=157") = U.L;
equilibrium("PC.L",i,"RA=157") = PC.L(i);
equilibrium("PF.L",f,"RA=157") = PF.L(f);
equilibrium("RA.L","_","RA=157") = RA.L;
equilibrium('PC(i)/PC("x")',i,"RA=157") = PC.L(i)/PC.L("x");
equilibrium('PF(f)/PC("x")',f,"RA=157") = PF.L(f)/PC.L("x");
equilibrium('RA.L/PC("x")',"_","RA=157") = RA.L/PC.L("x");

*   Fix a numeraire price index and recalculate:

PC.FX("x") = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

equilibrium("Y.L",i,'PC("x")=1') = Y.L(i);
equilibrium("U.L","_",'PC("x")=1') = U.L;
equilibrium("PC.L",i,'PC("x")=1') = PC.L(i);
equilibrium("PF.L",f,'PC("x")=1') = PF.L(f);
equilibrium('PC(i)/PC("x")',i,'PC("x")=1') = PC.L(i)/PC.L("x");
equilibrium('PF(f)/PC("x")',f,'PC("x")=1') = PF.L(f)/PC.L("x");
equilibrium("RA.L","_",'PC("x")=1') = RA.L;
equilibrium('RA.L/PC("x")',"_",'PC("x")=1') = RA.L/PC.L("x");

*   Recalculate with a different numeraire.
*   "Unfix" the price of X and fix the wage rate:

PC.UP("X") = +inf;  PC.LO("X") = 1e-5;  PF.FX("L") = 1;
$include TWOBYTWO.GEN
solve twobytwo using mcp;

equilibrium("Y.L",i,'PF("L")=1') = Y.L(i);
equilibrium("U.L","_",'PF("L")=1') = U.L;
equilibrium("PC.L",i,'PF("L")=1') = PC.L(i);
equilibrium("PF.L",f,'PF("L")=1') = PF.L(f);
equilibrium('PC(i)/PC("x")',i,'PF("L")=1') = PC.L(i)/PC.L("x");
equilibrium('PF(f)/PC("x")',f,'PF("L")=1') = PF.L(f)/PC.L("x");
equilibrium("RA.L","_",'PF("L")=1') = RA.L;
equilibrium('RA.L/PC("x")',"_",'PF("L")=1') = RA.L/PC.L("x");

option equilibrium:3:2:1;
display equilibrium;
```

Three Versions of Model JPMGE

MPSGE Model JPMGE

```
$title   Model with Joint Products and Intermediate Demand -- solved with GAMS/MPSGE

Sets    j      Sectors    / s1*s2 /,
        i      Goods      / g1*g2 /,
        f      Primary factors  / labor, capital / ;
```

```
alias (i,ii),(j,jj);

Table   make0(i,j)  Matrix -- supplies
           s1      s2
   g1       6       2
   g2       2      10 ;

Table   use0(i,j)   Use matrix -- intermediate demands
           s1      s2
   g1       4       2
   g2       2       6 ;

Table  fd0(f,j)     Factor demands
           s1       s2
   labor    1        3
   capital  1        1 ;

Parameters
  c0(i)     Consumer demand  / g1 2, g2 4 /
  e0(f)     Factor endowments;

e0(f) = sum(j, fd0(f,j));
display e0;

$ontext
$MODEL:jpmge

$SECTORS:
    X(j)    ! Activity index -- benchmark=1

$COMMODITIES:
    P(i)    ! Relative commodity price -- benchmark=1
    PF(f)   ! Relative factor price -- benchmark=1

$CONSUMERS:
    Y       ! Nominal household income=expenditure

$PROD:X(j)   s:1    t:1
      O:P(i)    Q:make0(i,j)    ! S(i,j) in the MCP and NLP models
      I:P(i)    Q:use0(i,j)     ! D(i,j) in the MCP and NLP models
      I:PF(f)   Q:fd0(f,j)      ! FD(f,j) in the MCP and NLP models

$REPORT:
    v:S(i,j)    O:P(i)      PROD:X(j)
    v:D(i,j)    I:P(i)      PROD:X(j)
    v:FD(f,j)   I:PF(f)     PROD:X(j)

$DEMAND:Y  s:1
    D:P(i)      Q:c0(i)
    E:PF(f)     Q:e0(f)

$REPORT:
    v:C(i)      D:P(i)      DEMAND:Y

$offtext
$sysinclude mpsgeset jpmge

* Benchmark replication
```

```
jpmge.iterlim = 0;
$include JPMGE.GEN
solve jpmge using mcp;
abort$(abs(jpmge.objval) gt 1e-7) "JPMGE does not calibrate!";
jpmge.iterlim = 1000;

* Counterfactual : 10% increase in labor endowment

e0("labor") = 1.1 * e0("labor");

$include JPMGE.GEN
solve jpmge using mcp;

Parameter   equilibrium   Equilibrium values;

*   Save counterfactual values:

equilibrium("X",j,"Y=6.4")   = X.L(j);
equilibrium(i,j,"Y=6.4")     = S.L(i,j)-D.L(i,j);
equilibrium(f,j,"Y=6.4")     = FD.L(f,j);
equilibrium("C",i,"Y=6.4")   = C.L(i);
equilibrium("X",j,"Y=6.4")   = X.L(j);
equilibrium("P",i,"Y=6.4")   = P.L(i);
equilibrium("PF",f,"Y=6.4")  = PF.L(f);
equilibrium("Y","_","Y=6.4") = Y.L;

*   Fix a numeraire price index and recalculate:

P.FX("g1") = 1;
$include JPMGE.GEN
solve jpmge using mcp;

equilibrium("X",  j,'P("g1")=1') = X.L(j);
equilibrium(i,    j,'P("g1")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f,    j,'P("g1")=1') = FD.L(f,j);
equilibrium("C",  i,'P("g1")=1') = C.L(i);
equilibrium("X",  j,'P("g1")=1') = X.L(j);
equilibrium("P",  i,'P("g1")=1') = P.L(i);
equilibrium("PF", f,'P("g1")=1') = PF.L(f);
equilibrium("Y","_",'P("g1")=1') = Y.L;

*   Recalculate with a different numeraire.
*   "Unfix" the price of X and fix the wage rate:

P.UP("g1") = +inf;
P.LO("g1") = 1e-5;
PF.FX("labor") = 1;

$include JPMGE.GEN
solve jpmge using mcp;

equilibrium("X",  j,'PF("labor")=1')   = X.L(j);
equilibrium(i,      j,'PF("labor")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f,    j,'PF("labor")=1')   = FD.L(f,j);
equilibrium("C",  i,'PF("labor")=1')   = C.L(i);
equilibrium("X",  j,'PF("labor")=1')   = X.L(j);
equilibrium("P",  i,'PF("labor")=1')   = P.L(i);
```

```
equilibrium("PF", f,'PF("labor")=1')   = PF.L(f);
equilibrium("Y","_",'PF("labor")=1')   = Y.L;

option equilibrium:3:2:1;
display equilibrium;
```

Algebraic Version of Model JPMGE: MCP Formulation

```
$title  Model with Joint Products and Intermediate Demand -- solved with GAMS/MCP

Sets    j      Sectors    / s1*s2 /,
        i      Goods      / g1*g2 /,
        f      Primary factors  / labor, capital / ;

alias (i,ii),(j,jj);

Table   make0(i,j)  Matrix  -- supplies
           s1      s2
   g1       6       2
   g2       2      10 ;

Table   use0(i,j)   Use matrix -- intermediate demands
           s1      s2
   g1       4       2
   g2       2       6 ;

Table  fd0(f,j)     Factor demands
           s1       s2
   labor    1        3
   capital  1        1 ;

Parameters
  c0(i)     Consumer demand  / g1 2, g2 4 /
  e0(f)     Factor endowments;

e0(f) = sum(j, fd0(f,j));
display e0;

*   =====================================================
*   Variables which appear explicitly in the MPSGE model:

Variables
    X(j)    ! Activity index -- benchmark=1
    P(i)    ! Relative commodity price -- benchmark=1
    PF(f)   ! Relative factor price -- benchmark=1
    Y       ! Nominal household income=expenditure;

X.L(j) = 1;  P.L(i) = 1; PF.L(f) = 1; Y.L = sum(f,e0(f));
P.LO(i) = 1e-4; PF.LO(f) = 1e-4;

*   =====================================================
*   Variables that enter the MPSGE model implicitly:

Variables
    S(i,j)      Compensated supply
    D(i,j)      Compensated intermediate demand
    FD(f,j)     Compensated factor demand
```

```
    C(i)        Final demand;

S.L(i,j) = make0(i,j);
D.L(i,j) = use0(i,j);
FD.L(f,j) = fd0(f,j);
C.L(i) = c0(i);

*   =====================================================
*   Calibration calculations provided automatically by MPSGE:

Parameter   thetad(i,j)  Intermediate demand value share
            thetas(i,j)  Output value share
            thetaf(f,j)  Factor demand value share
            thetac(i)    Final demand value share;

thetas(i,j)  = make0(i,j)/sum(ii,make0(ii,j));
thetad(i,j)  = use0(i,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetaf(f,j)  = fd0(f,j) /(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));

thetac(i) = c0(i)/sum(ii,c0(ii));

alias (i,i_), (f,f_);

*   Equations for the implicit variables:

Equations   sdef, ddef, fddef, cdef;

$macro REV(j)   (sqrt(sum(i_,thetas(i_,j)*sqr(P(i_)))))

sdef(i,j)..    S(i,j) =e= make0(i,j)*P(i)/REV(j);

$macro COST(j)  (prod(i_,P(i_)**thetad(i_,j))*prod(f_,PF(f_)**thetaf(f_,j)))

ddef(i,j)..    D(i,j) =e= use0(i,j) * COST(j)/P(i);

fddef(f,j)..   FD(f,j) =e= fd0(f,j) * COST(j)/PF(f);

cdef(i)..      C(i) =e= thetac(i) * Y/P(i);

*   =====================================================
*   Equilibrium conditions:

Equations   prf_X(j), mkt_p(i), mkt_pf(f), income;

*   Zero profit:

prf_X(j)..    sum(f,PF(f)*FD(f,j)) + sum(i,P(i)*D(i,j)) =e= sum(i,P(i)*S(i,j));

*   Market clearance:

mkt_P(i)..    sum(j, X(j)*(S(i,j)-D(i,j))) =e= C(i);

mkt_PF(f)..   e0(f) =e= sum(j, FD(f,j)*X(j));

*   Income balance:

income..      Y =e= sum(f, PF(f)*e0(f));
```

```
model jpmcp / sdef.S, ddef.D, fddef.FD, cdef.C, prf_X.X, mkt_P.P, mkt_PF.PF, income.Y /;

*   ====================================================

*   Benchmark replication with iteration limit zero.  Do not need
*   to fix the price level at this point:

jpmcp.iterlim = 0;
solve jpmcp using mcp;
abort$(abs(jpmcp.objval) gt 1e-7) "JPMCP does not calibrate!";
jpmcp.iterlim = 1000;

*   ====================================================
* Counterfactual : 10% increase in labor endowment

e0("labor") = 1.1 * e0("labor");

*   Fix the income level at the default level, i.e. the
*   income level corresponding to the counterfactual
*   endowment at benchmark price:

Y.FX = sum(f,e0(f));

solve jpmcp using mcp;

Parameter   equilibrium Equilibrium values;

*   Save counterfactual values:

equilibrium("X",j,"Y=6.4")   = X.L(j);
equilibrium(i,j,"Y=6.4")     = S.L(i,j)-D.L(i,j);
equilibrium(f,j,"Y=6.4")     = FD.L(f,j);
equilibrium("C",i,"Y=6.4")   = C.L(i);
equilibrium("X",j,"Y=6.4")   = X.L(j);
equilibrium("P",i,"Y=6.4")   = P.L(i);
equilibrium("PF",f,"Y=6.4")  = PF.L(f);
equilibrium("Y","_","Y=6.4") = Y.L;

*   Fix a numeraire price index and recalculate:

P.FX("g1") = 1;
Y.LO = -INF;
Y.UP =  INF;

solve jpmcp using mcp;

equilibrium("X",  j,'P("g1")=1') = X.L(j);
equilibrium(i,    j,'P("g1")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f,    j,'P("g1")=1') = FD.L(f,j);
equilibrium("C",  i,'P("g1")=1') = C.L(i);
equilibrium("X",  j,'P("g1")=1') = X.L(j);
equilibrium("P",  i,'P("g1")=1') = P.L(i);
equilibrium("PF", f,'P("g1")=1') = PF.L(f);
equilibrium("Y","_",'P("g1")=1') = Y.L;

*   Recalculate with a different numeraire.
*   "Unfix" the price of X and fix the wage rate:
```

```
P.UP("g1")     = +inf;
P.LO("g1")     = 1e-5;
PF.FX("labor") = 1;

solve jpmcp using mcp;

equilibrium("X",  j,'PF("labor")=1')   = X.L(j);
equilibrium(i,      j,'PF("labor")=1') = S.L(i,j)-D.L(i,j);
equilibrium(f,    j,'PF("labor")=1')   = FD.L(f,j);
equilibrium("C",  i,'PF("labor")=1')   = C.L(i);
equilibrium("X",  j,'PF("labor")=1')   = X.L(j);
equilibrium("P",  i,'PF("labor")=1')   = P.L(i);
equilibrium("PF", f,'PF("labor")=1')   = PF.L(f);
equilibrium("Y","_",'PF("labor")=1')   = Y.L;

option equilibrium:3:2:1;
display equilibrium;
```

Algebraic Version of Model JPMGE: NLP Formulation

```
$title  Model with Joint Products and Intermediate Demand -- solved as NLP

Sets    j      Sectors    / s1*s2 /,
        i      Goods      / g1*g2 /,
        f      Primary factors  / labor, capital / ;

alias (i,ii),(j,jj);

Table   make0(i,j)  Matrix -- supplies
           s1      s2
   g1       6       2
   g2       2      10 ;

Table   use0(i,j)   Use matrix -- intermediate demands
           s1      s2
   g1       4       2
   g2       2       6 ;

Table  fd0(f,j)     Factor demands
           s1       s2
   labor    1        3
   capital  1        1 ;

Parameters
  c0(i)     Consumer demand  / g1 2, g2 4 /
  e0(f)     Factor endowments;

e0(f) = sum(j, fd0(f,j));
display e0;


*   =====================================================
*   Variables which appear explicitly in the MPSGE model:

Nonnegative
Variable    X(j)    ! Activity index -- benchmark=1;
X.L(j) = 1;
```

```
*   =====================================================
*   Variables that enter the MPSGE model implicitly:

Variables
    U           Utility
    S(i,j)      Compensated supply
    D(i,j)      Compensated intermediate demand
    FD(f,j)     Compensated factor demand
    C(i)        Final demand;

S.L(i,j)  = make0(i,j);
D.L(i,j)  = use0(i,j);
FD.L(f,j) = fd0(f,j);
C.L(i)    = c0(i);

*   =====================================================
*   Calibration calculations provided automatically by MPSGE:

Parameter   thetad(i,j) Intermediate demand value share
            thetas(i,j) Output value share
            thetaf(f,j) Factor demand value share
            thetac(i)   Final demand value share;

thetas(i,j)  = make0(i,j)/sum(ii,make0(ii,j));
thetad(i,j)  = use0(i,j)/(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetaf(f,j)  = fd0(f,j) /(sum(ii,use0(ii,j))+sum(ff,fd0(ff,j)));
thetac(i)    = c0(i)/sum(ii,c0(ii));

*   =====================================================
*   Equilibrium conditions:

Equations   production, goods, factors, utility;

*   Zero profit:

production(j)..  prod(i, (D(i,j)/use0(i,j))**thetad(i,j)) *
        prod(f, (FD(f,j)/fd0(f,j))**thetaf(f,j)) =e=
        sqrt(sum(i, thetas(i,j)*sqr(S(i,j)/make0(i,j))));

*   Market clearance:

goods(i)..      sum(j, X(j)*(S(i,j)-D(i,j))) =e= C(i);

factors(f)..    e0(f) =e= sum(j, FD(f,j)*X(j));

*   Income balance:

utility..       U =E= prod(i, (C(i)/c0(i))**thetac(i));

model jpnlp / production, goods, factors, utility/;

*   =====================================================
*   Benchmark replication with iteration limit zero.  Do not need
*   to fix the price level at this point:

solve jpnlp using nlp maximizing U;
```

```
Parameter   equilibrium    Equilibrium values
            pnum           Numeraire price index;

*   Save benchmark values:

pnum = sum(f,factors.m(f)*e0(f))/sum(f,e0(f));
equilibrium("X",j,"bmk")   = X.L(j);
equilibrium(i,j,"bmk")     = X.L(j)*(S.L(i,j)-D.L(i,j));
equilibrium(f,j,"bmk")     = FD.L(f,j);
equilibrium("C",i,"bmk")   = C.L(i);
equilibrium("X",j,"bmk")   = X.L(j);
equilibrium("P",i,"bmk")   = goods.m(i)/pnum;
equilibrium("PF",f,"bmk")  = factors.m(f)/pnum;
equilibrium("Y","_","bmk") = sum(f,factors.m(f)*e0(f)/pnum);


*   =====================================================
* Counterfactual : 10% increase in labor endowment

e0("labor") = 1.1 * e0("labor");

solve jpnlp using nlp maximizing u;

*   Save counterfactual values:

pnum = sum(f,factors.m(f)*e0(f))/sum(f,e0(f));
equilibrium("X",j,'labor+10%')   = X.L(j);
equilibrium(i,j,'labor+10%')     = X.L(j)*(S.L(i,j)-D.L(i,j));
equilibrium(f,j,'labor+10%')     = FD.L(f,j);
equilibrium("C",i,'labor+10%')   = C.L(i);
equilibrium("X",j,'labor+10%')   = X.L(j);
equilibrium("P",i,'labor+10%')   = goods.m(i)/pnum;
equilibrium("PF",f,'labor+10%')  = factors.m(f)/pnum;
equilibrium("Y","_",'labor+10%') = sum(f,factors.m(f)*e0(f)/pnum);

option equilibrium:3:2:1;
display equilibrium;
```

Three Versions of a 123 Model

Data for Models: 123DATA

This data file is included in each of the following models.

```
$stitle Dataset for a 123 Model

set     mcmrow  Rows in the micro-consistent matrix /
                PFX     Current account,
                PD      Domestic ouputput
                TA      Sales and excise taxes
                TM      Import tariffs
                TX      Export taxes
                TK      Capital taxes
                TL      Labor taxes
                RK      Return to capital
```

```
                PL      Wage rate
                PA      Price of Armington composite /,

        mcmcol  Columns in the micro-consistent matrix /
                S       Supply,
                D       Demand,
                GOVT    Government,
                HH      Households
                INVEST  Investment /;

table mcm(mcmrow,mcmcol)  Microconsistent matrix

              S           D        GOVT          HH      INVEST

PFX     106.386    -144.701      38.315
PD      218.308    -218.308
TA                  -32.027      32.027
TM                  -18.617      18.617
TX       -1.136                   1.136
TK      -12.837                  12.837
TL       -3.539                   3.539
RK     -143.862                           143.862
PL     -163.320                           163.320
PA                  413.653     -35.583    -291.694     -86.376

*       Parameter values describing base year equilibrium:

parameter       px0     Reference price of exports
                d0      Reference domestic supply
                x0      Reference exports
                kd0     Reference net capital earnings
                ly0     Reference net labor earnings
                rr0     Reference price of capital
                pl0     Reference wage
                tk      Capital tax rate
                tl      Labor tax rate
                ta      Excise and sales tax rate
                tx      Tax on exports
                a0      Aggregate supply (gross of tax)
                g0      Government demand,
                dtax    Direct tax net transfers
                m0      Imports
                l0      Leisure demand
                c0      Household consumption,
                i0      Aggregate investment
                tm      Import tariff rate
                pm0     Reference price of imports
                pwm     World price of imports /1/
                pwx     World price of exports /1/
                bopdef  Balance of payments deficit
                etadx   Elasticity of transformation (D versus X) /4/,
                sigmadm Elasticity of substitution (D versus M) /4/,
                esubkl  Elasticity of substitution (K versus L) /1/,
                sigma   Elasticity of substitution (C versus LS) /0.4/;

d0  = mcm("pd","s");
x0  = mcm("pfx","s");
kd0 = -mcm("rk","s");
```

```
ly0 = -mcm("pl","s");

tx  = -mcm("tx","s")/mcm("pfx","s");
tk  = mcm("tk","s")/mcm("rk","s");
tl  = mcm("tl","s")/mcm("pl","s");

px0 = 1 - tx;
rr0 = 1 + tk;
pl0 = 1 + tl;

parameter       profit  Zero profit check;
profit("PD") = d0;
profit("PX") = x0;
profit("TX") = -tx*x0;
profit("TK") = -tk*kd0;
profit("TL") = -tl*ly0;
profit("PL") = -ly0;
profit("RK") = -kd0;
alias (u,*);
profit("CHK") = sum(u, profit(u));
display profit, tx, tk, tl;

m0 = -mcm("pfx","d");
tm = mcm("tm","d")/mcm("pfx","d");
pm0 = 1 + tm;
a0 = mcm("pa","d");
g0 = -mcm("pa","govt");
ta = -mcm("ta","d")/mcm("pa","d");
bopdef = mcm("pfx","govt");
dtax = g0 - bopdef - tm*m0 - ta*a0 - tl*ly0 - tk*kd0 - tx*x0;

i0 = -mcm("pa","invest");
c0 = a0 - i0 - g0;
l0 = 0.75*ly0;
display g0;
```

MGE123: MPSGE Model

This is the base 123 MPSGE model.

```
$title  Static 123 Model Ala Devarjan

$include 123data.gms

$ontext
$model:MGE123

$SECTORS:
        Y       ! Production
        A       ! Armington composite
        M       ! Imports
        X       ! Exports

$COMMODITIES:
        PD      ! Domestic price index
        PX      ! Export price index
```

```
        PM      ! Import price index
        PA      ! Armington price index
        PL      ! Wage rate index
        RK      ! Rental price index
        PFX     ! Foreign exchange

$CONSUMERS:
        HH      ! Private households
        GOVT    ! Government

$AUXILIARY:
        TAU_LS  ! Lumpsum Replacement tax
        TAU_TL  ! Labor tax replacement
        UR      ! Unemployment rate

$PROD:Y  t:etadx s:esubkl
        O:PD    Q:d0    P:1                             ! YD
        O:PX    Q:x0    P:px0   A:GOVT  T:tx            ! YX
        I:RK    Q:kd0   P:rr0   A:GOVT  T:tk            ! KD
        I:PL    Q:ly0   P:pl0   A:GOVT  T:tl N:TAU_TL   ! LY

$report:
         v:YD   o:PD    prod:Y
         v:YX   o:PX    prod:Y
         v:KD   i:RK    prod:Y
         v:LY   i:PL    prod:Y

$PROD:A  s:sigmadm
        O:PA    Q:a0    A:GOVT  t:ta
        I:PD    Q:d0                        ! DA
        I:PM    Q:m0    p:pm0 A:GOVT t:tm   ! MA

$report:
    v:DA    i:PD    prod:A
    v:MA    i:PM    prod:A

$PROD:M
        O:PM    Q:m0
        I:PFX   Q:(pwm*m0)

$PROD:X
        O:PFX   Q:(pwx*x0)
        I:PX    Q:x0

$DEMAND:GOVT
        E:PFX   Q:bopdef
        E:PA    Q:dtax
        E:PA    Q:g0            R:TAU_LS
        D:PA

$CONSTRAINT:UR
        PL =G= PA;

$CONSTRAINT:TAU_LS
        GOVT =e= PA * g0;

$CONSTRAINT:TAU_TL
        GOVT =e= PA * g0;
```

```
$DEMAND:HH  s:sigma
        E:PA    Q:(-g0)         R:TAU_LS
        E:PA    Q:(-dtax)
        E:RK    Q:kd0
        E:PA    Q:(-i0)
        E:PL    Q:(ly0+l0)      ! Labor endowment = ly0+l0 - UR * (ly0+l0)
        E:PL    Q:(-(ly0+l0))   R:UR
        D:PA    Q:c0
        D:PL    Q:l0

$report:
    v:W     w:HH
    v:C     d:PA    demand:HH
    v:LD    d:PL    demand:HH

$offtext
$sysinclude mpsgeset mge123

UR.FX = 0;
TAU_TL.FX = 0;
TAU_LS.UP =  INF;
TAU_LS.LO = -INF;
mge123.iterlim = 0;
$include MGE123.GEN
solve mge123 using mcp;
abort$(mge123.objval > 1e-4) "Benchmark model does not calibrate.";
mge123.iterlim = 10000;

mge123.savepoint = 2;

Parameter   report  Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mge123.objval > 1e-4) "Scenario fails to solve.";

report("W","%replacement%","%labormarket%") = 100*(W.L-1);
report("Y","%replacement%","%labormarket%") = 100*(Y.L-1);
report("A","%replacement%","%labormarket%") = 100 * (A.L-1);
report("M","%replacement%","%labormarket%") = 100 * (M.L-1);
report("X","%replacement%","%labormarket%") = 100 * (X.L-1);
report("YD","%replacement%","%labormarket%") = 100 * (YD.L/d0-1);
report("YX","%replacement%","%labormarket%") = 100 * (YX.L/x0-1);
report("KD","%replacement%","%labormarket%") = 100 * (KD.L/kd0-1);
report("LY","%replacement%","%labormarket%") = 100 * (LY.L/ly0-1);
report("DA","%replacement%","%labormarket%") = 100 * (DA.L/d0-1);
report("MA","%replacement%","%labormarket%") = 100 * (MA.L/m0-1);
report("C","%replacement%","%labormarket%") = 100 * (C.L/c0-1);
report("LD","%replacement%","%labormarket%") = 100 * (LD.L/l0-1);
report("PD","%replacement%","%labormarket%") = 100 * (PD.L/PL.L - 1);
report("PX","%replacement%","%labormarket%") = 100 * (PX.L/PL.L - 1);
report("PM","%replacement%","%labormarket%") = 100 * (PM.L/PL.L - 1);
report("PA","%replacement%","%labormarket%") = 100 * (PA.L/PL.L - 1);
report("PL","%replacement%","%labormarket%") = 100 * (PL.L/PL.L - 1);
report("RK","%replacement%","%labormarket%") = 100 * (RK.L/PL.L - 1);
report("PFX","%replacement%","%labormarket%") = 100 * (PFX.L/PL.L - 1);
report("HH","%replacement%","%labormarket%") = 100 * (HH.L/PL.L - 1);
report("GOVT","%replacement%","%labormarket%") = 100 * (GOVT.L/PL.L - 1);
```

```
report("TAU_LS","%replacement%","%labormarket%") = 100*TAU_LS.L;
report("TAU_TL","%replacement%","%labormarket%") = 100*TAU_TL.L;
report("UR","%replacement%","%labormarket%") = 100*UR.L;
$offecho

*   Tariff reform:

tm = 0;

UR.FX = 0;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Lump Sum
$set labormarket Flexible
$include %gams.scrdir%report

UR.FX = 0;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Flexible
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Lump Sum
$set labormarket Rigid Wage
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Rigid Wage
$include %gams.scrdir%report

option report:1:1:2;
display report;
```

Model MCP123: Algebraic Version in GAMS MCP

This GAMS model is the model `mge123` translated verbatim into GAMS/MCP (algebraic) format. This is the way the model will look in the MPSGEv2 framework.

```
$title  Static 123 Model Ala Devarjan -- GAMS/MCP Format

$include 123data

*   ====================================================
*   Variables which appear explicitly in the MPSGE model:

Nonnegative Variables

*$SECTORS:
        Y        Production
        A        Armington composite
        M        Imports
        X        Exports

*$COMMODITIES:
        PD       Domestic price index
        PX       Export price index
        PM       Import price index
        PA       Armington price index
        PL       Wage rate index
        RK       Rental price index
        PFX      Foreign exchange

*$CONSUMERS:
        HH       Private households
        GOVT     Government

*$AUXILIARY:
        TAU_TL   Wage replacement tax,
        TAU_LS   Lump sum replacement tax
        UR       Unemployment rate;

*   Assign default prices and activity levels:

Y.L = 1; A.L = 1; M.L = 1; X.L = 1;
PD.L = 1; PX.L = 1; PM.L = 1; PA.L = 1; PL.L = 1; RK.L = 1; PFX.L = 1;
HH.L = c0+l0; GOVT.L=g0;
TAU_TL.L = 0; TAU_LS.L = 0; UR.L = 0;

*   Insert lower bounds to avoid bad function calls:

PD.LO = 1e-4; PX.LO = 1e-4; PM.LO = 1e-4; PA.LO = 1e-4; PL.LO = 1e-4; RK.LO = 1e-4; PFX.LO = 1e-4;

*   ====================================================
*   Variables enter the MPSGE model implicitly:

Variable
    YD  Production for the domestic market,
    YX  Production for the export market,
    KD  Capital demand,
    LY  Labor demand,
```

```
    DA  Domestic absorption,
    MA  Imports,
    C   Consumption of goods (uncompensated),
    LD  Leisure demand (uncompensated);

*   Equations for the implicit variables:

Equations YDdef, YXdef, KDdef, LYdef, DAdef, MAdef, Cdef, LDdef;

*   Macros defining composite prices (unit cost and unit revenue):

Parameter thetal  Labor share in cost function,
          thetac  Consumption share in expenditure function,
          thetam  Share parameter in Armington function
          thetaz  Share parameter in transformation function  ;

thetal = ly0*pl0 /(kd0*rr0+ly0*pl0);
thetaz = x0 *px0 /(d0+x0*px0);
thetam = m0 *pm0 /(d0+m0*pm0);
thetac = c0/(c0+l0);


$macro CY   ((PL*(1+tl+TAU_TL)/pl0)**thetal * (RK*(1+tk)/rr0)**(1-thetal))
$macro RY   ((thetaz * (PX*(1-tx)/px0)**(1+etadx) + (1-thetaz) * PD**(1+etadx)) **(1/(1+etadx)))
$macro CA   ((thetam *(PM*(1+tm)/pm0)**(1-sigmadm) + (1-thetam)*PD**(1-sigmadm))**(1/(1-sigmadm)))
$macro CU   ((thetac*PA**(1-sigma) + (1-thetac)*PL**(1-sigma))**(1/(1-sigma)))
$macro W    (HH.L/((c0+l0)*CU))

*   Definitions of demand and supply functions:

YDdef..     YD =e= d0  * (PD/RY)**etadx;
YXdef..     YX =e= x0  * (PX*(1-tx)/(px0*RY))**etadx;
KDdef..     KD =e= kd0 * (CY*rr0/(RK*(1+tk)))**esubkl;
LYdef..     LY =e= ly0 * (CY*pl0/(PL*(1+tl+TAU_TL)))**esubkl;

DAdef..     DA =e= d0 * (CA/PD)**sigmadm;
MAdef..     MA =e= m0 * (CA*pm0/(PM*(1+tm)))**sigmadm;

Cdef..      C  =e= c0 * W * (CU/PA)**sigma;
LDdef..     LD =e= l0 * W * (CU/PL)**sigma;

*   Initialize:

YD.L = d0; YX.L = x0; KD.L = kd0; LY.L = ly0; DA.L = d0; MA.L = m0;
C.L = c0; LD.L = l0;

*   ====================================================

Equations

* Zero profit condition
         profity         domestic production,
         profita         Armington supply,
         profitm         imported goods production
         profitx         exported goods production

* Market clearing condition
         marketd         domestic  goods market,
```

```
         marketa         Armington goods market
         marketm         imported  goods market
         marketx         exported  goods market
         marketfx        balance of payment
         marketk         capital market
         marketl         labor   market

* Income balance
         incomeg budget
         incomeh         household budget

* Additional constraints
         tauTLdef   Equal yield constraint (TL),
         tauLSdef   Equal yield constraint (LS),
     URdef      Lower bound on the real wage;

marketd..  Y*YD =e= A*DA;

profity..  KD*RK*(1+tk) + LY*PL*(1+tl+TAU_TL) =E= YD*PD + YX*PX*(1-tx);

marketa..  A*a0 =g= GOVT/PA + i0 + C;

profita..  PD*DA + PM*(1+tm)*MA =e= PA*a0*(1-ta);

marketm..  M*m0 =e= A*MA;

profitm..  PFX*pwm =e= PM;

marketx..  Y*YX =e= X*x0;

profitx..  PX =e= PFX*pwx;

marketfx.. X*pwx*x0 + bopdef =E= M*pwm*m0;

marketk..  kd0    =e= Y*KD;

marketl..  ly0+l0 =e= Y*LY + LD + (ly0+l0)*UR;

incomeg..  GOVT =e= PFX*bopdef + PA*dtax +  PA*g0*TAU_LS + tx*PX*YX*Y +
            tk*RK*KD*Y + (tl+TAU_TL)*PL*LY*Y +  tm*PM*MA*A + ta*PA*a0*A;

tauTLdef..   GOVT =e= PA * g0;

tauLSdef..   GOVT =e= PA * g0;

URdef..     PL =G= PA;

incomeh..  HH =e= PL*(ly0+l0)*(1-UR) - PA*dtax - PA*g0*TAU_LS + RK*kd0 - PA*i0  ;

model mcp123 /marketd.PD, marketa.PA, marketm.PM, marketx.PX, marketfx.PFX, marketk.RK, marketl.PL,
              profity.Y,  profita.A,  profitm.M,  profitx.X,  incomeg.GOVT, incomeh.HH, tauLSdef.TAU
          YDdef.YD, YXdef.YX, KDdef.KD, LYdef.LY, DAdef.DA, MAdef.MA, Cdef.C, LDdef.LD /;

*   Establish a numeraire price index:

HH.FX = HH.L;
mcp123.iterlim = 0;
solve mcp123 using mcp;
```

```
abort$(mcp123.objval > 1e-4) "Benchmark model does not calibrate.";
mcp123.iterlim = 10000;

Parameter   report  Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mcp123.objval > 1e-4) "Scenario fails to solve.";

$ondotl
report("W","%replacement%","%labormarket%") = 100*(W-1);
report("Y","%replacement%","%labormarket%") = 100*(Y.L-1);
report("A","%replacement%","%labormarket%") = 100 * (A.L-1);
report("M","%replacement%","%labormarket%") = 100 * (M.L-1);
report("X","%replacement%","%labormarket%") = 100 * (X.L-1);
report("YD","%replacement%","%labormarket%") = 100 * (YD.L/d0-1);
report("YX","%replacement%","%labormarket%") = 100 * (YX.L/x0-1);
report("KD","%replacement%","%labormarket%") = 100 * (KD.L/kd0-1);
report("LY","%replacement%","%labormarket%") = 100 * (LY.L/ly0-1);
report("DA","%replacement%","%labormarket%") = 100 * (DA.L/d0-1);
report("MA","%replacement%","%labormarket%") = 100 * (MA.L/m0-1);
report("C","%replacement%","%labormarket%") = 100 * (C.L/c0-1);
report("LD","%replacement%","%labormarket%") = 100 * (LD.L/l0-1);
report("PD","%replacement%","%labormarket%") = 100 * (PD.L/PL.L - 1);
report("PX","%replacement%","%labormarket%") = 100 * (PX.L/PL.L - 1);
report("PM","%replacement%","%labormarket%") = 100 * (PM.L/PL.L - 1);
report("PA","%replacement%","%labormarket%") = 100 * (PA.L/PL.L - 1);
report("PL","%replacement%","%labormarket%") = 100 * (PL.L/PL.L - 1);
report("RK","%replacement%","%labormarket%") = 100 * (RK.L/PL.L - 1);
report("PFX","%replacement%","%labormarket%") = 100 * (PFX.L/PL.L - 1);
report("HH","%replacement%","%labormarket%") = 100 * (HH.L/PL.L - 1);
report("GOVT","%replacement%","%labormarket%") = 100 * (GOVT.L/PL.L - 1);

report("TAU_LS","%replacement%","%labormarket%") = 100*TAU_LS.L;
report("TAU_TL","%replacement%","%labormarket%") = 100*TAU_TL.L;
report("UR","%replacement%","%labormarket%") = 100*UR.L;
$offecho

*   Tariff reform:

tm = 0;

*   Consider four alternative closures depending on revenue
*   replacement (lumpsum versus wage tax) and labor market
*   (flexible versus fixed wages).

UR.FX = 0;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
solve mcp123 using mcp;

$set replacement Lump Sum
$set labormarket Flexible
$include %gams.scrdir%report

UR.FX = 0;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
```

```
TAU_LS.FX = 0;
solve mcp123 using mcp;

$set replacement Wage Tax
$set labormarket Flexible
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
solve mcp123 using mcp;

$set replacement Lump Sum
$set labormarket Rigid Wage
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
solve mcp123 using mcp;

$set replacement Wage Tax
$set labormarket Rigid Wage
$include %gams.scrdir%report

option report:1:1:2;
display report;
```

Model MGENESTED: MPSGE Model with Nesting

```
$title  Static 123 Model Ala Devarjan

$include 123data.gms

parameter   cd0         Final demand for domestic goods
            cm0         Final demand for imports
            sigmac      Armington elasticity in final demand /0.5/;

*   In this version of the model, we apply the tariff on imports
*   in the M block, so we then can measure imports as value gross
*   of tariff:

m0 = pm0*m0;

*   Store the benchmark tax revenue in the tax rate parameter:

ta = a0 * ta;

*   Impute final demand for domestic and imported goods:

cd0 = c0 * d0/(d0+m0);
cm0 = c0 * m0/(d0+m0);
```

```
*   Armington supply net final demand:

a0 = d0+m0-cd0-cm0+ta;

*   Recalibrate taxes on A so that tax revenue remains unchanged:

ta = ta/a0;

$ontext
$model:MGE123

$SECTORS:
        Y       ! Production
        A       ! Armington composite
        M       ! Imports
        X       ! Exports

$COMMODITIES:
        PD      ! Domestic price index
        PX      ! Export price index
        PM      ! Import price index
        PA      ! Armington price index
        PL      ! Wage rate index
        RK      ! Rental price index
        PFX     ! Foreign exchange

$CONSUMERS:
        HH      ! Private households
        GOVT    ! Government

$AUXILIARY:
        TAU_LS  ! Lumpsum Replacement tax
        TAU_TL  ! Labor tax replacement
        UR      ! Unemployment rate

$PROD:Y  t:etadx s:esubkl
        O:PD    Q:d0    P:1                             ! YD
        O:PX    Q:x0    P:px0   A:GOVT  T:tx            ! YX
        I:RK    Q:kd0   P:rr0   A:GOVT  T:tk            ! KD
        I:PL    Q:ly0   P:pl0   A:GOVT  T:tl N:TAU_TL   ! LY

$report:
     v:YD   o:PD    prod:Y
     v:YX   o:PX    prod:Y
     v:KD   i:RK    prod:Y
     v:LY   i:PL    prod:Y

$PROD:X
        O:PFX   Q:(pwx*x0)
        I:PX    Q:x0

$PROD:A  s:sigmadm
        O:PA    Q:a0            A:GOVT  t:ta
        I:PD    Q:(d0-cd0)                          ! DA
        I:PM    Q:(m0-cm0)

$report:
    v:DA    i:PD    prod:A
```

```
    v:MA    i:PM    prod:A


$PROD:M
        O:PM    Q:m0
        I:PFX   Q:(pwm*m0/pm0)  A:GOVT t:tm

$DEMAND:GOVT
        E:PFX   Q:bopdef
        E:PA    Q:dtax
        E:PA    Q:g0            R:TAU_LS
        D:PA

$CONSTRAINT:UR
        PL =G= PA;

$CONSTRAINT:TAU_LS
        GOVT =e= PA * g0;

$CONSTRAINT:TAU_TL
        GOVT =e= PA * g0;

$DEMAND:HH  s:sigma    c:sigmac
        E:PA    Q:(-g0)         R:TAU_LS
        E:PA    Q:(-dtax)
        E:RK    Q:kd0
        E:PA    Q:(-i0)
        E:PL    Q:(ly0+l0)          ! Labor endowment = ly0+l0 - UR * (ly0+l0)
        E:PL    Q:(-(ly0+l0))   R:UR
        D:PL    Q:l0
        D:PD    Q:cd0   c:
        D:PM    Q:cm0   c:

$report:
    v:W w:HH
    v:CD    d:PD    demand:HH
    v:CM    d:PM    demand:HH
    v:LD    d:PL    demand:HH

$offtext
$sysinclude mpsgeset mge123

UR.FX = 0;
TAU_TL.FX = 0;
TAU_LS.UP =  INF;
TAU_LS.LO = -INF;
mge123.iterlim = 0;
$include MGE123.GEN
solve mge123 using mcp;
abort$(mge123.objval > 1e-4) "Benchmark model does not calibrate.";
mge123.iterlim = 10000;


parameter   report  Tariff Remove with Revenue Replacement (% impact);

$onechov >%gams.scrdir%report.gms
abort$(mge123.objval > 1e-4) "Scenario fails to solve.";
```

```
report("W","%replacement%","%labormarket%") = 100*(W.L-1);
report("Y","%replacement%","%labormarket%") = 100*(Y.L-1);
report("A","%replacement%","%labormarket%") = 100 * (A.L-1);
report("M","%replacement%","%labormarket%") = 100 * (M.L-1);
report("X","%replacement%","%labormarket%") = 100 * (X.L-1);
report("YD","%replacement%","%labormarket%") = 100 * (YD.L/d0-1);
report("YX","%replacement%","%labormarket%") = 100 * (YX.L/x0-1);
report("KD","%replacement%","%labormarket%") = 100 * (KD.L/kd0-1);
report("LY","%replacement%","%labormarket%") = 100 * (LY.L/ly0-1);
report("DA","%replacement%","%labormarket%") = 100 * (DA.L/(d0-cd0)-1);
report("MA","%replacement%","%labormarket%") = 100 * (MA.L/(m0-cm0)-1);
report("CD","%replacement%","%labormarket%") = 100 * (CD.L/cd0-1);
report("CM","%replacement%","%labormarket%") = 100 * (CM.L/cm0-1);
report("LD","%replacement%","%labormarket%") = 100 * (LD.L/l0-1);
report("PD","%replacement%","%labormarket%") = 100 * (PD.L/PL.L - 1);
report("PX","%replacement%","%labormarket%") = 100 * (PX.L/PL.L - 1);
report("PM","%replacement%","%labormarket%") = 100 * (PM.L/PL.L - 1);
report("PA","%replacement%","%labormarket%") = 100 * (PA.L/PL.L - 1);
report("PL","%replacement%","%labormarket%") = 100 * (PL.L/PL.L - 1);
report("RK","%replacement%","%labormarket%") = 100 * (RK.L/PL.L - 1);
report("PFX","%replacement%","%labormarket%") = 100 * (PFX.L/PL.L - 1);
report("HH","%replacement%","%labormarket%") = 100 * (HH.L/PL.L - 1);
report("GOVT","%replacement%","%labormarket%") = 100 * (GOVT.L/PL.L - 1);

report("TAU_LS","%replacement%","%labormarket%") = 100*TAU_LS.L;
report("TAU_TL","%replacement%","%labormarket%") = 100*TAU_TL.L;
report("UR","%replacement%","%labormarket%") = 100*UR.L;
$offecho

*   Tariff reform:

tm = 0;

UR.FX = 0;
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Lump Sum
$set labormarket Flexible
$include %gams.scrdir%report

UR.FX = 0;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Flexible
$include %gams.scrdir%report

*   Lump sum revenue replacement -- downward rigid wage:
UR.LO = 0;
UR.UP = +inf;
```

```
TAU_LS.UP = +inf;
TAU_LS.LO = -inf;
TAU_TL.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Lump Sum
$set labormarket Rigid Wage
$include %gams.scrdir%report

UR.LO = 0;
UR.UP = +inf;
TAU_TL.UP = +inf;
TAU_TL.LO = -inf;
TAU_LS.FX = 0;
$include MGE123.GEN
solve mge123 using mcp;

$set replacement Wage Tax
$set labormarket Rigid Wage
$include %gams.scrdir%report

option report:1:1:2;
display report;
```

4.53.2 MPSGE Models in GAMS

MPSGE is a *mathematical programming system for general equilibrium analysis* which operates as a subsystem within GAMS. MPSGE is essentially a library of function and Jacobian evaluation routines which facilitates the formulation and analysis of AGE models. MPSGE simplifies the modeling process and makes AGE modeling accessible to any economist who is interested in the application of these models. In addition to solving specific modeling problems, the system serves a didactic role as a structured framework in which to think about general equilibrium systems.

MPSGE requires the GAMS/BASE Module including the MILES MCP solver. Optionally it can use the PATH MCP solver.

4.53.2.1 Introduction

This paper introduces a programming language for economic equilibrium modeling. The paper presents the motivation for the system, the programming syntax, and three small scale examples. A library of larger models are provided with the program. The purpose of the paper is to provide a concise introduction to the modeling environment.

MPSGE is a modeling language specially designed for solving Arrow-Debreu economic equilibrium models. (See Rutherford (1987, 1989).) The name stands for "mathematical programming system for general equilibrium". The idea of the *MPSGE* program is to provide a transparent and relatively painless way to write down and analyze complicated systems of nonlinear inequalities. The language is based on nested constant elasticity of substitution utility functions and production functions. The data requirements for a model include share and elasticity parameters for all the consumers and production sectors included in the model. These may or may not be calibrated from a consistent benchmark equilibrium dataset.

GAMS, the "Generalized Algebraic Modeling System", is a modeling language which was originally developed for linear, nonlinear and integer programming. This language was developed over 15 years ago by Alex Meeraus when he was working at the World Bank. (See Brooke, Kendrick and Meeraus (1988).)

Since that time, *GAMS* has been widely applied for large-scale economic and operations research modeling projects.

MPSGE and *GAMS* embody different philosophies in their designs. *MPSGE* is appropriate for a specific class of nonlinear equations, while *GAMS* is capable of representing any system of algebraic equations. While *GAMS* is applicable in several disciplines, *MPSGE* is only applicable in the analysis of economic models within a particular domain. The expert knowledge embodied in *MPSGE* is of particular use to economists who are interested in the insights provided by formal models but who are unable to devote many hours to programming. *MPSGE* provides a structured framework for novice modellers. When used by experts, *MPSGE* reduces the setup cost of producing an operational model and the cost of testing alternative specifications.

Prior to the connection with *GAMS*, the "achilles heel" of *MPSGE* had been the process by which input data was translated into the tabular format of the *MPSGE* input file. For small models, this translation was not difficult. Given a calibrated "benchmark equilibrium dataset", a couple of hours with a word processor is usually enough time to generate and investigate a moderately large model. If, however, a model involves several classes of sectors and agents, a wide range of tax instruments and large tables of input data, the word- processor approach is impossible. When there are many numbers, there are many opportunities for oversights and typographical errors.

In contrast, the *GAMS* modeling language is designed for managing large datasets. The use of sets and detached-coefficient matrix notation makes the *GAMS* environment very nice for both developing balanced benchmark datasets and for writing solution reports. For large complicated models, a shortcoming of the *GAMS* modeling environment lies in the specification of the nonlinear equations. Economic equilibrium models, particularly those based on complicated functions such as nested CES, are easier to understand at an abstract level than they are to specify in detail, and the translation of a model from input data into algebraic relations can be a tedious and error- prone undertaking.

The interface between *GAMS* and *MPSGE* combines the strengths of both programs. The system uses *GAMS* as the "front end" and "back end" to *MPSGE* facilitating data handling and report writing. The language employs an extended *MPSGE* syntax based on *GAMS* sets, so that model specification is very concise. In addition, the system includes two large-scale solvers, MILES (Rutherford (1993)) and PATH (Ferris and Dirkse (1993)), which may be used interchangeably. The availability of two algorithms greatly enhances robustness and reliability.

The remainder of this paper is organized as follows. Section 2 introduces *MPSGE* input syntax and the *GAMS* interface using a small two-sector model of taxation. Section 3 extends the 2x2 model to illustrate how the software is used to perform equal-yield (differential) tax policy analysis and to analyze tax reform in a model with endogenous taxation. Section 4 provides a brief summary and conclusion. The paper introduces language features largely through example. Details on language syntax and program structure are provided in two appendices. Appendix A provides a complete statement of *MPSGE* syntax and a summary of differences with the original (1989) language. Appendix B provides an overview of the modeling environment and the structure of *GAMS* input files which employ *MPSGE*.

Before proceeding, both to placate impatient readers and to provide some hands-on experience for novices, I recommend that readers install the *GAMS* system, then retrieve and process the library file THREEMGE which contains three *MPSGE* models (`HARBERGER`, `SHOVEN` and `SAMUELSON`). Two commands to retrieve and run these models:

```
gamslib threemge
gams threemge
```

After having processed this file, print the listing files (`THREEMGE.LST`) for reference.

4.53.2.2 A Mathematical Introduction

Mathiesen (1985) demonstrated that an Arrow-Debreu general economic equilibrium model could be formulated and efficiently solved as a complementarity problem. Mathiesen's formulation may be posed in terms of three sets of "central variables":

p = a non-negative n-vector of commodity prices including all final goods, intermediate goods and primary factors of production;

y = a non-negative m-vector of activity levels for constant returns to scale production sectors in the economy; and

M = an h-vector of income levels, one for each "household" in the model, including any government entities.

An equilibrium in these variables satisfies a system of three classes of nonlinear inequalities.

Zero Profit

The first class of constraint requires that in equilibrium no producer earns an "excess" profit, i.e. the value of inputs per unit activity must be equal to or greater than the value of outputs. This can be written in compact form as:

$$-\prod_j(p) = C_j(p) - R_j(p) \geq 0 \quad \forall j$$

where $\prod_j(p)$ is the unit profit function, the difference between unit revenue and unit cost, defined as:

$$C_j(p) = \min\{\sum_i p_i x_i | f_j(x) = 1\}$$

and

$$R_j(p) = \max\{\sum_i p_i y_i | g_j(y) = 1\}$$

where f and g are the associated production functions characterizing feasible input and output. For example, if we have:

$$f(x) = \phi \prod_i x_i^{\alpha_i} \quad \sum_i \alpha_i = 1, \quad \alpha_i \geq 0$$

and

$$g(y) = \psi \max_i \frac{y_i}{\beta_i} \quad \beta_i \geq 0$$

then the dual functions will be:

$$C(p) = \frac{1}{\phi} \prod_i (\frac{p_i}{\alpha_i})^{\alpha_i}$$

and

$$R(p) = \sum_i \beta_i p_i$$

Market Clearance

The second class of equilibrium conditions is that at equilibrium prices and activity levels, the supply of any commodity must balance or exceed excess demand by consumers. We can express these conditions as:

$$\sum_j y_i \frac{\delta \Pi_j(p)}{\delta p_i} + \sum_h \omega_{ih} \geq \sum_h d_{ih}(p, M_h)$$

in which the first sum, by Shepard's lemma, represents the net supply of good i by the constant-returns to scale production sectors, the second sum represents the aggregate initial endowment of good i by households, and the sum on the right-hand-side represents aggregate final demand for good i by households, given market prices p and household income levels M. Final demand are derived from budget-constrained utility maximization:

$$d_{ih}(p, M_h) = argmax\{U_h(x) | \sum_i p_i x_i = M_h\}$$

in which U_h is the utility function for household h.

Income Balance

The third condition is that at an equilibrium, the value of each agent's income must equal the value of factor endowments:

$$M_h = \sum_i p_i \omega_{ih}$$

We always work with utility functions which exhibit non-satiation, so Walras' law will always hold:

$$\sum_i p_i d_{ih} = M_h = \sum_i p_i \omega_{ih}$$

Aggregating market clearance conditions using equilibrium prices and the zero profit conditions using equilibrium activity levels, it then follows that:

$$\sum_j y_j \Pi_j(p) = 0$$

or

$$y_j \Pi_j(p) = 0 \quad \forall_j$$

Furthermore, it follows that:

$$p_i \left(\sum_j y_i \frac{\delta \Pi_j(p)}{\delta p_i} + \sum_h \omega_{ih} - \sum_h d_{ih}(p, M_h) \right) = 0 \quad \forall_i$$

In other words, complementary slackness is a feature of the equilibrium allocation even though it is not imposed as an equilibrium condition, per-se. This means that in equilibrium, an production activity which is operated makes zero profit and any production activity which earns a negative net return is idle. Likewise, any commodity which commands a positive price has a balance between aggregate supply and demand, and any commodity in excess supply has an equilibrium price of zero.

4.53.2.3 A small example: Harberger

This section of the paper introduces *MPSGE* model building using a two- good, two-factor (2x2) example. This is addressed to readers who may be unfamiliar with *GAMS* and/or the original (scalar) *MPSGE* syntax. The discussion provides some details on the formulation and specification of one small model from the *MPSGE* library. Subsequently, two extensions are presented, one which illustrates equal yield constraints and another which introduces a pure public good. These examples are by no means exhaustive of the classes of equilibrium structures which can be investigated using the software, but they do provide a starting point for new users.

The structure of *MPSGE* model `HARBERGER` is "generic" Arrow-Debreu with taxes. Households obtain income by supplying factors of production to industry or collecting tax revenue. This income is then allocated between alternative goods in order to maximize welfare subject to the budget constraint.

Firms operate subject to constant returns to scale, selecting factor inputs in order to minimize cost subject to technological constraints. For an algebraic description of a model closely related to this one, see Shoven and Whalley (1984). The present model differs in two respects from the Shoven-Whalley example. First, in this model there are intermediate inputs to production while in the Shoven-Whalley model goods are produced using only value-added. Second, this model incorporates a labor-leisure choice so that the excess burden of factor taxes here incorporates the disincentive to work associated with a lower net wage.

Benchmark Data

Table 1 presents most of the input data for a two good, two factor, closed economy model. This is an economy in which, initially, taxes are levied only on capital inputs to production. We treat tax revenue as though it were returned lump-sum to the households.

	Sectors		Consumers		
	X	Y	OWNERS	WORKERS	GOVT
PX	100	-20	-30	-50	
PY	-10	80	-40	-30	
PK	-20	-40	60		
PL	-50	-10		100	-40
TRNS			10	20	-30
TK	-20	-10			30

The input data is presented in the form of a balanced matrix, the entries in which represent the value of economic transactions in a given period (typically one year). Social accounting matrices (SAMs) can be quite detailed in their representation of an economy, and they are also quite flexible. All sorts of inter-account taxes, subsidies and transfers can be represented through an appropriate definition of the accounts.

Traditionally, a SAM is square with an exact correspondence between rows and columns. (For an introduction, see Pyatt and Round, "Social Accounting Matrices: A Basis for Planning", The World Bank, 1985.) The numbers which appear in a conventional SAM are typically positive, apart from very special circumstances, whereas the rectangular SAM displayed in Table 1 follows a sign convention wherein supplies or receipts are represented by positive numbers and demands or payments are represented by negative numbers. Internal consistency of a rectangular SAM implies that row sums and column sums are zero. This means that supply equals demand for all goods and factors, tax payments equal tax receipts, there are no excess profits in production, the value of each household expenditure equals the value of factor income plus transfers, and the value of government tax revenue equals the value of transfers to households.

With simple *MPSGE* models, it is convenient to use a *rectangular* SAM format. This format emphasizes how the *MPSGE* program structure is connected to the benchmark data. In the rectangular SAM, we have one row for every market (traded commodity). In the present model, there are four markets, for goods X and Y and factors L and K.

There are two types of columns in the rectangular SAM, corresponding to production sectors and consumers. In the present model, there are two production sectors (X and Y) and three consumers (`OWNERS`, `WORKERS` and `GOVT`).

Data Entry in *GAMS*

Consider a generalized version of the model in which the set of production sectors be denoted S (here, $S = \{X, Y\}$). Let the set of goods be G. Production sectors are mapped one-to-one with the goods, so we see that sets S and G are in fact the same set. Let F denote the set of primary factors (here, $F = L, K$), and let H denote the set of households (here `H={OWNER,WORKER}`). Now that we have identified the underlying sets, we may interpret the input matrix as a set of parameters with which we can easily specify the benchmark equilibrium. (See Table 2.) It is quite common to begin a general equilibrium modeling project with a large input-output table or social accounting matrix which may then be mapped onto a number of submatrices, each of which is dimensioned according to the underlying sets used in the model.

```
                     Sectors        Consumers
                     ----------------------------------------
                        (S)         Households(H)   Government
                     ----------------------------------------
Goods Markets (G):   A(G,S)-B(G,S)   -C(G,H)
Factor Markets (F): -FD(F,S)        E(F,H)-D(F,H)
Capital taxes:      -T("K",S)                          GREV
Transfers:                            TRN(H)          -GREV
```

The *GAMS* specification of benchmark data is presented in Table 3 which begins with a statement of the underlying sets (G, F, H). The statement `"ALIAS (S,G)"`; simply says that S and G both reference X, Y. Thereafter follows the social accounting data table and declarations for the various submatrices. The parameters `ELAS()` and `ESUB()` are elasticities ("free parameters") which can be chosen independently from the benchmark accounts. The parameters $TF$and PF are calibrated tax and reference price arrays which are computed given benchmark factor and tax payments. (In this model, average and marginal tax rates are not distinguished, so the benchmark marginal tax rate is simply the tax payment divided by the net factor income.)

A general equilibrium model determines only relative prices. For purposes of reporting or constructing value-indices, we use Laspeyres quantity index, `THETA(G)`, the elements of which correspond to shares of aggregate consumer expenditure in the benchmark period.

Table 3: Data Specification in *GAMS* for the 2x2 Model Harberger

```
*       SECTION (i)     DATA SPECIFICATION AND BENCHMARKING

SETS        G           GOODS AND SECTORS /X, Y/,
            F           PRIMARY FACTORS   /K, L/,
            H           HOUSEHOLDS     /OWNER, WORKER/;

ALIAS (S,G);

TABLE SAM(*,*)  SOCIAL ACCOUNTING MATRIX

                X       Y       OWNER   WORKER  GOVT
X               100     -20     -30     -50
Y               -10      80     -40     -30
```

```
K               -20     -40      60
L               -50     -10              60
TK              -20     -10                       30
TRN                             10      20     -30

PARAMETER
        A(S)            BENCHMARK OUTPUT
        B(G,S)          USE MATRIX (GOODS INPUTS BY SECTOR)
        C(G,H)          HOUSEHOLD DEMAND
        FD(F,S)         FACTOR DEMAND BY SECTOR
        E(F,H)          FACTOR ENDOWMENTS
        D(F,H)          FACTOR DEMAND BY HOUSEHOLDS
        T(F,S)          TAX PAYMENT BY FACTOR BY SECTOR
        TRN(H)          TRANSFER REVENUE
        ELAS(S)         ELASTICITY OF SUBSTITUTION IN PRODUCTION
        ESUB(H)         ELASTICITY OF SUBSTITUTION IN DEMAND
        GREV            BENCHMARK GOVERNMENT REVENUE
        TF(F,S)         FACTOR TAX RATE
        PF(F,S)         BENCHMARK FACTOR PRICES GROSS OF TAX
        THETA(G)        WEIGHTS IN NUMERAIRE PRICE INDEX
        WBAR(H)         BENCHMARK WELFARE INDEX;

*       EXTRACT DATA FROM THE SOCIAL ACCOUNTING MATRIX:

A(S)   = SAM(S,S);
B(G,S) = MAX(0, -SAM(G,S));
C(G,H) = -SAM(G,H);
FD(F,S) = -SAM(F,S);
E(F,H) = SAM(F,H);
D(F,H) = 0;
TRN(H) = SAM("TRN",H);
T("K",S) = -SAM("TK",S);

*       INSTALL "FREE" ELASTICITY PARAMETERS:

E("L","WORKER") = 100;
D("L","WORKER") =  40;
ELAS(S) = 1;
ESUB(H) = 0.5;

*       INSTALL FUNCTIONS OF BENCHMARK VALUES:

GREV     = SUM(H, TRN(H));
TF(F,S) = T(F,S) / FD(F,S);
PF(F,S) = 1 + TF(F,S);
THETA(G) = SUM(H, C(G,H));
THETA(G) = THETA(G) / SUM(S, THETA(S));
WBAR(H) = SUM(G, C(G,H)) + SUM(F, D(F,H));
```

Model Specification

The *MPSGE* description of this model is shown in Table 4. Declarations following the `$MODEL` statement indicate that the model involves one class of production activities ($AL(S)$), three classes of commodities ($P(G)$, $W(F)$ and PT), and two types of consumers, private consumers ($RA(H)$), and a government "consumer" (`GOVT`).

One `$PROD:` block describes the single class of production activities, and two `$DEMAND:` blocks characterize endowments and preferences for the two classes of consumers.

Consider the records associated with production sector $AL(S)$. The entries on the first line of a `$PROD:` block are elasticity values. The "s:0" field indicates that the top-level elasticity of substitution between inputs is zero (Leontief). The entry `"a:ELAS(S)"` indicates that inputs identified as belonging to the `"a:"` aggregate trade off with an elasticity of substitution `ELAS(S)` (at the second level of the production function). In these production functions, the primary factors ($W(F)$) are identified as entering in the $a : aggregate$.

TABLE 4: MPSGE Model Specification and Benchmark Replication

```
*       SECTION (ii)    MPSGE MODEL DECLARATION

$ONTEXT

$MODEL:HARBERGER

$SECTORS:
      AL(S)

$COMMODITIES:
      P(G)  W(F)  PT

$CONSUMERS:
      RA(H) GOVT

$PROD:AL(S)  s:0  a:ELAS(S)
      O:P(S)      Q:A(S)
      I:P(G)      Q:B(G,S)
      I:W(F)      Q:FD(F,S)   P:PF(F,S)   A:GOVT   T:TF(F,S)  a:

$DEMAND:RA(H)  s:1  a:ESUB(H)
      D:P(G)      Q:C(G,H)   a:
      D:W(F)      Q:D(F,H)
      E:W(F)      Q:E(F,H)
      E:PT        Q:TRN(H)

$DEMAND:GOVT
      D:PT        Q:GREV

$REPORT:
        V:CD(G,H)       D:P(G)          DEMAND:RA(H)
        V:DF(F,H)       D:W(F)          DEMAND:RA(H)
        V:EMPLOY(S)     I:W("L")        PROD:AL(S)
        V:WLF(H)        W:RA(H)


$OFFTEXT

*       Invoke the preprocessor to declare the model for GAMS:

$SYSINCLUDE mpsgeset HARBERGER

*       ---------------------------------------------------------------
*       SECTION (iii)   BENCHMARK REPLICATION

HARBERGER.ITERLIM = 0;
$INCLUDE HARBERGER.GEN
SOLVE HARBERGER USING MCP;
ABORT$(ABS(HARBERGER.OBJVAL) GT 1.E-4)
```

```
    "*** HARBERGER benchmark does not calibrate.";
HARBERGER.ITERLIM = 1000;
```

The records within a `$PROD:` block begin with `"O:"` or `"I:"`. An `"O:"` indicates an output and an `"I:"` represents an input. In both types of records, `"Q:"` is a "quantity field" indicating a reference input or output level of the named commodity. A `"P:"` signifies a reference price field. This price is measured as a user cost, gross of applicable taxes. The default values for reference price and reference quantity are both unity (i.e., a value of 1 is installed if a `P:` or `Q:` field is missing).

The `A:` and `T:` fields in a `$PROD:` block indicate tax agent and ad-valorem tax rate, respectively. The tax agent is specified before the tax rate. A single input or output coefficient may have two or more taxes applied. Consumers are treated symmetrically, and there is thus no restriction on the consumer to whom the tax is paid. Typically, however, one consumer is associated with the government.

To better understand the relationship between reference prices and tax rate specification, consider inputs of $W.K$ to sector $AL.X$ in this model. The benchmark payment to capital in the X sector is 20 and the tax payment is 20. Hence the ad-valorem tax rate in the benchmark equilibrium is 100% ($T:1$), and the reference price of capital, market price of unity times $(1 + 100\%)$, is 2 ($P:2$). If in a counterfactual experiment the tax rate on capital inputs to sector X is altered, this will change the $T:$ field but it will not change the $P:$ field. $Q:$ and $P:$ characterize a reference equilibrium point, and these are therefore unaffected by subsequent changes in the exogenous parameters.

It is important to remember that the `$PROD:AL(S)` block represents as many individual production functions as there are elements in set S (two in this case). Within the `$PROD:AL(S)` block, inputs refer to sets G and F , while the output coefficient, $O : P(S)$, refers only to set S. Sets referenced within a commodity name in an $I:$ or $O:$ field may be sets which are "controlled" by the sets referenced in the function itself, in which case only a scalar entry is produced. In `$PROD:AL(S)` there are only outputs of commodity S in sector S.

The first line of a `$DEMAND` block also contains fields (e.g.,`s:, a:, b:` etc.) which represent elasticities of substitution. The subsequent records may begin with either an $E:$ field or a $D:$ field. These, respectively, represent commodity endowments and demands. In the demand fields, the $P:$ and $Q:$ entries are interpreted as reference price and reference quantity, analogous to the input fields in a `$PROD` block. Ad-valorem taxes may not be applied on final demands, so that if consumption taxes are to be applied in a model they must be levied on production activities upstream of the final demand.

The benchmark values for all activity levels and prices are equal to the default value of unity, and therefore we are able to specify values in the $Q:$ fields directly from the benchmark data. An equivalent model could be specified in which the benchmark activity levels for `AL(S)` equal, for example, `A(S,S)`. This would then require rescaling the input and output coefficients for those sectors, and it would not necessarily be helpful, because in a scaled model it is more difficult to verify consistency of the benchmark accounts and MPSGE input file. Furthermore, for numerical reasons it is advisable to scale equilibrium values for the central variables to be close to unity.

Government transfers to households are accomplished through the use of an "artificial commodity" `(PT)`. The government is identified as the agent who receives all tax revenue (see the `A:GOVT` entry in both of the `$PROD:` blocks). Commodity PT is the only commodity on which `GOVT` spends this income, hence government tax revenue is divided between the two households in proportion to their endowments of the artificial good. In order to scale units so that the benchmark price of `PT` is unity, the $30 of government tax revenue chases 10 units of PT assigned to OWNER and 20 units assigned to WORKER. (See values for `TRN(H)` in Table 3.)

The `$REPORT` section of the input file requests the solution system to return values for inputs, outputs, final demands or welfare indices at the equilibrium. Only those items which are requested will be written to the solution file. Each record in the report block begins with a $V:$ (variable name) field. These names must be distinct from all other names in the model. The second field of the report record must have one of the labels $I:$, $O:$ or $D:$ followed by a commodity name, or the label $W:$ followed by a consumer name.

The third field's label must be `"PROD:"` in an I : or O : record, and it must be `"DEMAND:"` if it is a `D:` record.

An Algebraic Formulation of Harberger Model

The unit cost of production in sector $A_h s$ is given by a nested Leontief-CES function defined over the cost of intermediate inputs and primary factors with ad-valorem taxes on factor demands. In equilibrium, the unit cost must be no less than the market price of output:

Zero Net Profit

$$-\prod_s = \sum_g p_g B_{bs} \quad + \quad \phi_s \left(\sum_f \alpha_{fs} (\omega_f (1 + t_{fs}))^{1-\sigma_s} \right)^{\frac{1}{1-\sigma_s}} \quad - \quad p_s \geq 0 \quad \forall s | \sigma_s \neq 1$$

and

$$-\prod_s = \sum_g p_g B_{bs} \quad + \quad \phi_s \left(\prod_f (\omega_f (1 + t_{fs}))^{\alpha_{fs}} \right) \quad - \quad p_s \geq 0 \quad \forall s | \sigma_s = 1$$

Income Balance for Government

Government tax income (PT) is determined by the value of tax revenue, calculated using activity levels, compensated demands, market prices and ad-valorem tax rates:

$$PT = \sum_s \frac{\partial \prod_s}{\partial (\omega_f (1 + t_{fs}))} \quad AL_s \omega_f t_{fs}$$

Income Balance for Households

Household income is determined by the net of tax return to primary factors plus the household share of government revenue:

$$M_h = \sum_f \omega_f E_{fh} + s_h PT$$

Market Clearance for Goods

Producer output is equal to the sum of intermediate plus final demand:

$$AL_g \geq \sum_s AL_s B_{gs} + \sum_h \frac{\gamma h M_h}{e_h(p)} \left(\frac{e_h(p)}{p_g} \right)^{\sigma_h}$$

where γ_h is the household budget share devoted to the consumption of goods, and $e_h h$ is the "unit expenditure function" which may be written:

$$e_h(p) \equiv \left(\sum_i \beta_{ih} p_i^{1-\sigma_h} \right)^{\frac{1}{1-\sigma_h}}$$

Market Clearance for Factors

The aggregate supply of factors equals the sum of producer and consumer demand. Producers pay taxes on factor inputs, consumers do not because we consider these demands to be "leisure" or "household production". Consumer demands for factors are specified as Cobb-Douglas (constant budget shares):

$$\sum_h E_{fh} = \sum_s AL_s \frac{\partial \prod_s}{\partial (\omega_f (1 + t_{fs}))} + \sum_h \frac{\mu_{fh} M_h}{\omega_f}$$

***MPSGE* Formulation: Key Ideas**

There are two points regarding the MPSGE function format which are important yet easily misunderstood by new users:

1. *The elasticities together with the reference quantities and reference prices of inputs and outputs completely characterize the underlying nested CES functions.* No other data fields in the `$PROD:` block alters the technology. If, for example, a tax rate changes as part of a counter-factual experiment, this has no effect on the reference price. The value in the $P:$ field depends on the benchmark value of the $T:$ field if the model has been calibrated, but subsequent changes in $T:$ do not change the underlying technology.

2. *Tax rates are interpreted differently for inputs and outputs.* The tax rate on inputs is specified on a net basis, while the tax rate on outputs is specified on a gross basis. That is, the user cost of an input with market price p subject to an ad-valorem tax at rate t is $p(1+t)$, while the user cost of an output subject to an ad-valorem tax at rate t is $p(1-t)$. (A tax increases the producer cost of inputs and decreases the producer value of outputs.)

MPSGE provides a limited number of economic components with which complex models can be constructed. There are some models which lie outside the *MPSGE* domain, but in many cases it is possible to recast the equilibrium structure in order to produce an *MPSGE* model which is logically equivalent to the original model - usually after having introduced some sort of artificial commodity or consumer. In the present model, the use of commodity PT to allocate government revenue between households provides a fairly typical example of how this can be done. In the process of making such a transformation, one often gains a meaningful economic insight.

The Solution Listing

The detailed solution listing for model HARBERGER is shown in Table 5. The standard *GAMS* report facilities display solution values. Central variables are always either fixed (upper = lower), or they are non-negative (lower bound = 0, upper bound = `+INF`). The MARGINAL column in the solution report returns the value of the associated slack variable. Complementarity implies that in equilibrium, either the level value of a variable will be positive or the marginal value will be positive, but not both.

The output file (not shown) also provides details on the computational process. For an explanation of these statistics, see Rutherford (1993).

TABLE 5: *GAMS* Solution Listing for Model HARBERGER

```
---- VAR AL

     LOWER     LEVEL     UPPER    MARGINAL
X      .        1.000     +INF       .
Y      .        1.000     +INF       .

---- VAR P

     LOWER     LEVEL     UPPER    MARGINAL
X      .        1.000     +INF       .
Y      .        1.000     +INF       .

---- VAR W

     LOWER     LEVEL     UPPER    MARGINAL
K      .        1.000     +INF       .
L      .        1.000     +INF       .

                       LOWER     LEVEL     UPPER    MARGINAL
---- VAR PT              .        1.000     +INF       .

---- VAR RA

          LOWER     LEVEL     UPPER    MARGINAL
```

```
OWNER       .       70.000     +INF       .
WORKER      .      120.000     +INF       .

                       LOWER     LEVEL     UPPER    MARGINAL
---- VAR GOVT            .       30.000     +INF       .

---- VAR CD

            LOWER     LEVEL     UPPER    MARGINAL
X.OWNER       .       30.000     +INF       .
X.WORKER      .       50.000     +INF       .
Y.OWNER       .       40.000     +INF       .
Y.WORKER      .       30.000     +INF       .

---- VAR DF

            LOWER     LEVEL     UPPER    MARGINAL
K.OWNER       .         .        +INF       EPS
K.WORKER      .         .        +INF       EPS
L.OWNER       .         .        +INF       EPS
L.WORKER      .       40.000     +INF       .

---- VAR EMPLOY

     LOWER     LEVEL     UPPER    MARGINAL
X      .       50.000     +INF       .
Y      .       10.000     +INF       .

---- VAR WLF

           LOWER     LEVEL     UPPER    MARGINAL
OWNER        .        1.000     +INF       .
WORKER       .        1.000     +INF       .


**** REPORT SUMMARY :        0     NONOPT
                             0 INFEASIBLE
                             0  UNBOUNDED
                             0     ERRORS
```

Computing Counter-factual Scenarios

Table 6 presents the *GAMS* code for specification and solution of three counterfactual equilibria. In these experiments, the nonuniform system of capital taxes from the benchmark is replaced by three alternative uniform factor tax structures: a tax on labor, a tax on capital, and a tax on both labor and capital. In each case, the tax rate is chosen to replace the benchmark tax revenue at benchmark prices and demand (ignoring induced substitution effects). Following each solution, the equilibrium values for tax revenue, welfare (Hicksian equivalent variation), employment, prices and output are stored in parameter `REPORT`.

TABLE 6: Specification and Processing of Counter-Factual Scenarios

```
*       ---------------------------------------------------------------
*       SECTION (iv)  COUNTER-FACTUAL SPECIFICATION AND SOLUTION:

SET SC COUNTERFACTUAL SCENARIOS TO BE COMPUTED /
        L       UNIFORM TAX ON LABOR,
        K       UNIFORM TAX ON CAPITAL,
```

```
        VA      UNIFORM VALUE-ADDED TAX/

PARAMETER       TAXRATE(F,S,SC) COUNTERFACTUAL TAX RATES,
                REPORT(*,*,*,SC)  SOLUTION REPORT - % CHANGES,
                PINDEX          PRICE DEFLATOR;


*       SPECIFY COUNTER-FACTUAL TAX RATES TO ACHIEVE CETERIS
*       PARIBUS BALANCED BUDGET:

TAXRATE("L",S,"L")  = GREV / SUM(G, FD("L",G));
TAXRATE("K",S,"K")  = GREV / SUM(G, FD("K",G));
TAXRATE("L",S,"VA") = GREV / SUM((F,G), FD(F,G));
TAXRATE("K",S,"VA") = GREV / SUM((F,G), FD(F,G));

LOOP(SC,

*       INSTALL TAX RATES FOR THIS COUNTERFACTUAL:

        TF(F,S) = TAXRATE(F,S,SC);

$INCLUDE HARBERGER.GEN
SOLVE HARBERGER USING MCP;

*       ---------------------------------------------------------------
*       SECTION (v)  REPORT WRITING:

*       REPORT SOME RESULTS:

        PINDEX = SUM(G, P.L(G) * THETA(G));

        REPORT("REVENUE","_",SC) = 100 * (PT.L/PINDEX - 1);
        REPORT("TAXRATE","_",SC) =
                100 * SMAX((F,S), TAXRATE(F,S,SC));
        REPORT("WELFARE",H,SC)   = 100 * (WLF.L(H) - 1);
        REPORT("EMPLOY",S,SC)    = 100 * (EMPLOY.L(S)/FD("L",S) - 1);
        REPORT("PRICE",G,SC)     = 100 * (P.L(G)/PINDEX - 1);
        REPORT("PRICE",F,SC)     = 100 * (W.L(F)/PINDEX - 1);
        REPORT("OUTPUT",S,SC)    = 100 * (AL.L(S) - 1);
);
DISPLAY REPORT;
```

4.53.2.4 Alternative models: Shoven and Samuelson

The "standard" *MPSGE* model is based on fixed endowments and tax rates, but many empirical models do not fit into this structure. For example, in the model HARBERGER, the level of each replacement tax was specified to be consistent with "equal yield", but as a result of the endogenous response of prices and quantities, the resulting tax revenues differed significantly from the benchmark levels. For example, when the capital tax is replaced by a uniform labor tax at a rate which, in the absence of labor supply response, produces "equal yield", we find that tax revenue in fact declines by 39%. In order to perform differential (equal yield) tax policy analysis, it is therefore necessary to accommodate the endogenous determination of tax rates as part of the equilibrium computation. This is one of many possible uses of "auxiliary variables" in *MPSGE*.

Tax Analysis with Equal Yield

Table 7 presents the *MPSGE* model definition for test problem SHOVEN. This model is equivalent to the HARBERGER, apart from the addition of an auxiliary variable TAU. Within *MPSGE*, auxiliary variables

can either represent price-adjustment instruments (endogenous taxes) or they can represent a quantity-adjustment instruments (endowment rations). In model SHOVEN, TAU is used to proportionally scale factor taxes in order to achieve a target level of government revenue. The auxiliary variable first appears in the $PROD:AL(S) block, following the declaration of a tax agent. There are two fields associated with an endogenous tax. The first field (N:) gives the name of the auxiliary variable which will scale the tax rate. The second field (M:) specifies the multiplier. If the M : field is omitted, the multiplier assumes a default value of unity. If the value in the M : field is zero, the tax does not apply.

The auxiliary variable TAU also appears at the bottom of the file where it labels an associated inequality constraint. This constraint exhibits complementary slackness with the associated non-negative auxiliary variable (i.e., if TAU is positive, the constraint must hold with an equality, whereas if the constraint is non-binding TAU must be zero). An auxiliary variable may or may not appear in its associated constraint.

The constraint associated with TAU is based on a price index defined by THETA(G). The constraint assures a level of tax revenue such that the value of transfers to households is held constant. (Endowments of the commodity PT are fixed, so when the value of PT is fixed, then so too are the value of transfers from GOVT to each of the households.)

SHOVEN illustrates how an auxiliary variable can be interpreted as a tax instrument. In the *MPSGE* syntax, auxiliary variables may also be employed to endogenously determine commodity endowments. There is no restrictions on how a particular auxiliary variable is to be interpreted. A single variable could conceivably serve simultaneously as an endogenous tax as well as a endowment ratio, although this would be rather unusual.

TABLE 7: Differential Tax Policy Analysis

```
MPSGE PREPROCESSOR VERSION 1/94      286/386/486 DOS

 0  $MODEL: SHOVEN
 1
 2  $SECTORS:
 3        AL(S)
 4
 5  $COMMODITIES:
 6        P(G)  W(F)  PT
 7
 8  $CONSUMERS:
 9        RA(H) GOVT
10
11  $AUXILIARY:
12        TAU
13
14  $REPORT:
15          V:CD(G,H)       D:P(G)          DEMAND:RA(H)
16          V:DF(F,H)       D:W(F)          DEMAND:RA(H)
17          V:EMPLOY(S)     I:W("L")        PROD:AL(S)
18          V:WLF(H)        W:RA(H)
19
20  $PROD:AL(S)  s:0  a:ELAS(S)
21        O:P(G)      Q:A(G,S)
22        I:P(G)      Q:B(G,S)
23        I:W(F)      Q:FD(F,S)   P:PF(F,S)
24  +       A:GOVT  N:TAU$TF(F,S)  M:TF(F,S)$TF(F,S) a:
25
26  $DEMAND:RA(H)  s:1  a:ESUB(H)
27        D:P(G)      Q:C(G,H)   a:
28        D:W(F)      Q:D(F,H)
29        E:W(F)      Q:E(F,H)
```

```
30        E:PT         Q:TRN(H)
31
32  $DEMAND:GOVT
33        D:PT         Q:GREV
34
35  $CONSTRAINT:TAU
36                   PT =G= SUM(G, THETA(G) * P(G));
37
38  $OFFTEXT
```

An Algebraic Formulation of Shoven Model

Zero Net Profit

The unit cost of production in sector $A_h s$ is given by a nested Leontief-CES function defined over the cost of intermediate inputs and primary factors with ad-valorem taxes on factor demands. Unlike the Harberger models, tax rates in this model are determined endogenously. In equilibrium, the unit cost must be no less than the market price of output:

$$-\prod_s = \sum_g p_g B_{bs} \quad + \quad \phi_s\left(\sum_f \alpha_{fs}(\omega_f(1+\tau t_{fs}))^{1-\sigma_s}\right)^{\frac{1}{1-\sigma_s}} \quad - \quad p_s \geq 0 \quad \forall s|\sigma_s \neq 1$$

and

$$-\prod_s = \sum_g p_g B_{bs} \quad + \quad \phi_s\left(\prod_f(\omega_f(1+\tau t_{fs}))^{\alpha f_s}\right) \quad - \quad p_s \geq 0 \quad \forall s|\sigma_s = 1$$

Income Balance for Government

Government tax income (PT) is determined by the value of tax revenue, calculated using activity levels, compensated demands, market prices and ad-valorem tax rates:

$$PT = \sum_s \frac{\partial \prod_s}{\partial(\omega_f(1+\tau t_{fs}))} \quad AL_s \omega_f \tau t_{fs}$$

Equal Yield

In equilibrium, tax rates are multiplicatively adjusted to achieve a target level of government revenue:

$$PT = \sum_g \theta_g p_g$$

Income Balance for Households

Household income is determined by the net of tax return to primary factors plus the household share of government revenue:

$$M_h = \sum_f \omega_f E_{fh} + s_h PT$$

Market Clearance for Goods

Producer output is equal to the sum of intermediate plus final demand:

$$AL_g \geq \sum_s AL_s B_{gs} + \sum_h \frac{\gamma h M_h}{e_h(p)} \left(\frac{e_h(p)}{p_g} \right)^{\sigma_h}$$

where γ_h is the household budget share devoted to the consumption of goods, and $e_h h$ is the "unit expenditure function" which may be written:

$$e_h(p) \equiv \left(\sum_i \beta_{ih} p_i^{1-\sigma_h} \right)^{\frac{1}{1-\sigma_h}}$$

Market Clearance for Factors

The aggregate supply of factors equals the sum of producer and consumer demand. Producers pay taxes on factor inputs, consumers do not because we consider these demands to be "leisure" or "household production". Consumer demands for factors are specified as Cobb-Douglas (constant budget shares):

$$\sum_h E_{fh} = \sum_s AL_s \frac{\partial \prod_s}{\partial(\omega_f(1+\tau t_{fs}))} + \sum_h \frac{\mu_{fh} M_h}{\omega_f}$$

Public Goods and Endogenous Taxation

Consider a final extension of the 2x2 model in which tax revenue funds a pure public good. Model SAMUELSON presented in Table 8. This model illustrates one of several ways that public goods can be modelled in *MPSGE*. Here the level of public provision is determined by a Samuelson-condition equating the sum of individual marginal rates of substitution (marginal benefit) with the marginal rate of transformation (marginal cost). Unlike the equal yield formulation, the tax revenues collected by `GOVT`are not returned lump-sum but are instead used to finance provision of a pure public good. This representation of government has not been widely adopted in the CGE literature, perhaps because of the difficulties involved in specifying preferences for public goods.

The relevant characteristic of a pure public good entering final demand is that each consumer "owns" the same quantity. Agents' attitudes toward public goods differ, and because there is no market, agents' valuations of the public good will also differ. In an *MPSGE* model, the separate valuations are accommodated through the introduction of "personalized" markets for public good - one market for each consumer. In the model, consumer expenditure encompasses both private and public "purchases", and consumer income encompasses both private and public "endowments". An individual is endowed with a quantity of her own version of the public good determined by the level of public expenditures. An increase in taxes, to the extent that it increases tax revenue, will increase the level of public provision.

In this model, the structure of relative factor taxes is exogenous but the aggregate level of taxes is not. Tax rates are scaled up or down so that the sum of individual valuations of the public good (the marginal benefit) equals the cost of supply of the public good (the direct marginal cost).

Consider features of model `SAMUELSON` which do not appear in `SHOVEN`:

1. There are new commodities `PG` and `VG(H)`. The first of these represents the direct marginal cost of public output from sector `GP`, a Leontief technology which converts private goods inputs into the public good. For the `SAMUELSON` structure, all government revenues apply to purchases of the public good (observe that the only good demanded by consumer `GOVT`is `PG`). The prices `VG(H)` represent the individual consumer valuations of the public good. Commodity `VG(H)` appears only in the endowments and demands of consumer `RA(H)`. The endowment record for `VG(H)` includes a quantity `V(H)` which is the benchmark valuation of the public good by agent `H`.
2. There are two auxiliary variables. `TAU` has the same interpretation as in the `SHOVEN`, determining the aggregate tax level. Auxiliary variable `LGP` is a rationing instrument representing an index of the "level of public goods provision", scale to equal 1 in the benchmark. Consumer `RA(H)` thus is endowed with a quantity of `VG(H)` given by `V(H) * LGP` .

3. The constraint for TAU in SAMUELSON differs from the TAU constraint in SHOVEN. Here the constraint represents the Samuelson condition, equating the marginal cost (PG*GREV) and the sum of individuals' marginal benefit (SUM(H,VG(H)*V(H))). The constraint for LGP simply assigns LGP equal to the sector GP activity level. (The LGP variable and constraint are only needed because the R: field only accepts auxiliary variables.)

TABLE 8: Endogenous Determination of Tax Revenue

```
MPSGE PREPROCESSOR VERSION 1/94   286/386/486 DOS

 0  $MODEL: SAMUELSON
 1
 2  $SECTORS:
 3        AL(S)  GP
 4
 5  $COMMODITIES:
 6        P(G)  W(F)  PG  VG(H)
 7
 8  $CONSUMERS:
 9        RA(H) GOVT
10
11  $AUXILIARY:
12        TAU  LGP
13
14  $REPORT:
15          V:CD(G,H)       D:P(G)          DEMAND:RA(H)
16          V:DF(F,H)       D:W(F)          DEMAND:RA(H)
17          V:EMPLOY(S)     I:W("L")        PROD:AL(S)
18          V:WLF(H)        W:RA(H)
19
20  $PROD:AL(S)  s:0  a:ELAS(S)
21        O:P(G)      Q:A(G,S)
22        I:P(G)      Q:B(G,S)
23        I:W(F)      Q:FD(F,S)   P:PF(F,S)
24  +       A:GOVT  N:TAU$TF(F,S)  M:TF(F,S)$TF(F,S) a:
25
26  $PROD:GP  s:0
27          O:PG    Q:GREV
28          I:P(G)  Q:GD(G)
29
30  $DEMAND:RA(H)  s:1  a:ESUB(H)
31        D:P(G)      Q:C(G,H)   a:
32        D:W(F)      Q:D(F,H)
33        D:VG(H)     Q:V(H)
34        E:VG(H)     Q:V(H)   R:LGP
35        E:W(F)      Q:E(F,H)
36
37  $DEMAND:GOVT
38        D:PG        Q:GREV
39
40  $CONSTRAINT:TAU
41          GREV * PG =G= SUM(H, V(H) * VG(H));
42
43  $CONSTRAINT:LGP
44                  LGP =G= GP;
45
46  $OFFTEXT
```

An Algebraic Formulation of Samuelson Model

Zero Net Profit for Private Production

The unit cost of production in sector $A_h s$ is given by a nested Leontief - Cobb-Douglas function defined over the cost of intermediate inputs and primary factors with ad-valorem taxes on factor demands. Unlike the Harberger models, tax rates in this model are determined endogenously. In equilibrium, the unit cost must be no less than the market price of output:

$$-\prod_s = \sum_g p_g B_{bs} \quad + \quad \phi_s \left(\prod_f (\omega_f (1 + \tau t_{fs}))^{\alpha f_s} \right) \quad - \quad p_s \geq 0 \quad \forall s$$

Zero Net Profit for Public Sector Contractors

The unit cost of public provision is determined by the market price of commodity inputs to the Leontief activity. Input requirements are defined by a vector of public sector input coefficients, $A_h g$. In equilibrium, the price paid by the government equals the cost of market inputs:

$$\sum_g p_g a_g \quad - \quad p_G \geq 0$$

Income Balance for Government

Government tax income (PT) is determined by the value of tax revenue, calculated using activity levels, compensated demands, market prices and ad-valorem tax rates. In equilibrium, the value of tax revenue equals the market cost of public sector output:

$$PT = \sum_s \frac{\partial \prod_s}{\partial(\omega_f (1+\tau t_{fs}))} \quad AL_s \omega_f \tau t_{fs} = p_G G$$

Income Balance for Households

Household income is determined by the net of tax return to primary factors plus the imputed value of public provision:

$$M_h = \sum_f \omega_f E_{fh} + v_h G$$

Market Clearance for Private Goods

Producer output is equal to the sum of intermediate plus final demand:

$$AL_g \geq \sum_s AL_s B_{gs} + \sum_h \frac{\gamma h M_h}{e_h(p)} \left(\frac{e_h(p)}{p_g} \right)^{\sigma_h}$$

where γ_h is the household budget share devoted to the consumption of goods, and $e_h h$ is the "unit expenditure function" which may be written:

$$e_h(p) \equiv \left(\sum_i \beta_{ih} p_i^{1-\sigma_h} \right)^{\frac{1}{1-\sigma_h}}$$

Personalized Markets for Public Goods

We assume a "pure" public good in this model, hence each household may attach a different marginal valuation to public provision in an equilibrium. In order to compute these marginal values, we include a separate public good "market" for each houshold which balances the level of provision with the household "demand":

$$G = \frac{\mu_{Gh} M_h}{v_h}$$

in which μ_{Gh} is the budget share of public goods in the top-level Cobb-Douglas preferences of household h.

Market Clearance for Factors

The aggregate supply of factors equals the sum of producer and consumer demand. Producers pay taxes on factor inputs, consumers do not because we consider these demands to be "leisure" or "household production". Consumer demands for factors are specified as Cobb-Douglas (constant budget shares):

$$\sum_h E_{fh} = \sum_s AL_s \frac{\partial \prod_s}{\partial(\omega_f(1+\tau t_{fs}))} + \sum_h \frac{\mu_{fh} M_h}{\omega_f}$$

Samuelson Rule for "Optimal" Provision of Public Goods

The tax rate multiplier is adjusted to balance the marginal cost of public provision with the summation across households of marginal willingness to pay. Due to the existence of household factor demand, factor taxes are necessarily distortionary and there will be an excess social cost of public funds. For this reason, the Samuelson rule is neither necessary nor sufficient for optimal provision. We apply the rule here merely to illustrate the programming methodology, even though the resulting equilibrium may be "suboptimal":

$$PG = \sum_h v_h$$

Comparing Model Results

Although the foregoing discussion has focused on the nuances of *MPSGE* model syntax, but there are many interesting economic questions which can be addressed using even small-scale models such as tho ones described here. Consider the output listing from parameter REPORT is displayed in Table 9. It is perhaps surprising to note that none of the uniform tax structures represents a Pareto-superior choice compared to the benchmark tax structure. Furthermore, from the standpoint of aggregate welfare (`"WELFARE.TOTAL"` = income-weighted sum of individual EV's), only the uniform capital tax represents an improvement.

Table 9: Numerical Results from Alternative Models

```
INDEX 1 = HARBERGER
                          K           L           VA
REVENUE._              3.9       -38.9        -0.8
TAXRATE._             50.0        50.0        25.0
WELFARE.OWNER          1.9        42.4        18.5
WELFARE.WORKER        -0.1       -26.8       -10.9
WELFARE.TOTAL          0.6        -1.3 -3.48143E-2
EMPLOY .X             -5.3        -6.9        -8.4
EMPLOY .Y             20.5        34.4        22.1
PRICE  .X            -10.4       -11.2       -10.3
PRICE  .Y             11.8        12.8        11.8
PRICE  .K              3.9        59.5        24.5
PRICE  .L             -4.7       -38.9       -23.5
OUTPUT .X              3.6        -1.0         0.4
OUTPUT .Y             -3.7         2.0        -2.0
```

```
INDEX 1 = SHOVEN
                          K           L           VA
TAXRATE._              47.1       134.2        25.3
WELFARE.OWNER           3.3        40.2        18.3
WELFARE.WORKER         -1.0       -29.2       -10.8
WELFARE.TOTAL           0.6        -3.6 -3.51710E-2
EMPLOY .X              -5.0       -19.7        -8.5
EMPLOY .Y              21.5        12.1        21.9
PRICE  .X             -10.4        -9.0       -10.3
PRICE  .Y              11.9        10.2        11.8
PRICE  .K               6.2        49.8        24.2
PRICE  .L              -5.0       -56.5       -23.6
OUTPUT .X               3.6        -7.9         0.3
OUTPUT .Y              -3.4        -2.0        -2.1

INDEX 1 = SAMUELSON
                            K           L           VA
REVENUE  ._              -1.4       -14.5        -6.7
TAXRATE  ._              45.7        88.8        22.8
WELFARE  .OWNER           4.7        43.9        21.1
WELFARE  .WORKER         -2.0       -31.4       -12.9
WELFARE  .TOTAL           0.5        -3.7        -0.4
EMPLOY   .X              -4.9        -7.5        -5.9
EMPLOY   .Y              24.5        37.5        29.7
PRICE    .X             -10.7       -11.3       -10.9
PRICE    .Y              12.2        13.0        12.5
PRICE    .K               7.8        60.3        29.0
PRICE    .L              -6.0       -51.8       -24.5
OUTPUT   .X               3.0        -2.2         0.9
OUTPUT   .Y              -2.3         3.3 -2.58148E-2
PROVISION._              -0.8       -13.9        -6.1
```

4.53.2.5 Summary

This paper has provided an introduction to a new *GAMS* subsystem for applied general modeling. This extension of *GAMS* accomodates a tabular representation of highly nonlinear cost and expenditure functions through which model specification is concise and transparent. The paper has presented three small examples which illustrate the programming environment and its application to traditional economic issues in public finance for which applied general equilibrium analysis is a standard tool. Further work is underway in the development and evaluation of solution algorithms for applied general equilibrium models implemented within *GAMS/MPSGE*. In addition to providing a convenient framework for model-builders, the new *GAMS/MPSGE* system also simplifies the implementation and testing of algorithms for complementarity problems. Information on the relative effectiveness of different solution strategies should prove quite helpful to users who are using the system to solve very large systems of nonlinear equations.

4.53.2.6 References

- Ballard and Fullerton (1992) "Distortionary Taxes and the Provision of Public Goods", *Journal of Economic Perspectives* 6(3).
- Brooke, T., D. Kendrick and A. Meeraus (1988) GAMS: A User's Guide, The Scientific Press, Redwood City, California.
- Rutherford, T. (1993) "MILES: A Mixed Inequality and nonLinear Equation Solver", Working Paper, Department of Economics, University of Colorado.

- Rutherford, T. (1987) "Applied General Equilibrium Modeling", Ph.D. thesis, Department of Operations Research, Stanford University.
- Rutherford, T.F. (1989) "General Equilibrium Modeling with <cite>MPSGE</cite>", The University of Western Ontario.
- Shoven, J. and J. Whalley (1984) "Applied General Equilibrium Models of Taxation and International Trade: Introduction and Survey", *Journal of Economic Literature* 22, 1007-1051.
- Thompson, G. and S. Thore (1992) *Computational Economics* , Scientific Press, Redwood City, California.

4.53.2.7 Appendix A: Language Syntax

General syntax rules

- All input is free format (spaces and tabs are ignored) except keywords for which "$" must appeatr in column 1.
- End-of-line is significant. Continuation lines are indicated by a "+" in column 1.
- In general, input is not case sensitive, except in the specification of sub-nests for production and demand functions.
- Numeric expression involving GAMS parameters or constants must be enclosed in parentheses.

Keywords

Keywords typically appear in the following order:

Keywords	Description
`$ONTEXT`	Indicate the beginning of a GAMS comment block containing an MPSGE model.
`$MODEL:model_name`	*model_name* must be a legitimate file name. This name is subsequently used to form `MODEL_NAME.GEN` (this file name must be upper case when running under UNIX).
`$SECTORS:`, `$COMMODITIES:`, `$AUXILIARY:`, `$CONSUMERS:`	Four keywords define variables which are used in the model. Entries in these blocks share the same syntax. The `$AUXILIARY` block is only used in models with side constraints and endogenous taxes or rationed endowments.
`$PROD:sector`	Production functions must be specified for each production sector in the model.
`$DEMAND:consumer`	Demand functions must be specified for every consumer in the model. General structure is the same as for production functions above.
`$CONSTRAINT:auxiliary`	Specifies a side constraint to be associated with a specified auxiliary variable.
`$REPORT:`	Identifies the set of additional variables to be calculated. These include outputs and inputs by sector and demands by individual consumers.
`$OFFTEXT`	Indicates the end of model specification.

Variable Declarations

There are four classes of variables within an *MPSGE* model: activity levels for production sectors, prices for commodities, income levels for consumers and level values for auxiliary variables. These classes of variables are distinguished in order to permit additional semantic checking by the *MPSGE* preprocessor. For example, if `P` has been declared as a price (within the `$COMMODITIES:` block), then the preprocessor would report an error if it encountered `"$PROD:P"`. |

The `$SECTORS:`, `$COMMODITIES:`, `$CONSUMERS:` and `$AUXILIARY:` blocks contain implicit *GAMS* variable declarations in which the index sets must be specified in the *GAMS* program above and the variable names must be distinct from all other symbols in the *GAMS* program. One or more variables may be declared per line separated by one or more spaces.

```
$SECTORS:
          Y(R,T)      ! Output in region R in period T
          K(T)        ! "Aggregate capital stock, period T"
```

In these declarations, the trailing comments (signified by "!") are interpreted as variable name descriptors which subsequently appear in the solution listing.

The equivalent *GAMS* declaration for these variables would be:

```
 VARIABLES   Y(R,T)  Output in region R in period T
             K(T)    "Aggregate capital stock, period T";
```

As with the usual *GAMS* syntax, when a variable descriptor contains a punctuation symbol such as ",", it is required to enclosed in quotes.

```
$SECTORS:
        X(R,T)$(X0(R) GT 0)
```

Here, the *GAMS* conditional operator "$" is used to restrict the domain of the variable X. The expression following the dollar sign is passed through to the *GAMS* compiler and must conform to *GAMS* syntax rules.

```
$SECTORS:
          X Y(R)$Y0(R) Z    !  Descriptor for Z
          W(G,R,T)          !  Descriptor for W
```

More than one symbol may appear on a single line. The descriptor only applies to the last one.

All *MPSGE* variables must be declared. When multidimensional variables are specified, they must be declared explicitly - declarations like `X(*)` are not permitted. Two further restrictions are that the sets used in the declaration must be static rather than dynamic, and any variable which is declared must be used in the model. There is a simple way to work around these restrictions. Let me illustrate with an example. Suppose that in a model the set of production sectors AL is employed for all elements of a static set S which satisfy a particular condition, for example `BMX(S)` not equal to 0. This would require that AL be declared as follows:

```
$SECTORS:
    AL(S)$BMX(S)
```

In this context, the symbol "$" is used as an "exception operator" which should be read as "such that". In this case, we have generated one AL sector for each element of the set S for which BMX(S) is nonzero.

Function Declarations

Functional declarations characterize nested CES functions which characterize both preferences and technology. The former are written within a $DEMAND: block and the latter within a $PROD: block. Tax entries may appear within a $PROD: block but not within a $DEMAND: block, otherwise the syntax is nearly identical. The syntax for these blocks will be described through a sequence of examples:

```
$PROD:Y(R)  s:1
O:P(R)      Q:YO(R)
I:W(F,R)    Q:FDO(F,R)
```

This block characterizes a Cobb-Douglas production function in which the elasticity of substitution between inputs is one - "s:1" in the first line which sets a top level substitution elasticity equal to unity. Variable Y(R) is an activity level declared within the $SECTORS: block. Variables P(R) and W(F,R) are prices declared within the $COMMODITIES: block. The O: label indicates an output, and the I: prefix indicates an input. The Q: fields in both records represent "reference quantities". YO(R) and FDO(F,R) must be GAMS parameters defined previously in the program.

```
$PROD:X(R)  s:ESUB(R)  a:0  b:(ESUB(R)*0.2)
O:PX        Q:XO(R)
I:PY(G)     Q:YXO(G,R)  a:
I:PL        Q:LXO(R)  b:
I:PK        Q:KXO(R)  b:
```

The keyword line specifies three separate elasticities related to this function. ESUB(R) is the top level elasticity of substitution. There are two sub-nests in the function. Nest a: is a Leontief nest (in which the compensated elasticity is zero). The elasticity of subtitution in nest b: is one-fifth of the top level elasticity.

In the function specification, commodities PY(G) (one input for each element of set G) enter in fixed proportions. Commodities PL and PK enter in nest b.

If this function has been specified using a balance benchmark dataset with reference prices equal to unity, then the following identity should be satisfied:

```
XO(R) = SUM(G, YXO(G,R)) + LXO(R) + KXO(R)
```

```
$PROD:AL(S)  s:0  a:ELAS(S)
O:P(G)    Q:A(G,S)
I:P(G)    Q:B(G,S)
I:W(F)    Q:FD(F,S)  P:PF(F,S)  A:GOVT  T:TF(F,S)  a:
```

In this function, we have two new ideas. The first is the use of a reference price denoted by "P:". This entry indicates that the function should be calibrate to a reference point where individual input prices (gross of tax) equal PF(F,S). If P: does not appear, prices of one are assumed.

The second new idea here is that taxes may be levied on production inputs. The A: label identifies the name of the tax agent (a $CONSUMER:). The T: label identifies the ad-valorem tax rate.

```
$DEMAND:RA(R)$RAO(R)   s:1
E:PL                   Q:LE(R)
D:P(G)$DG(G)           Q:DO(G,R)$DD(G,R) P:PO(G,R)
```

This function specification demonstrates the use of conditionals. This function is only generated when `RAO(R)` is nonzero. The demands D: for a particular element of set G are suppressed entirely when `DG(G)` equals 0. The `Q:` field also has an exception operator, so that the default value for `Q:` (unity) is applied when `DD(G,R)` equals zero.

This example is somewhat artificial, but it illustrates the distinction between how exception operators affect lead entries (`I:`, `O:`, `D:` and `E:`) and subsequent entries. When an exception is encountered on the lead entry, the entire record may be suppressed. Exceptions on subsequent entries only applied to a single field.

The valid labels in a function declaration (`$PROD:` or `$DEMAND:`) line include:

`s:` Top level elasticity of substitution between inputs or demands.

`t:` Elasticity of transformation between outputs in production. (valid only in `$PROD` blocks)

`a:,b:,...` Elasticities of substitution in individual input nests.

The recognized labels in an `I:` or `O:` line include:

`Q:` Reference quantity. Default value is 1. When specified, it must be the second entry.

`P:` Reference price. Default value is 1.

`A:` Tax revenue agent. Must be followed by a consumer name.

`T:` Tax rate field identifier. (More than one tax may apply to a single entry.)

`N:` Endogenous tax. This label must be followed by the name of an auxiliary variable.

`M:` Endogenous tax multiplier. The advalorem tax rate is the product of the value of the endogenous tax and this multiplier.

`a:,b:,..` Nesting assignments. Only one such label may appear per line.

The valid labels in an `E:` line include:

`Q:` Reference quantity

`R:` Rationing instrument indicating an auxiliary variable.

The valid labels in a `D:` line include:

`Q:` Reference quantity

`P:` Reference price

`a:,b:...` Nesting assignment

Constraints

Auxiliary constraints in *MPSGE* models conform to standard *GAMS* equation syntax. The may refer to any of the four classes of variables, `$SECTORS`, `$COMMODITIES`, `$CONSUMERS` and `$AUXILIARY`, but they may not reference variables names declared within a `$REPORT` block. Complementarity conditions apply to upper and lower bounds on auxiliary variables and the associated constraints. For this reason, the orientation of the equation is important. When an auxiliary variable is designated POSITIVE (the default), the auxiliary constraint should be expressed as a "greater or equal" inequality (`=G=`). If an auxiliary variable is designated FREE, the associated constraint must be expressed as an equality (`=E=`).

```
$CONSTRAINT:TAU
                  G =G= X * Y;
$CONSTRAINT:MU(I)$MU0(I)

            MU(I) * P(I) * Q(I) =G= SUM(J, THETA(I,J) * PX(J));
```

The exception applied in this example restricts the equation only to those elements of set `I` for which `MU0(I)` is not zero.

Report Declaration

The GAMS interface to MPSGE normally returns level values only for the central variables - those declared within `$SECTORS:`, `$COMMODITIES:`, `$CONSUMERS:` and `$AUXILIARY:` sections. An equilibrium determines not only these values, but also levels of demand and supply by individual sectors and consumers. Given benchmark information, elasticities and the equilibrium values, all such demands can be computed, but this can be tedious to do by hand. In order to have these values returned by MPSGE, it is necessary to indicate the name of the variable into which the value is to be returned. The general form is as follows:

```
$REPORT:
      V:variable name   I:commodity       PROD:sector
      V:variable name   O:commodity       PROD:sector
      V:variable name   D:commodity       DEMAND:consumer
      V:variable name   W:consumer
```

The first row returns an input quantity, the second row returns an output quantity, the third returns a demand quantity, and the fourth row returns a consumer welfare index. (Note: the level value returned for a "consumer variable" is an income level, not a welfare index.)

```
$REPORT:
      V:DL(S)                 I:PF("L")         PROD:Y(S)
      V:DK(S)                 I:PF("K")         PROD:Y(S)
      V:SX(G,S)$SX0(G,S)      O:PX(G)           PROD:X(S)
      V:D(G,H)                D:P(G)            DEMAND:RA(H)
      V:W(H)                  W:RA(H)
```

Note that the "$" exception is only meaningful on the first entry. Also notice that the domain of the report variable must conform to the domain of the subsequent two entries.

Differences between Scalar and Vector *MPSGE Syntax*

1. `$MODEL:` statement The `$MODEL` statement is required in the vector format and it must precede all other statements. A name which is an acceptable file name prefix must be used. The preprocessor does not begin translation of an*MPSGE* model until it encounters a `$MODEL` statement following an `$ONTEXT` record. The preprocessor continues to translate until it reaches an $OFFTEXT statement, skipping blank lines and comment lines identified by a "$*$" in column 1.

2. Case folding: In the vector syntax, upper and lower case letters are not distinguished. The entire file is processed as though it were written in upper case. This is not compatible with the earlier version of*MPSGE* in which "P" and "p" were distinct.

3. Distinct names: Names used for variables in the*MPSGE* model must be distinct from each other as well as from all other symbols in the *GAMS* program. If there is a *GAMS* set or parameter or model named X, then X may not be used to identify an*MPSGE* sector or commodity.

4. Tabs:*MPSGE* fields are free format and tabs are translated to spaces by the preprocessor. Tabs are permitted in *GAMS* provided that the compiler is properly configured (under DOS, "TABIN 8" must be inserted in file *GAMS*PARM.TXT in the *GAMS* system directory).

Exception Handling

The *GAMS* exception operator can be used on virtually any entry in the*MPSGE* input file. For example, if you want to have sector X(S) have one production structure for elements S in a subset T(S), you can provide separate production function declarations as follows:

```
$PROD:X(S)$T(S)
...         ! sector X described for S in T

$PROD:X(S)$(NOT T(S))
...         ! sector X described for S not in T.
```

The preprocessor does not require one and exactly one declaration for each sector. If multiple declarations appear, the later set of coefficients overwrites the earlier set.

Switches and Debug Output

Run-time tolerances and output switches may be specified within the vector-syntax model specification or using the PUT facility, they can be written directly to the MPS input file. Output switches control the level of debug output written by the*MPSGE* subsystem to the solver status file. Reports provided by $ECHOP, $FUNLOG and $DATECH can be returned to the listing file by specifying "OPTION SYSOUT=ON;" within the *GAMS* program prior to the SOLVE statement. The recognized*MPSGE* parameters are:

```
$ECHOP: logical         Default=.FALSE.
```

is a switch for returning all or part of the scalar*MPSGE* file to the solver status file. In order to have this output printed in the listing file, enter the *GAMS* statement "OPTION SYSOUT=ON;" prior to solving the model.

```
$PEPS:  real            Default=1.0E-6
```

is the smallest price for which price-responsive demand and supply functions are evaluated. If a price is below PEPS, it is perturbed (set equal to PEPS) prior to the evaluation.

```
$EULCHK:    logical     Default=.TRUE.
```

is a switch for evaluating Euler's identity for homogeneous equations. The output is useful for monitoring the numerical precision of a Jacobian evaluation. When a price or income level is perturbed in a function, the Euler check may fail.

```
$WALCHK:    logical     Default=.TRUE.
```

is a switch for checking Walras's law. Like EULCHK, this switch is provided primarily to monitor numerical precision. When an income level is perturbed, the Walras check may fail.

```
$FUNLOG:    logical     Default=.FALSE.
```

is a switch to generate a detailed listing of function evaluations for all production sectors and consumers.

FUNLOG triggers a function evaluation report which provides detailed output describing the evaluation of supply and demand coefficients. The information provide is sufficient that an industrious graduate student should be able to reproduce the results (given a pencil, paper and slide rule).

The evaluation report has the following headings:

Heading	Description
T	Coefficient "type" with the following interpretation: IA Input aggregate OA Output aggregate I Input O Output D Demand E Endowment
N	Name (either nest identifier or commodity name)
PBAR	Benchmark price (the P: field value)
P	Current price (gross of tax)
QBAR	Benchmark quantitity (the Q: field value)
Q	Current quantity
KP	Identifier for parent entry in nesting structure.
ELAS	Associated elasticity (input or output aggregates only)

When $FUNLOG:.TRUE is specified, a complete report of demand and supply coefficients for every production and demand function in every iteration. Be warned that with large models this can produce an enrmous amount of output!

The following two function evaluation reports are generated in the first iteration in solving case "L" for model HARBERGER:

```
Function Evaluation for: AL.X

 T   N     PBAR         P            QBAR         Q            KP     ELAS
 ------------------------------------------------------------------------
 IA  s     1.0000E+00   8.9198E-01   1.0000E+02   1.0000E+02          0.00
 OA  t     1.0000E+00   1.0000E+00   1.0000E+02   1.0000E+02          0.00
 IA  a     1.0000E+00   8.7998E-01   9.0000E+01   9.0000E+01   s      1.00
 O   P.X   1.0000E+00   1.0000E+00   1.0000E+02   1.0000E+02   t
 I   P.Y   1.0000E+00   1.0000E+00   1.0000E+01   1.0000E+01   s
 I   W.K   2.0000E+00   1.5000E+00   2.0000E+01   2.3466E+01   a
 I   W.L   1.0000E+00   1.0000E+00   5.0000E+01   4.3999E+01   a

 Function evaluation for: RA.OWNER

 T   N       PBAR        P             QBAR        Q          KP     ELAS
 ------------------------------------------------------------------------
 IA  s     1.0000E+00   1.0000E+00   7.0000E+01   7.0000E+01          1.00
 OA  t     1.0000E+00   0.0000E+00   0.0000E+00   0.0000E+00          0.00
 IA  a     1.0000E+00   1.0000E+00   7.0000E+01   7.0000E+01   s      0.50
 D   P.X   1.0000E+00   1.0000E+00   3.0000E+01   3.0000E+01   a
 D   P.Y   1.0000E+00   1.0000E+00   4.0000E+01   4.0000E+01   a
 E   W.K   1.0000E+00   1.0000E+00   6.0000E+01   0.0000E+00
 E   PT    1.0000E+00   1.0000E+00   1.0000E+01   0.0000E+00


$DATECH:    logical     Default=.FALSE.
```

is a switch to generate a annotated listing of the function and Jacobian evaluation including a complete listing of all the nonzero coefficients.

MPSGE generates an analytic full first-order Taylor series expansion of the nonlinear equations in every iteration. Nonzero elements of the Jacobian matrix are passed to the system solver (*MILES* or *PATH*) which uses this information in the direction-finding step of the Newton algorithm. Coefficients are produced with codes which help interpret where they came from. The following codes are used:

- `W0` indicates an element from the orthogonal part of F().
- `W1` indicates an element from the non-orthogonal part of F().
- `B` indicates a linear term from F.
- `E0` indicates a homogeneous Jacobian entry.
- `E1` indicates a non-homogeneous Jacobian entry.

The Euler checksum examines elements from the linearization which are type `"E0"`. The Walras check sum examines elements from the function evaluation which are type `"W0"`.

Needless to say, the `$DATECH:.TRUE.` switch produces a very big status file for large models. It is not something which is very useful for the casual user.

Here is a partial listing of nonzeros generated during the first linearization for scenario `"L"` in model `HARBERGER:`

```
---------------------------------------- Coefficients for sector:AL.X
P.X             AL.X              1.0000E+02    B
AL.X            P.X              -1.0000E+02    B
P.Y             AL.X             -1.0000E+01    B
AL.X            P.Y               1.0000E+01    B
W.K             AL.X             -2.3466E+01    B
AL.X            W.K               3.5199E+01    B
W.L             AL.X             -4.3999E+01    B
AL.X            W.L               4.3999E+01    B
W.K             W.K               1.3037E+01   E0      1.3037E+01
W.K             W.L              -1.3037E+01   E0     -1.3037E+01
W.L             W.K              -1.9555E+01   E0     -1.9555E+01
W.L             W.L               1.9555E+01   E0      1.9555E+01
GOVT            AL.X             -1.1733E+01    B
GOVT            W.K              -1.1733E+01   E1
GOVT            W.K               6.5184E+00   E0      6.5184E+00
GOVT            W.L              -6.5184E+00   E0     -6.5184E+00

---------------------------------------- Income for consumer:RA.OWNER
W.K                               6.0000E+01   W0
RA.OWNER        W.K              -6.0000E+01    B
PT                                1.0000E+01   W0
RA.OWNER        PT               -1.0000E+01    B
RA.OWNER        RA.OWNER          1.0000E+00    B

---------------------------------------- Demands for consumer:RA.OWNER
P.X                              -3.0000E+01   W0
P.Y                              -4.0000E+01   W0
P.X             P.X               8.5714E+00   E0      8.5714E+00
P.X             P.X               1.2857E+01   E0      1.2857E+01
P.X             P.Y              -8.5714E+00   E0     -8.5714E+00
P.X             P.Y               1.7143E+01   E0      1.7143E+01
P.Y             P.X              -8.5714E+00   E0     -8.5714E+00
P.Y             P.X               1.7143E+01   E0      1.7143E+01
P.Y             P.Y               8.5714E+00   E0      8.5714E+00
P.Y             P.Y               2.2857E+01   E0      2.2857E+01
P.X             RA.OWNER         -4.2857E-01   E0     -3.0000E+01
P.Y             RA.OWNER         -5.7143E-01   E0     -4.0000E+01
```

4.53.2.8 Appendix B: File Structure

This appendix provides an overview of the structure of *GAMS* input files which include*MPSGE* models. The text of the paper presents many of these ideas by way of example, but it may also be helpful for some users to have a "template" for constructing*MPSGE* models. The discussion in this section focuses on a "generic" input file, the schematic form of which is presented in Table 10. This section first presents a "top down" view of program organization, and then it discusses aspects of the new syntax for model specification.

Flow of Control

When a model is developed using *GAMS* as a front-end to*MPSGE*, the input file generally has five sections as identified in Table 10. Section (i), the benchmarking section, contains standard *GAMS* statements. This includes *GAMS* `SET` declarations, input data (`SCALARS`, `PARAMETERS` and `TABLES`), and `PARAMETER` declarations for intermediate arrays used in benchmarking or model specification. In complex models, this section of the file will typically contain some algebraic derivations, the result of which is a calibrated benchmark equilibrium dataset.

Users who are unfamiliar with *GAMS* can consult the manual. Beginning *GAMS* programmers should remember that the*MPSGE* interface to *GAMS* is unlike other solution subsystems. "Level values" are passed between the *GAMS* program and*MPSGE* in the usual fashion, but*MPSGE* models do not require the explicit use of the `VARIABLE` or `EQUATION` statements.)

The second section of the file consists of a *GAMS* comment range, beginning with an $ONTEXT record and ending with an $OFFTEXT record, followed by an invocation of the preprocessor. The preprocessor writes operates on statements in the*MPSGE* model declaration which are "invisible" to the *GAMS* compiler. This program reads the*MPSGE* model statements and generates *GAMS*-readable code, including a model_name.gen file. Additional *GAMS* code produced by the preprocessor includes declarations for each of the central variables and report variables in the*MPSGE* model.

The third section of the generic input file performs a "benchmark replication" and may not be present in all applications. There are four statements required for benchmark validation. The first statement sets the iteration limit to be zero; the second statement causes the*MPSGE* model to be "generated", and the third statement causes the*MPSGE* solver to read the model and return the deviations. In this call, the level values passed to the solver are unaltered because the iteration limit is zero. Market excess supplies and zero profit checks are returned in the "marginals" of the associated commodity prices and activity levels, respectively. The final statement in this section resets the iteration limit to 1000 (the default value) for subsequent counter-factual computations.

Section (iv) defines and then computes a counter-factual equilibrium. A counter-factual equilibrium is defined by parameter values such as tax rates or endowments which take on values different from those in the benchmark equilibrium. Within the *GAMS* interface to*MPSGE*, it is also possible to fix one or more central variables. When any variable is fixed, the associated equation is omitted from the equilibrium system during the solution process but the resulting imbalance is then reported in the solution returned through the marginal.

The final section of the file represents the *GAMS* algebra required for comparing counter-factual equilibria. It would be possible, for example, to construct welfare measures or to report percentage changes in certain values. All of these calculations are quite easy because the equilibrium values are returned as level values in the associated variables.

The final section of the file represents the *GAMS* algebra required for comparing counter-factual equilibria. It would be possible, for example, to construct welfare measures or to report percentage changes in certain values. All of these calculations are quite easy because the equilibrium values are returned as level values in the associated variables.

For large models, the advantage of the vector format is that by using appropriately defined *GAMS* sets, the number of individual functions which need to be defined is reduced only to the number of "classes" of functions. This makes it possible to represent large dimensional models using only a few lines of code.

To summarize, here are the basic features of a program which uses *GAMS* as a front-end to *MPSGE*:

1. An *MPSGE* model is defined within a *GAMS* comment range followed by

   ```
   $sysinclude mpsgeset model_name
   ```

2. Every `SOLVE` statement for a particular model is preceded by `$INCLUDE MODEL.GEN`. The GEN file is written by the preprocessor based on the model structure.

3. Solution values for the cental variables in the *MPSGE* model and any declared "report variables" are returned in *GAMS* variable level values. Level values for slacks are returned as "marginals" for the associated variables.

4. The model description follows a format which is a direct extension of the scalar data format. Certain aspects of the new language, such as case folding, are incompatible with the original *MPSGE* syntax.

GAMS Code Generated by the Preprocessor: the GEN File

Most novice users will find it easiest to treat the preprocessor output files as "black boxes". These files contain *GAMS* source code required for declaring and generating the *MPSGE* input file. Table 11 contains portions of the GEN file for the same model. Table 12 shows the preprocessor-generated listing and symbol table which are always appended to the bottom of the GEN file. If a preprocessor error occurs, it can be helpful to consult the symbol table to track down the bug. Finally, Table 13 shows the first page of scalar format *MPSGE* input file produced by HARBERGER.GEN. Normally, this file is written and then erased in the course of a *GAMS* run, although all or part of the file may be retained using the `$ECHOP:` switch.

4.53.3 Demand Theory and General Equilibrium: An Intermediate Level Introduction to MPSGE

This research was supported by the *GAMS* Applied General Equilibrium Research Fund. The author remains responsible for any bugs which exist in this software.

Author

Thomas F. Rutherford, rutherford@colorado.edu Department of Economics University of Colorado

Date

1995

4.53.3.1 An Overview

This document describes a mathematical programming system for general equilibrium analysis named *MPSGE* which operates as a subsystem to the mathematical programming language *GAMS*. *MPSGE* is library of function and Jacobian evaluation routines which facilitates the formulation and analysis of AGE models. *MPSGE* simplifies the modeling process and makes AGE modeling accessible to any economist who is interested in the application of these models. In addition to solving specific modeling problems, the system serves a didactic role as a structured framework in which to think about general equilibrium systems.

MPSGE separates the tasks of model formulation and model solution, thereby freeing model builders from the tedious task of writing model-specific function evaluation subroutines. All features of a particular model are communicated to *GAMS*/*MPSGE* through a tabular input format. To use *MPSGE*, a user must learn the syntax and conventions of this model definition language.

The present paper is intended for students who have completed two semesters of study in microeconomics. The purpose of this presentation is to give students a practical perspective on microeconomic theory. The diligent student who works through all of the examples provided here should be capable of building small models "from scratch" to illustrate basic theory. This is a first step to acquiring a set of useable tools for applied work.

The remainder of this paper is organized as follows. Section The Theory of Consumer Demand review the theory of consumer demand; section Getting Started provides guidance on how to verify that the *GAMS/MPSGE* software is operational; section Modeling Consumer Demand introduces the modeling framework with three models illustrating the representation of consumer demand within the *MPSGE* language. Section The Pure Exchange Model reviews the pure exchange model, and section Modeling Pure Exchange with MPSGE presents two *MPSGE* models of exchange. Each of the model-oriented sections present exercises based on the models which give students a chance to work through the material on their own. Additional introductory examples for self-study can be found in the `Markusen library` as well as in the **GAMS Model Library** (look for models with names ending in "MGE").

The level of presentation and diagrammatic exposition adopted here is based on Hal Varian's undergraduate microeconomics textbook (*Intermediate Microeconomics: A Modern Approach*, Third Edition, W. W. Norton & Company, Inc., 1993).

The ultimate objective of this piece is to remind students of some theory which they have already seen and illustrate how these ideas can be used to build numerical models using *GAMS* with *MPSGE*. It is not my intention to provide a graduate level presentation of this material. So far as possible, I have avoided calculus and even algebra. The objective here is to demonstrate that what matters are economic ideas. With the proper tools, it is possible to do concrete economic modeling without a lot of mathematical formalism.

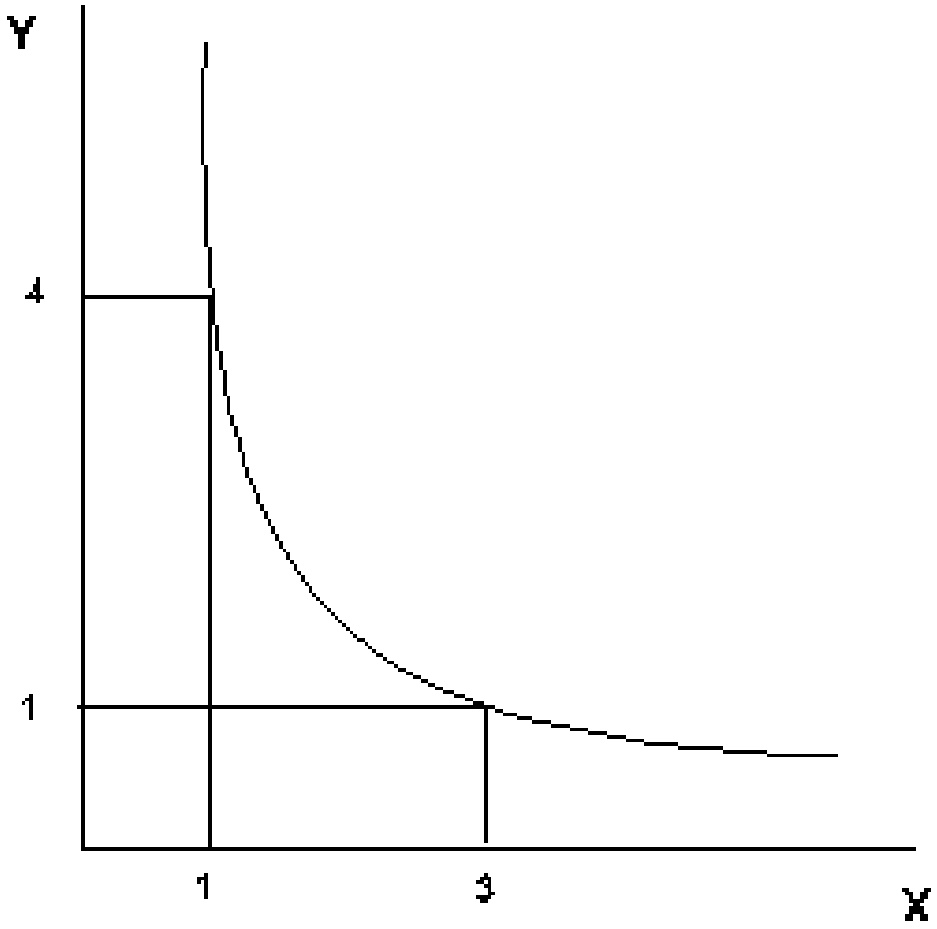
4.53.3.2 The Theory of Consumer Demand

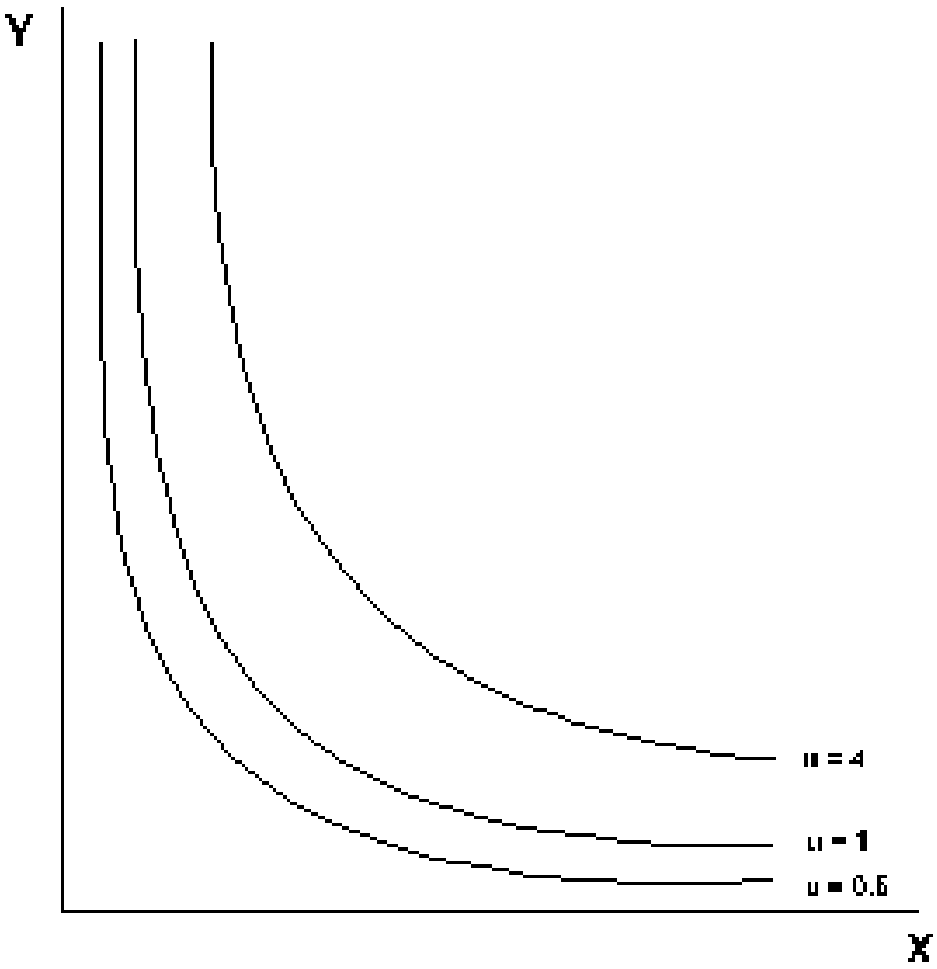
A central idea underlying most microeconomic theory is that agents optimize subject to constraints. The optimizing principle applied to consumer choice begins from the notion that agents have preferences over consumption bundles and will always choose the most preferred bundle subject to applicable constraints. To operationalize this theory, three issues which must be addressed:

1. How can we represent preferences?
2. What is the nature of constraints on consumer choice? and
3. How can the choice be modelled?

Preferences are relationships between alternative consumption "bundles". These can be represented graphically using "indifference curves", as illustrated in Figure 1. Focusing now on the preferences of a single consumer, the indifference curve is a line which connects all combinations of two goods `x` and `y` between which our consumer is indifferent. As this curve is drawn, we have represented an agent with "well-behaved preferences": at any allocation, more is better (monotonicity), and averages are preferred to extremes (convexity). Exactly one such indifference curve goes through each positive combination of `x` and `y`. Higher indifference curves lie to the "north-east".

Figure 1: An Indifference Curve


Y
4
1
1
3
X



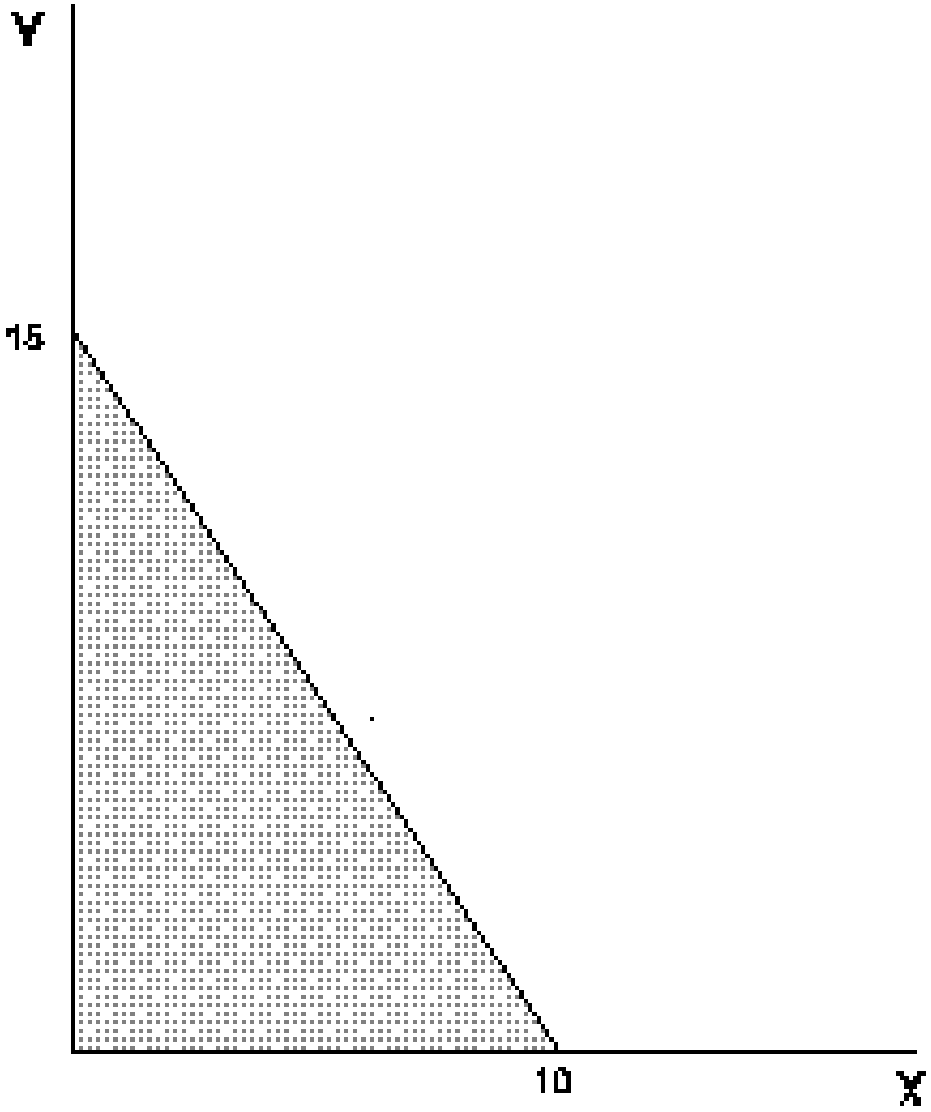
If we wish to characterize an agent's preferences, the "marginal rate of substitution" (MRS) is a useful point of reference. At a given combination of `x` and `y`, the marginal rate of substitution is the slope of the associated indifference curve. As drawn, the MRS increases in magnitude as we move to the northwest and the MRS decreases as we move to the south east. The intuitive understanding is that the MRS measures the willingness of the consumer to trade off one good for the other. As the consumer has greater amounts of `x`, she will be willing to trade more units of `x` for each additional unit of `y` – this results from convexity.

An "ordinal" utility function $U(x, y)$ provides a helpful tool for representing preferences. This is a function which associates a number with each indifference curve. These numbers increase as we move to the northeast, with each successive indifference curve representing bundles which are preferred over the last. The particular number assigned to an indifference curve has no intrinsic meaning. All we know is that if $U(x1, y1) > U(x2, y2)$, then the consumer prefers bundle 1 to bundle 2.

Figure 2 illustrates how it is possible to use a utility function to generate a diagram with the associated indifference curves. This figure illustrates Cobb-Douglas well-behaved preferences which are commonly employed in applied work.

Figure 2: Indifference Curves and Utility Levels

Up to this point, we have we have focused exclusively on the characterization of preferences. Let us now consider the other side of the consumer model – the budget constraint. The simplest approach to



characterizing consumer income is to assume that the consumer has a fixed money income which she may spend on any goods. The only constraint on this choice is that the value of the expenditure may not exceed the money income. This is the standard budget constraint:

$$Px\ x + Py\ y = M.$$

Graphically, this equation defines the line depicted in Figure 3. All points inside the budget line are affordable. The consumer faces a choice of which affordable bundle to select.

Figure 3: The Budget Set

Within the framework of our theory, the answer is straightforward – the consumer will choose the one combination of x and y from the set of affordable bundles which maximizes her utility. This combination of x and y will be at the point where the indifference curve is tangent the budget line. This point is called the optimal choice. We see this illustrated in Figure 4.

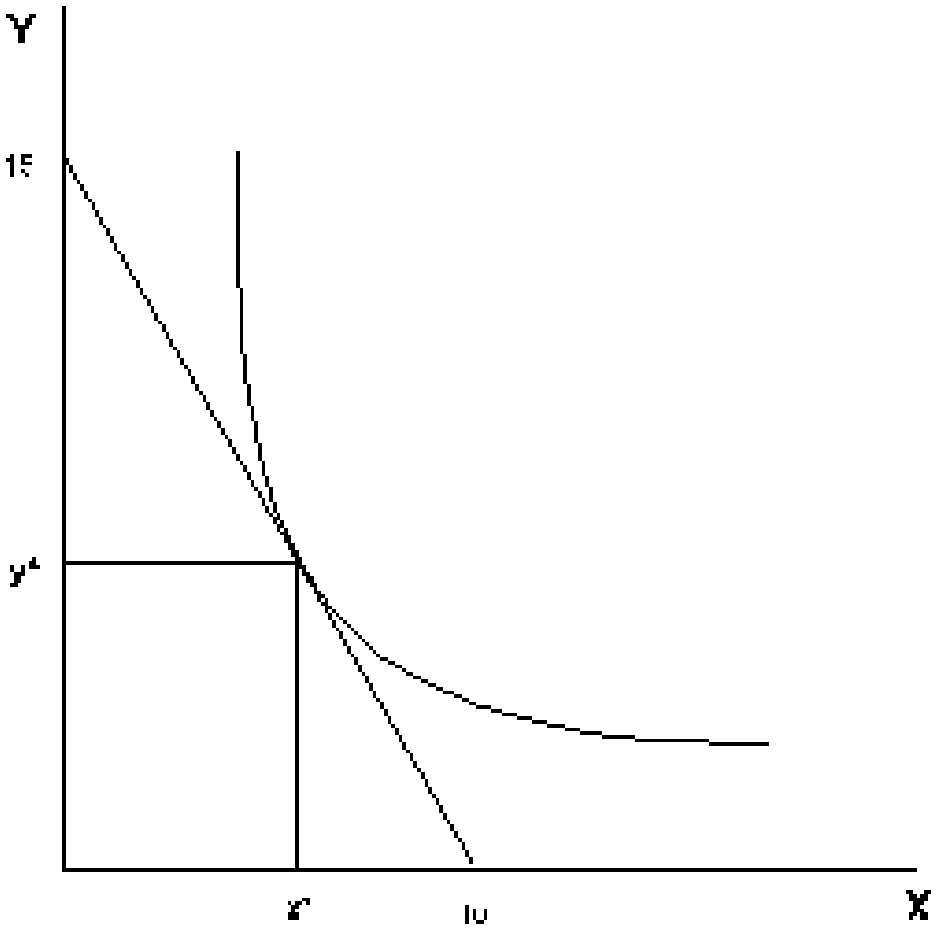

Y
15
y*
x*
10
X

Figure 4: Utility Maximization on the Budget Set

The standard model of consumer behavior provides a starting point for learning *MPSGE*. This introduction is "hands on" – I will discuss issues as they arise, assuming that you have access to a computer and can invoke the program and read the output file. You may wish to learn the rudiments of *GAMS* syntax before starting out, although you may be able to pick up these ideas as they come, depending on your aptitude for computer languages.

4.53.3.3 Getting Started

For running MPSGE models, the GAMS/BASE module is required along with a suitable MCP solver such as MILES or PATH. To verify that the system is operational, run one of the *MPSGE* library models, e.g. SCARFMGE. This can be done easily via GAMS Studio (search for *SCARFMGE* in the Model Library Explorer). Alternatively, from a command prompt, connect to a working directory (never run models from the *GAMS* system directory!). Then, extract and run the model:

```
C:\>MKDIR WORK

C:\>CD WORK

C:\WORK>GAMSLIB SCARFMGE

C:\WORK>GAMS SCARFMGE
```

If the *GAMS* system is properly installed, these commands will cause *GAMS* to solve a sequence of models from the `SCARFMGE` sample problem. The output from this process is written to file `SCARFMGE.LST` . If you search for the word "STATUS", you can verify that all the cases are processed.
There are a number of *MPSGE* models included in the **GAMS Model Library**. If you are using a *GAMS* demo license, you will be able to process most but not all of the library models. The demo license limits the number of variables in the model to 1000.
Assuming that you have successfully installed the software, let us now proceed to some examples which illustrate both the computing syntax and the underlying economics.

4.53.3.4 Modeling Consumer Demand

Example 1: Evaluating a Demand Function

Consider a standard consumer choice problem, one which might appear on a midterm examination in intermediate microeconomics:

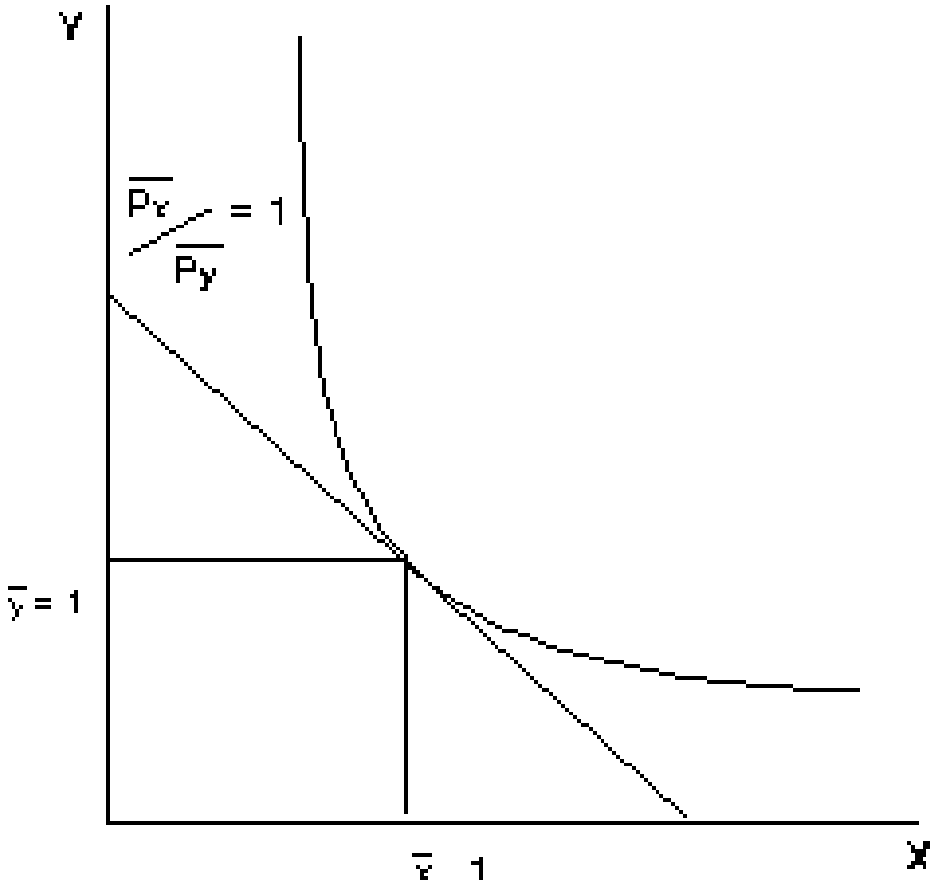
$$\begin{aligned} &\max U(x,y) = \ln(x) + 2\ln(y) \\ &\text{subject to:} \\ &1x + 2y = 120 \end{aligned}$$

where 1 is the exogenous price of `x` and `2` is the price of `y`.

This type of problem is solved easily using GAMS/MINOS (as a nonlinear program). Strictly speaking, it is not the sort of model for which you would need to use *MPSGE*. At the same time, this can be an instructive example.

The key issue in this example is learning how to represent utility functions. *MPSGE* is "non-algebraic" – so function specification depends on an intuitive understanding of the underlying economic structure.

Consider Figure 5 and focus on a single point, $x = 1$, $y = 1$. There is an indifference curve through this point, and the marginal rate of substitution (MRS) at this point is simply the slope of this curve. The benchmark MRS does not uniquely determine the underlying preferences.


Y
$\frac{\overline{P_x}}{\overline{P_y}} = 1$
$\bar{y} = 1$
$\bar{x}$ 1
X

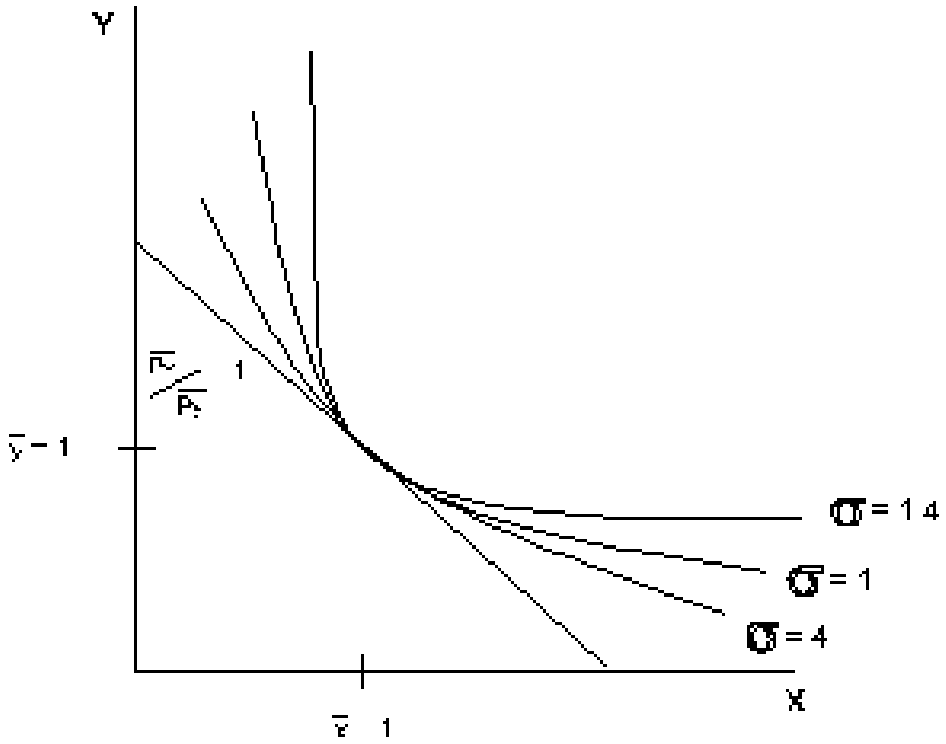


Figure 5: A Calibrated Benchmark

A utility function is represented in *MPSGE* by the specification of: (i) benchmark demand quantities, (ii) benchmark demand prices (iii) an elasticity of substitution at the benchmark point. Benchmark quantities determine an anchor point for the set of indifference curves. Benchmark prices fix the slope of the indifference curve at that point, and the elasticity describes the curvature of the indifference curve. Speaking formally, elasticities provide a "second order approximation" of the utility function. To understand the importance of the benchmark elasticity of substitution, consider Figure 6. This figure shows three indifference curves all of which share the same benchmark quantities and benchmark prices. They differ only in the elasticities of substitution. The least convex (flattest) curve has the highest elasticity, the most convex curve has the lowest elasticity. (When the elasticity of substitution is 0, the indifference curve is L-shaped with the corner at the benchmark point.)

Figure 6:The Elasticity of Substitution

Let us now consider how the consumer optimization problem can be cast as a general equilibrium model. We do this by adding a single factor of production and two "production" sectors. For concreteness, let the factor of production be called labor with a price PL. One production function converts one unit of labor into one unit of `x`, the other sector converts 2 units of labor into one unit of`y`. Setting the labor endowment equal 120, the market clearance condition for labor reads:

$$1x + 2y = 120$$

which is precisely the budget constraint for the consumer's problem.

We will now present the program code, a few lines at a time. As part of working through the example, the student should type these lines into a file.

A *MPSGE* model specification is always listed between `$ONTEXT` and `$OFFTEXT` statements. The first statement within an *MPSGE* model-description assigns a name to the model. The model name must begin with a letter and must have 10 or fewer characters.

```
$ONTEXT

$MODEL:DEMAND
```

The model specification begins by declaring variables for the model. In a standard model, there are three types of variables: commodity prices, sectoral activity levels, and consumer incomes. The end of each line may include "! variable description ".

N.B. The variables associated with commodities are prices, not quantities. (In this and subsequent models, I use P as the first letter for each of the commodity variables to remind us that these variables are prices.)

N.B. The variable associated with a consumer is an income level, not a welfare index.

```
$SECTORS:
        X       ! ACTIVITY LEVEL FOR X = DEMAND FOR GOOD X
        Y       ! ACTIVITY LEVEL FOR Y = DEMAND FOR GOOD Y

$COMMODITIES:
        PX      ! PRICE OF X WHICH WILL EQUAL PL
        PY      ! PRICE OF Y WHICH WILL EQUAL 2 PL
        PL      ! PRICE OF THE ARTIFICIAL FACTOR L

$CONSUMERS:
        RA      ! REPRESENTATIVE AGENT INCOME
```

Function specifications follow the variable declarations. In this model, our first declarations correspond to the two production sectors. In this model, the production structures are particularly simple. Each of the sectors has one input and one output. In the *MPSGE* syntax, `I:` denotes an input and `O:` denotes an output. The output quantity coefficients for both sectors are unity `(Q:1)`. This means that the level values for `x` and `y` correspond to the actual quantities produced. The final function specified in the model represents the utility function and endowments for our single consumer. In this function, the `E:` entries correspond to endowments and the `D:` entries are demands. Reference demands, reference prices and the substitution elasticity `(s:1)` characterize preferences.

The demand entries shown here are consistent with a Cobb-Douglas utility function in which the budget share for y is twice the budget share for x (i.e. the MRS at `(1,1)` equals `1/2`):

```
$PROD:X
      O:PX  Q:1
      I:PL  Q:1

$PROD:Y
      O:PY  Q:1
      I:PL  Q:2

$DEMAND:RA  s:1
      E:PL  Q:120
      D:PX  Q:1   P:(1/2)
      D:PY  Q:1   P:1

$OFFTEXT
```

The final three statements in this file invoke the *MPSGE* preprocessor, "generate" and solve the model:

```
$SYSINCLUDE mpsgeset DEMAND

$INCLUDE DEMAND.GEN
SOLVE DEMAND USING MCP;
```

The preprocessor invocation (`$SYSINCLUDE mpsgeset`) should be placed immediately following the `$OFFTEXT` block containing the model description. The model generator code, `DEMAND.GEN`, is produced by the previous statement and must be referenced immediately before each subsequent SOLVE statement.

At this point, the reader should take the time to type the example into a file and execute the program with GAMS/*MPSGE*.

This is possibly the first GAMS model which some readers have solved, so it is worth looking through the listing file in some detail. After running the solver, we examine the listing file. I typically begin my assessment of a model's solution by searching for "STATUS". For this model, we have the following:

```
               S O L V E      S U M M A R Y

     MODEL   DEMAND
     TYPE    MCP
     SOLVER  PATH                FROM LINE  263

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL

 RESOURCE USAGE, LIMIT          1.432     1000.000
 ITERATION COUNT, LIMIT         5         1000
 EVALUATION ERRORS              0            0

 Work space allocated           --    4.86 Mb

Default price normalization using income for RA
```

This information is largely self-explanatory. The most important items are the `SOLVER STATUS` and `MODEL STATUS` indicators. When the solver status is 1 and the model status is 1, the system has returned an equilibrium.

For small models such as this, the limits on resource usage (time) and solver iterations have no effect. (You can modify these values with the statements:

```
 model.RESLIM  =  number of cpu seconds ;

 model.ITERLIM =  number of iterations ;
```

entered into the program before the SOLVE statement.)

The work space allocation for *MPSGE* models is determined by the number of variables in the model. It is possible to exogenously specify the work space allocation by assigning

```
model.WORKSPACE = xx ;
```

where `xx` is the desired number of megabytes.

The final message, `"Default price normalization..."`, is significant. It reminds the user that an Arrow-Debreu general equilibrium model determines only relative prices. In such an equilibrium, the absolute scaling of prices is indeterminant. (I.e., if `(p*,M*)` are a set of equilibrium prices and income levels, then `(2 p*,2 M*)` is also a solution, etc.)

It is common practice in economics to address the normalization issue through the specification of a numeraire commodity. You can do this for an *MPSGE* model by "fixing" a price, with a statement like:

```
PX.FX = 1;
```

entered following the model declaration (`$SYSINCLUDE mpsgeset`) but prior to the solver invocation. When any price or income level is fixed, *MPSGE* recognizes that a numeraire has been specified and does no automatic normalization.

Following some output from the solver (`PATH` in this case), the listing file provides a complete report of equilibrium values. With *MPSGE* models, the equation listings are superfluous. The variable listings provide all the relevant information.

For this model, the solution listing appears as follows:

```
                          LOWER     LEVEL     UPPER    MARGINAL
---- VAR X                  .       40.000     +INF       .
---- VAR Y                  .       40.000     +INF       .
---- VAR PX                 .        1.000     +INF       .
---- VAR PY                 .        2.000     +INF       .
---- VAR PL                 .        1.000     +INF       .
---- VAR RA                 .      120.000     +INF       .

  X           SUPPLY AND DEMAND OF GOOD X
  Y           SUPPLY AND DEMAND OF GOOD Y
  PX          PRICE OF X WHICH WILL EQUAL CX * PL
  PY          PRICE OF Y WHICH WILL EQUAL CY * PL
  PL          PRICE OF THE ARTIFICIAL FACTOR L
  RA          REPRESENTATIVE AGENT INCOME
```

The LOWER and UPPER columns report variable bounds applied in the model. In these columns, zero is represented by "." and infinity is represented by `"+INF"`. The LEVEL column reports the solution value returned by the algorithm. Here we see that the equilibrium price of `x` is 1 and the price of `y` is 2, as determined by the specification of labor inputs.

Exercises 1

(a) The utility function calibration point is arbitrary. Here, we have selected $x = y = 1$ as the reference quantity. Revise the program to use a different calibration point where $x = 2$ and $y = 1$, where $MRS(2, 1) = 1/4$. (Remember to modify both the $Q:$ and $P:$ fields.) Rerun the model to demonstrate that this does not change the result.

(b) Increase the price of x from 1 to 2 by changing the `Q:` coefficient for `PL` in sector X from 1 to 2. What happens to the demand for x? Explain why a change in the price of x is represented by a change in the `Q:` field for sector X.

(c) Compute an equilibrium in which commodity y is defined as the numeraire.

Example 2: Evaluating the MRS

This example further explores the representation of demand functions with *MPSGE*. It sets up a trivial equilibrium model with two goods and one consumer which returns the marginal rate of substitution of good x for good y at a given level of demand. The underlying utility function is:

$$U(x, y) = \ln(x) + 4\ln(y)$$

When $x = y = 1$, the marginal rate of substitution of x for y is $1/4$. We use this information to calibrate the demand function, specifying the ratio of the reference prices of x to y equal to $1/4$.

In an equilibrium, final demand always equals endowments for both goods, because these are the only sources of demand and supply. The model as parameterized demonstrates that if we set endowments for this model equal to the demand function calibration point, the model equilibrium price ratio equals the benchmark MRS.

This program begins with some GAMS statements in which three scalar parameters are declared. These parameters will be used in the place of numbers within the *MPSGE* model. The syntax for these GAMS statements is introduced in Chapter 2 of the *GAMS* manual. In short, we declare x, y and MRS as scalar parameters and initialize the first two of these to unity. The MRS parameter is assigned a value following the solution of the model.

```
SCALAR
      X     QUANTITY OF X FOR WHICH THE MRS IS TO BE EVALUATED /1/
      Y     QUANTITY OF Y FOR WHICH THE MRS IS TO BE EVALUATED /1/
      MRS   COMPUTED MARGINAL RATE OF SUBSTITUTION;
The remainder of the <em>MPSGE</em> program is, in fact, simpler than Example 1.

$ONTEXT

$MODEL:MRSCAL

$COMMODITIES:
      PX    ! PRICE INDEX FOR GOOD X
      PY    ! PRICE INDEX FOR GOOD Y

$CONSUMERS:
      RA    ! REPRESENTATIVE AGENT

$DEMAND:RA  s:1
      D:PX  Q:1   P:(1/4)
      D:PY  Q:1   P:1
      E:PX  Q:X
      E:PY  Q:Y
$OFFTEXT
$SYSINCLUDE mpsgeset MRSCAL

$INCLUDE MRSCAL.GEN
SOLVE MRSCAL USING MCP;
```

Following the solution, we compute a function of the solution values, the ratio of the price of x to the price of y. We do this using the GAMS syntax which references the equilibrium level values of the `PX` and `PY` and storing this result in the scalar MRS. This scalar value is then displayed in the listing file with 8 digits:

```
MRS = PX.L / PY.L;
OPTION MRS:8;
DISPLAY MRS;
```

Exercises 2

(a) Show that the demand function is homothetic by uniform scaling of the x and y endowments. The resulting MRS should remain unchanged.

(b)Modify the demand function calibration point so that the reference prices of both x and y equal unity (hint: the marginal rate of substitution is:

$$MRS = x/(4y).$$

Example 3: Leisure Demand and Labor Supply

This model investigates the labor-leisure decision. A single consumer is endowed with labor which is either supplied to the market or "repurchased" as leisure. The consumer utility function over market goods (x and y) and leisure is Cobb-Douglas:

$$U(x, y, L) = \ln(x) + \ln(y) + \ln(L)$$

Goods x and y may only be purchased using funds obtained from labor sales. This constraint is written:

$$x + y = \text{LPROD LS}$$

where goods x and y both have a price of unity at base year productivity and `LPROD` is an index of labor productivity. An increase in productivity is equivalent to a proportional decrease in the prices of x and y.

The model declaration is as follows:

```
SCALAR          LPROD   AGGREGATE LABOR PRODUCTIVITY /1/,
                CX      COST OF X AT BASE YEAR PRODUCTIVITY /1/,
                CY      COST OF Y AT BASE YEAR PRODUCTIVITY /1/;
$ONTEXT

$MODEL:LSUPPLY

$SECTORS:
        X       ! SUPPLY=DEMAND FOR X
        Y       ! SUPPLY=DEMAND FOR Y
        LS      ! LABOR SUPPLY

$COMMODITIES:
        PX      ! MARKET PRICE OF GOOD X
        PY      ! MARKET PRICE OF GOOD Y
        PL      ! MARKET WAGE
        PLS     ! CONSUMER VALUE OF LEISURE

$CONSUMERS:
        RA      ! REPRESENTATIVE AGENT

$PROD:LS
        O:PL    Q:LPROD
        I:PLS   Q:1
```

```
$PROD:X
        O:PX    Q:1
        I:PL    Q:CX

$PROD:Y
        O:PY    Q:1
        I:PL    Q:CY

$DEMAND:RA  s:1
        E:PLS   Q:120
        D:PLS   Q:1     P:1
        D:PX    Q:1     P:1
        D:PY    Q:1     P:1

$OFFTEXT
$SYSINCLUDE mpsgeset LSUPPLY

$INCLUDE LSUPPLY.GEN
SOLVE LSUPPLY USING MCP;
```

We can use this model to evaluate the wage elasticity of labor supply. In the initial equilibrium (computed in the last statement) the demands for x, y and L all equal 40. A subsequent assignment to `LPROD` (below) increases labor productivity. After computing a new equilibrium, we can use the change in labor supply to determine the wage elasticity of labor supply, an important parameter in labor market studies. It should be emphasized that the elasticity of labor supply should be an input rather than an output of a general equilibrium model – this is a parameter for which econometric estimates can be obtained.

Here is how the programming works. First, we declare some scalar parameters which we will use for reporting, then save the "benchmark" labor supply in `LS0`:

```
SCALAR
      LS0   REFERENCE LEVEL OF LABOR SUPPLY
      ELS   ELASTICITY OF LABOR SUPPLY WRT REAL WAGE;

LS0 = LS.L;
```

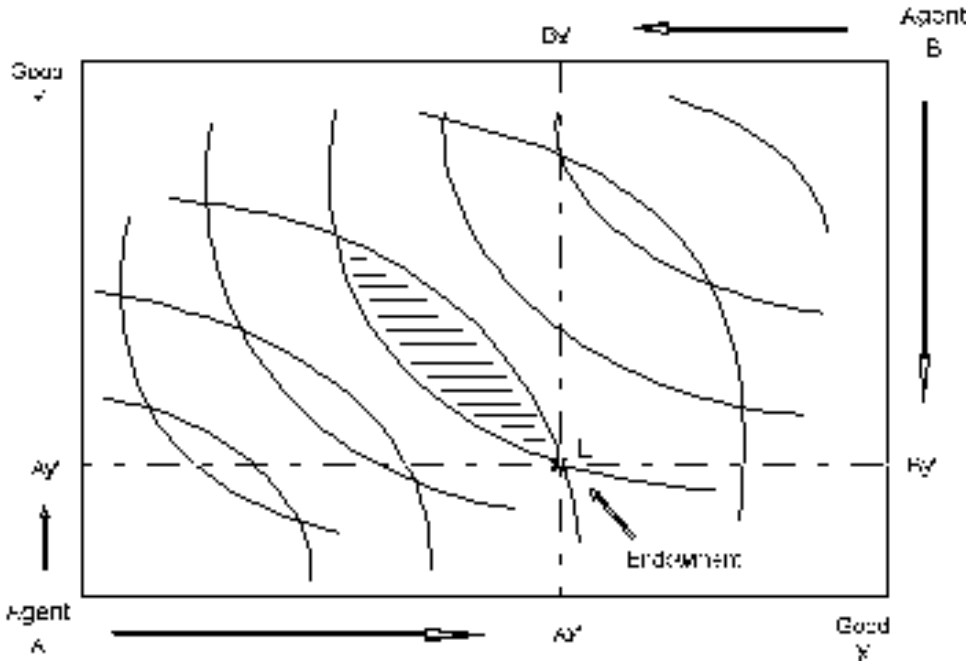
Next, we modify the value of scalar `LPROD`, increasing labor productivity by 1%. Because this is a neoclassical model, this change is equivalent to increasing the real wage by 1%. We need to recompute equilibrium prices after having changed the LPROD value:

```
LPROD = 1.01;
$INCLUDE LSUPPLY.GEN
SOLVE LSUPPLY USING MCP;
```

We use this solution to compute and report the elasticity of labor supply as the percentage change in the LS activity:

```
ELS = 100 * (LS.L - LS0) / LS0;
DISPLAY ELS;
```

As the model is currently constructed, the wage elasticity of labor supply equals zero. This is because the utility function is Cobb-Douglas over goods and leisure, and the consumer's only source of income is labor. As the real wage rises, this increases both the demand for goods (labor supply) and the demand for leisure. These effect exactly balance out and the supply of labor is unchanged.



Exercises 3

(a) One way in which the labor supply elasticity might differ from zero in a model with Cobb-Douglas final demand is if there were income from some other source. Let the consumer be endowed with good x in addition to labor. What x endowment is consistent with a labor supply elasticity equal to 0.15?

(b) A second way to "calibrate" the labor supply elasticity is to change the utility function. We can do this by changing the `s:1` to `s:SIGMA`, where `SIGMA` is a scalar value representing the benchmark elasticity of substitution between x, y and L in final demand. Modify the program to include SIGMA as a scalar, and find the value for `SIGMA` consistent with a labor supply elasticity equal to 0.15.

4.53.3.5 The Pure Exchange Model

Partial equilibrium analysis forms the basis of most economics courses at the undergraduate level. In these models we focus on price, supply and demand for a single commodity. The partial equilibrium approach neglects indirect effects, through which changes in the market for one good may influence the market for another good.

In the previous section, we focused on the choices of a single consumer. In the present section, we will explore the implications of interactions between many consumers with heterogeneous preferences. Furthermore, the analysis will explore the potentially important interaction between market prices and income which are determined jointly in a general equilibrium.

The most widely-used graphical framework for multi-agent exchange equilibrium analysis is the Edgeworth-Bowley box as illustrated in Figure 7. In this diagram we model the following economy:

Figure 7: The Edgeworth-Bowley Box

- Two types of consumers, denoted A and B. We consider A and B to each represent multiple consumers, each with the same endowments and preferences. This assumption is helpful for justifying our assumption of perfectly competitive, price-taking behavior.

- Two commodities, denoted x and y
- Fixed endowments of both goods. The horizontal axis measures the total world endowment of good X. The vertical axis measure the total world endowment of good Y. Any point in the box then represents an allocation of goods between the two agents. The agent H allocation is measured with respect to the lower left origin. The agent F allocation is measured with respect to the upper right origin.

Each agent has a given initial endowment, here denoted point E. Furthermore, we assume that there is no possibility for trade. The indifference curves through point E therefore represent autarchy welfare levels.

The key idea in this model is that trade can improve both agents' welfare. One agent gives some amount good x to the other in return for an amount of good y. The "terms of trade", the rate of exchange between x and y, is determined by the model. The model illustrates a number of important properties of market economies:

- Trade is mutually beneficial. So long as the transactions are voluntary, neither H nor F will be hurt by engaging in trade.
- Market prices can be used to guide the economy to a Pareto-efficient allocation, a state of affairs in which further mutually-beneficial trades are not possible.
- There is no guarantee that the gains from trade will be "fairly distributed" across consumers. A competitive equilibrium may produce a significant welfare increase for one consumer while have negligible impact on the other.
- There are multiple Pareto-efficient allocations, typically only one of which is a competitive equilibrium. We can use this model to demonstrate that the issues of efficiency and equity can be separated when there is the possibility of lump-sum income transfers between agents.

4.53.3.6 Modeling Pure Exchange with MPSGE

Example 4: A 2x2 Exchange Model

In this program, we examine the simple two good, two agent model of exchange equilibrium. The world endowments for goods x and y are both equal to 1. Six parameters are used to parameterize the model. These are declared as scalars at the top of the program:

```
SCALAR  XA              AGENT A ENDOWMENT OF X ( 0 < XA < 1)    /0.2/
        YA              AGENT A ENDOWMENT OF Y ( 0 < YA < 1)    /0.8/
        THETA_A         AGENT A DEMAND SHARE PARAMETER FOR X    /0.5/
        THETA_B         AGENT B DEMAND SHARE PARAMETER FOR X    /0.8/
        SIGMA_A         AGENT A ELASTICITY PARAMETER            /2.0/
        SIGMA_B         AGENT B ELASTICITY PARAMETER            /0.5/;
```

This model is actually simpler than the models presented above because we have no need for production. There are simply two commodities and two consumers. The consumers differ in terms of commodity endowments and preferences. The competitive equilibrium prices are such that supply equals demand for both goods and both agents spend an amount equal to their endowment income.

This model illustrates how to use computed function coefficients. See, for example, `Q:(1-THETA_A)` in the `$DEMAND:A` block. Any numeric input field in an MPSGE model may be "computed", provided that the algebraic expression is enclosed within parentheses and legitimate *GAMS* code.

This model specification uses the default values for reference prices in the demand function blocks. When P:value is not specified in a `D:`,`I:` or `O:` record, `P:1` is assumed.

This model uses the more general constant-elasticity-of-substitution utility function.

```
$ONTEXT

$MODEL:EXCHANGE

$COMMODITIES:
      PX    ! EXCHANGE PRICE OF GOOD X
      PY    ! EXCHANGE PRICE OF GOOD Y

$CONSUMERS:
      A     ! CONSUMER A
      B     ! CONSUMER B

$DEMAND:A  s:SIGMA_A
      E:PX  Q:XA
      E:PY  Q:YA
      D:PX  Q:THETA_A
      D:PY  Q:(1-THETA_A)

$DEMAND:B  s:SIGMA_B
      E:PX  Q:(1-XA)
      E:PY  Q:(1-YA)
      D:PX  Q:THETA_B
      D:PY  Q:(1-THETA_B)

$OFFTEXT
$SYSINCLUDE mpsgeset EXCHANGE

$INCLUDE EXCHANGE.GEN
SOLVE EXCHANGE USING MCP;

SCALAR
      PRATIO  EQUILIBRIUM PRICE OF X IN TERMS OF Y,
      IRATIO  EQUILIBRIUM RATIO OF CONSUMER A INCOME TO CONSUMER B INCOME;

PRATIO = PX.L / PY.L;
IRATIO = A.L  / B.L;

DISPLAY IRATIO, PRATIO;
```

The foregoing sets up the model and computes the competitive equilibrium. After *GAMS* returns from the solver, we declare and compute some report values.

Absolute levels of income and price returned from a general equilibrium model are not meaningful because a model determines only relative prices. For this reason, we report equilibrium income and price levels in relative terms.

In the final step, we compute an alternative efficient equilibrium, one in which the income levels for A and B are equal. The purpose of this exercise is to demonstrate the second welfare theorem. When incomes are both fixed, the equilibrium remains efficient, but the connection between market prices and endowment income is eliminated.

In *GAMS/MPSGE*, a variable may be fixed using the *GAMS* syntax

```
variable.fx= value;
```

as illustrated in this model:

```
A.FX = 1;
B.FX = 1;
$INCLUDE EXCHANGE.GEN
SOLVE EXCHANGE USING MCP;

SCALAR      TRANSFER  IMPLIED TRANSFER FROM A TO B AS A PERCENTAGE OF INCOME;

TRANSFER = 100 * ( A.L - PX.L * XA - PY.L * YA ); PRATIO = PX.L / PY.L;
IRATIO = A.L  / B.L;

DISPLAY TRANSFER, PRATIO, IRATIO;
```

Exercises 4

(a) Set up a separate models which computes the autarchy price ratios for consumers A and B. (You can use one of the earlier models as a starting point.)

(b) Determine parameter values for which the endowment point is the equilibrium point.

(c) Set up a series of computations from which you can sketch the efficiency locus. Draw the Edgeworth box diagram which is consistent with these values.

Example 5: Import Tariffs and Market Power

The exchange model provides a remarkably useful tool for analyzing issues related to international trade. Formal trade theory is more complicated with the inclusion of separate production technologies. We will present some of those models below. Before going forward, however, we will consider a slight generalization of the 2x2 model exchange model. In this extension, we introduce independent markets for consumers A and B and trade activities which deliver goods from one market to the other.

The set of input parameters largely the same as in the previous example. Two new parameters are ad-valorem tariffs which apply on imports to each of the regions.

```
SCALAR  XA              AGENT A ENDOWMENT OF X ( 0 le XA le 1)    /0.2/
        YA              AGENT A ENDOWMENT OF Y ( 0 le YA le 1)    /0.8/
        THETA_A         AGENT A DEMAND SHARE PARAMETER FOR X    /0.4/
        THETA_B         AGENT B DEMAND SHARE PARAMETER FOR X    /0.6/
        SIGMA_A         AGENT A ELASTICITY PARAMETER            /1.0/
        SIGMA_B         AGENT B ELASTICITY PARAMETER            /1.0/,
        T_A             AD-VALOREM TARIFF ON IMPORTS TO AGENT A /0.10/
        T_B             AD-VALOREM TARIFF ON IMPORTS TO AGENT B /0.10/;
```

The program differs from the previous example in several respects. First, we introduce a separate commodity price for each agent. In the absence of tariffs, these prices are identical.

A second difference is that in this model trade activities deliver goods from one agent to the other. These are denoted `M{good}{agent}` for imports of `{good}` to `{agent}`. There are four flows which may be operated in only one direction (the activity levels are non-negative). In terms of initial endowments and preferences, this model has exactly the same economic structure as the previous model.

```
$ONTEXT

$MODEL:TARIFFS

$SECTORS:
        MXA     ! TRADE IN X FROM B TO A
        MXB     ! TRADE IN X FROM A TO B
        MYA     ! TRADE IN Y FROM B TO A
        MYB     ! TRADE IN Y FROM A TO B


$COMMODITIES:
        PXA     ! PRICE OF GOOD X FOR AGENT A
        PYA     ! PRICE OF GOOD Y FOR AGENT A
        PXB     ! PRICE OF GOOD X FOR AGENT B
        PYB     ! PRICE OF GOOD Y FOR AGENT B

$CONSUMERS:
        A       ! CONSUMER A
        B       ! CONSUMER B

$DEMAND:A  s:SIGMA_A
        E:PXA   Q:XA
        E:PYA   Q:YA
        D:PXA   Q:THETA_A
        D:PYA   Q:(1-THETA_A)

$DEMAND:B  s:SIGMA_B
        E:PXB   Q:(1-XA)
        E:PYB   Q:(1-YA)
        D:PXB   Q:THETA_B
        D:PYB   Q:(1-THETA_B)
```

The trade activities each have one input and one output. They simply deliver a good (X or Y) from one agent to the other. The new syntax presented here is specification of an ad-valorem tax. Adding a tax requires two new fields. The first is `"A:"` which specifies the tax agent, a consumer who collects the tax revenue as part of income. The second is `"T:"` which specifies the ad- valorem tax rate.

MPSGE permits taxes to applied on production inputs and outputs but it does not permit taxes on final demand.

The tax applies on a net basis on inputs. For example, if we consider the MXA sector, the price of one unit of input is given by:

$$Px\_B * (1 + Ta)$$

where Px_B is the net of tax price of a unit of x in the agent B market and Ta is the ad-valorem tariff rate.

```
$PROD:MXA
      O:PXA Q:1
      I:PXB Q:1   A:A  T:T_A

$PROD:MXB
      O:PXB Q:1
      I:PXA Q:1   A:B  T:T_B
```

```
$PROD:MYA
      O:PYA Q:1
      I:PYB Q:1   A:A  T:T_A

$PROD:MYB
      O:PYB Q:1
      I:PYA Q:1   A:B  T:T_B
```

The final portions of the file introduces one use of "MPSGE report variables". In this case, report variables are used to recover a Hicksian money-metric welfare index for each of the agents. We compute the initial, tariff-ridden equilibrium in order to compute the benchmark welfare levels. We then set all tariffs to zero and compute the free-trade equilibrium. Using the final welfare indices and the saved values of the benchmark welfare levels, we are able to report the change in welfare from removing tariff distortions.

```
$REPORT:
      V:WA  W:A
      V:WB  W:B

$OFFTEXT
$SYSINCLUDE mpsgeset TARIFFS

$INCLUDE TARIFFS.GEN
SOLVE TARIFFS USING MCP;

SCALAR
      WA0   BENCHMARK WELFARE INDEX FOR AGENT A
      WB0   BENCHMARK WELFARE INDEX FOR AGENT B;

WA0 = WA.L;
WB0 = WB.L;
T_A = 0;
T_B = 0;

$INCLUDE TARIFFS.GEN
SOLVE TARIFFS USING MCP;

SCALAR
      EVA   HICKSIAN EQUIVALENT VARIATION FOR AGENT A
      EVB   HICKSIAN EQUIVALENT VARIATION FOR AGENT B;

EVA = 100 * (WA.L-WA0)/WA0;
EVB = 100 * (WB.L-WB0)/WB0;
DISPLAY EVA, EVB;
```

Exercises 5

(a) Find the "optimal tariff" in this model for agent A, assuming that agent B does not retaliate and leaves her tariff rate at the benchmark level.

(b) Insert the endowment and preference parameters from the previous problem, retaining the same "benchmark" tariff rates. Does free trade benefit both countries? If not, why not?

4.53.4 Constant Elasticity of Substitution Functions: Some Hints and Useful Formulae

Notes prepared for GAMS General Equilibrium Workshop held December, 1995 in Boulder Colorado.

Author

Thomas F. Rutherford, Department of Economics, University of Colorado

Date

December, 1995

4.53.4.1 The Basics

In many economic textbooks the constant elasticity of substitution (CES) utility function is defined as:

$$U(x,y) = (\alpha x^\rho + (1-\alpha)y^\rho)^{\frac{1}{\rho}},$$

It is a fairly routine but tedious calculus excercise to demonstrate that the associated demand functions are:

$$x(p_x,p_y,M) = \left(\frac{\alpha}{p_x}\right)^\sigma = \frac{M}{\alpha^\sigma p_x^{1-\sigma} + (1-\alpha)^\sigma p_y^{1-\sigma}},$$

and

$$y(p_x,p_y,M) = \left(\frac{1-\alpha}{p_y}\right)^\sigma \frac{M}{\alpha^\sigma p_x^{1-\sigma} + (1-\alpha)^\sigma p_y^{1-\sigma}},$$

The corresponding indirect utility function has is:

$$V(p_x,p_y,M) = M\left(\alpha^\sigma p_x^{1-\sigma} + (1-\alpha)^\sigma p_y^{1-\sigma}\right)^{\frac{1}{\sigma-1}},$$

Note that U(x,y) is linearly homogeneous:

$$U(\lambda_x,\lambda_y) = \lambda U(x,y)$$

This is a convenient cardinalization of utility, because percentage changes in U are equivalent to percentage Hicksian equivalent variations in income.

Because U is linearly homogeneous, V is homogeneous of degree one in M and degree -1 in p.

In the representation of technology, we have an analogous set of relationships, based on the cost and compensated demand functions. If we have a CES production function of the form:

$$y(K,L) = \gamma\left(\alpha K^\rho + (1-\alpha)L^\rho\right)^{\frac{1}{\rho}},$$

the unit cost function then has the form:

$$c(p_K, p_L) = (\frac{1}{\gamma})\left(\alpha^\sigma p_K^{1-\sigma} + (1-\alpha)^\sigma p_L^{1-\sigma}\right)^{\frac{1}{1-\sigma}},$$

and associated demand functions are:

$$K(p_K, p_L) = \left(\frac{y}{\gamma}\right)\left(\frac{\alpha\gamma c(p_K, p_L)}{p_K}\right)^\sigma,$$

and

$$L(p_K, p_L) = \left(\frac{y}{\gamma}\right)\left(\frac{(1-\alpha)\gamma c(p_K, p_L)}{p_L}\right)^\sigma.$$

In most large-scale applied general equilibrium models, we have many function parameters to specify with relative ly few observations. The conventional approach is to *calibrate* functional parameters to a single benchmark equilibrium. For example, if we have benchmark estimates for output, labor, capital inputs and factor prices , we calibrate function coefficients by inverting the factor demand functions:

$$\theta = \frac{\bar{p}_k \bar{K}}{\bar{p}_k \bar{K} + \bar{P}_L \bar{L}}, \quad \rho = \frac{\sigma - 1}{\sigma}, \quad \alpha = \theta \bar{K}^{\frac{-1}{\rho}},$$

and

$$\gamma = \bar{y}\left[\alpha \bar{K}^\rho + (1-\alpha)\bar{L}^\rho\right]^{\frac{-1}{\rho}}.$$

4.53.4.2 The Calibrated Share Form

Calibration formulae for CES functions are messy and difficult to remember. Consequently, the specification of function coefficients is complicated and error-prone. For applied work using calibrated functions, it is much easier to use the "calibrated share form" of the CES function. In the calibrated form, the cost and demand functions explicitly incorporate

- benchmark factor demands
- benchmark factor prices
- the elasticity of substitution
- benchmark cost
- benchmark output
- benchmark value shares

In this form, the production function is written:

$$y = \bar{y}\left[\theta\left(\frac{K}{\bar{K}}\right)^\rho + (1-\theta)\left(\frac{L}{\bar{L}}\right)^\rho\right]^{\frac{1}{\rho}}$$

The only calibrated parameter, θ , represents the value share of capital at the benchmark point. The corresponding cost functions in the calibrated form is written:

$$c(p_K, p_L) = \bar{c}\left[\theta\left(\frac{p_K}{\bar{p}_k}\right)^{1-\sigma} + (1-\theta)\left(\frac{p_L}{\bar{p}_L}\right)^{1-\sigma}\right]^{\frac{1}{1-\sigma}}$$

where $\bar{c} = \bar{p}_L\bar{L} + \bar{p}_K\bar{K}$and the compensated demand functions are:

$$K(p_K, p_L, y) = \bar{K}\frac{y}{\bar{y}}\left(\frac{\bar{p}_K}{p_K}\frac{c}{\bar{c}}\right)^{\sigma}$$

and

$$L(p_K, p_L, y) = \bar{L}\frac{y}{\bar{y}}\left(\frac{c}{\bar{c}}\frac{\bar{p}_L}{p_K}\right)^{\sigma}.$$

Normalizing the benchmark utility index to unity, the utility function in calibrated share form is written:

$$U(x, y) = \left[\theta\left(\frac{x}{\bar{x}}\right)^{\rho} + (1-\theta)\left(\frac{y}{\bar{y}}^{\rho}\right)\right]^{\frac{1}{\rho}}$$

Defining the unit expenditure function as:

$$e(p_x, p_y) = \left[\theta\left(\frac{p_x}{\bar{p}_x}\right)^{1-\sigma} + (1-\theta)\left(\frac{p_y}{\bar{p}_y}\right)\right]^{\frac{1}{1-\sigma}}$$

the indirect utility function is:

$$V(p_x, p_y, M) = \frac{M}{\bar{M}e(p_x, p_y)}$$

and the demand functions are:

$$x(p_x, p_y, M) = \bar{x}\ V(p_x, p_y, M)\left(\frac{e(p_x, p_y)\bar{p}_x}{p_x}\right)^{\sigma}$$

and

$$y(p_x, p_y, M) = \bar{y}\ V(p_x, p_y, M)\left(\frac{e(p_x, p_y)\bar{p}_y}{p_y}\right)^{\sigma}$$

The calibrated form extends directly to the n-factor case. An n-factor production function is written:

$$y = f(x) = \bar{y}\left[\sum_i \theta_i\left(\frac{x_i}{\bar{x}_i}\right)^{\rho}\right]^{\frac{1}{\rho}}$$

and has unit cost function:

$$C(p) = \bar{C}\left[\sum_i \theta_i\left(\frac{p_i}{\bar{p}_i}\right)^{1-\sigma}\right]^{\frac{1}{1-\sigma}}$$

and compensated factor demands:

$$x_i = \bar{x}_i\frac{y}{\bar{y}}\left(\frac{C\bar{p}_i}{\bar{C}p_i}\right)^{\sigma}$$

4.53.4.3 Excercises

(i) Show that given a generic CES utility function:

$$U(x,y) = (\alpha^\rho + (1-\alpha)y^\rho)^{\frac{1}{\rho}}$$

can be represented in share form using:

$$\bar{x} = 1,\ \bar{y} = 1,\ \bar{p}_x = t\alpha,\ \ \bar{p}_y = t(1-\alpha),\ \bar{M} = t.$$

for any value of $t > 0$.

(ii) Consider the utility function defined:

$$U(x,y) = (x-a)^\alpha (y-b)^{1-\alpha}$$

A benchmark demand point with both prices equal and demand for y equal to twice the demand for x. Find values for which are consistent with optimal choice at the benchmark. Select these parameters so that the income elasticity of demand for x at the benchmark point equals 1.1.

(iii) Consider the utility function:

$$U(x,L) = (\alpha L^\rho + (1-\alpha)x^\rho)^{\frac{1}{\rho}}$$

which is maximized subject to the budget constraint:

$$p_x x = M + \omega(\bar{L} - L)$$

in which M is interpreted as non-wage income, ω is the market wage rate. Assume a benchmark equilibrium in which prices for x and L are equal, demands for x and L are equal, and non-wage income equals one-half of expenditure on x. Find values of α and ρ consistent with these choices and for which the price elasticity of labor supply equals 0.2.

(iv) Consider a consumer with CES preferences over two goods. A price change makes the benchmark consumption bundle unaffordable, yet the consumer is indifferent. Graph the choice. Find an equation which determines the elasticity of substitution as a function of the benchmark value shares. (You can write down the equation, but it cannot be solved in closed form.)

(v) Consider a model with three commodities, x, y, and z. Preferences are CES. Benchmark demands and prices are equal for all goods. Find demands for x, y and z for a doubling in the price of x as a function of the elasticity of substitution.

(iv) Consider the same model in the immediately preceeding question, except assume that preferences are instead given by:

$$U(x,y,z) = (\beta \min(x,y)^\rho + (1-\beta)z^\rho)^{\frac{1}{\rho}}$$

Determine $\beta$from the benchmark, and find demands for x, y and z if the price of x doubles.

4.53.4.4 Flexibility and Non-Separable CES functions

We let π_i denote the user price of the ith input, and let $x_i(\pi)$ be the cost-minizing demand for the ith input. The reference price and quantities are $\bar{\pi}_i$ and $\bar{x}_i$. One can think of set i as $\{K, L, E, M\}$ but the methods we employ may be applied to any number of inputs. Define the reference cost, and reference value share for ith input by $\bar{C}$ and θ_i , where

$$\bar{C} \equiv \sum_i \bar{\pi}_i \bar{x}_i$$

and

$$\theta_i \equiv \frac{\pi_i \bar{x}_i}{\bar{C}}.$$

The single-level constant elasticity of substitution cost function in "calibrated share form" is written:

$$C(\pi) = \bar{C}\left(\sum_i \theta_i \left(\frac{\pi_i}{\bar{\pi}_i}\right)^{1-\sigma} \right)^{\frac{1}{1-\sigma}}$$

Compensated demands may be obtained from Shephard's lemma:

$$x_i(\pi) = \frac{\delta C}{\delta \pi_i} \equiv C_i = \bar{x}_i \left(\frac{C(\pi)}{\bar{C}} \frac{\bar{\pi}_i}{\pi_i} \right)^{\sigma}$$

Cross-price Allen-Uzawa elasticities of substitution (AUES) are defined as:

$$\sigma_{ij} \equiv \frac{C_{ij} C}{C_i C_j}$$

where

$$C_{ij} = \frac{\delta^2 C(\pi)}{\delta \pi_i \, \delta \pi_j} = \frac{\delta x_i}{\delta \pi_j} = \frac{\delta x_j}{\delta \pi_i}$$

For single-level CES functions:

$$\sigma_{ij} = \sigma \quad \forall_{i \neq j} \quad .$$

The CES cost function exibits homogeneity of degree one, hence Euler's condition applies to the second derivatives of the cost function (the Slutsky matrix):

$$\sum_j C_{ij}(\pi)(\pi_j) = 0$$

or, equivalently:

$$\sum_j \sigma_{ij}(\theta_j) = 0$$

The Euler condition provides a simple formula for the diagonal AUES values:

$$\sigma_{ii} = \frac{-\sum_{j \neq i} \sigma_{ij} \theta_j}{\theta_i}$$

As an aside, note that convexity of the cost function implies that all minors of order 1 are negative, i.e. $\sigma_{ii} < 0 \; \forall_i$. Hence, there must be at least one positive off-diagonal element in each row of the AUES or Slutsky matrices. When there are only two factors, then the off-diagonals must be negative. When there are three factors, then only one pair of negative goods may be complements.

Let:

k be the reference the index of second-level nest

s_{ik} denote the fraction of good i inputs assigned to the kth nest

ω_k denote the benchmark value share of total cost which enters through the kth nest

γ denote the top-level elasticity of substitution

σ^k denote the elasticity of substitution in the kth aggregate

$p_k(\pi)$ denote the price index associated with aggregate k, normalized to equal unity in the benchmark, i.e.:

$$p_k(\pi) = \left[\frac{\sum_i s_{ik}\theta_i}{\omega_k \left(\frac{\pi_i}{\bar{\Pi}_i}\right)^{1-\sigma^k}} \right]^{\frac{1}{1-\sigma^k}}$$

The two-level nested, nonseparable constant-elasticity-of-substitution (NNCES) cost function is then defined as:

$$C(\pi) = \bar{C}\left(\sum_k \omega_k p_k(\pi)^{1-\gamma} \right)^{\frac{1}{1-\gamma}}$$

Demand indices for second-level aggregates are needed to express demand functions in a compact form. Let $z_k(\pi)$ denote the demand index for aggregate k, normalized to unity in the benchmark; i.e.

$$z_k(\pi) = \left(\frac{C(\pi)}{\bar{C}} \; \frac{1}{p_k(\pi)} \right)^{\gamma}$$

Compensated demand functions are obtained by differentiating $C(\pi)$. In this derivative, one term arise for each nest in which the commodity enters, so:

$$x_i(\pi) = \bar{x}_i \sum_K z_k(\pi) \left(\frac{p_k(\pi)\bar{\pi}_i}{\pi_i} \right)^{\sigma^k} = \bar{x}_i \sum_k \left(\frac{C(\pi)}{\bar{C}} \; \frac{1}{p_k(\pi)} \right)^{\gamma} \left(\frac{p_k(\pi)\bar{\pi}_i}{\pi_i} \right)^{\sigma^k}$$

Simple differentiation shows that benchmark cross-elasticities of substitution have the form:

$$\sigma_{ij} = \gamma + \sum_k \frac{(\sigma^k - \gamma) s_{ik} s_{jk}}{\omega_k}$$

Given the benchmark value shares θ_i and the benchmark cross-price elasticities of substitution, σ_{ij} , we can solve for values of , s_{ik} , ω_k, σ^k and γ . We compute these parameters using a constrained nonlinear programming algorithm, CONOPT, which is available through GAMS, the same programming environment in which the equilibrium model is specified. Perroni and Rutherford (EER, 1994) prove that calibration of the NNCES form is possible for arbitrary dimensions whenever the given Slutsky matrix is negative semi-definite. The two-level (NxN) function is flexible for three inputs; and although we have not proven that it is flexible for 4 inputs, the only difficulties we have encountered have resulted from indefinite calibration data points.

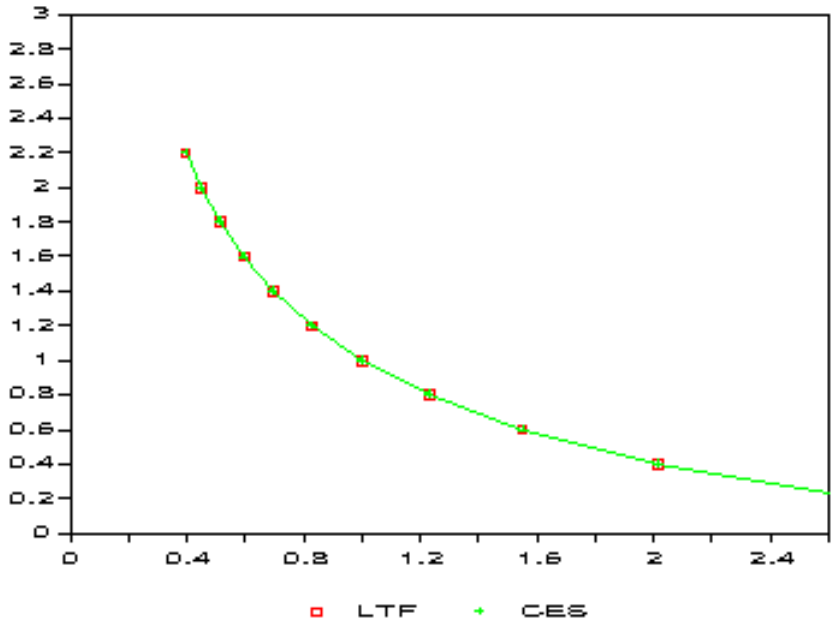
Two GAMS programs are listed below. The first illustrates two analytic calibrations of the three-factor cost function. The second illustrates the use of numerical methods to calibrate a four-factor cost function.

4.53.4.5 Two NNCES calibrations for a 3-input cost functions

```
* =============================================================================
*       Model-specific data defined here:

SET     I  Production input aggregates / A,B,C /;  ALIAS (I,J);

PARAMETER

    THETA(I)    Benchmark value shares /A 0.2, B 0.5, C 0.3/

    AUES(I,J)   Benchmark cross-elasticities (off-diagonals) /
                        A.B     2
                        A.C   -0.05
                        B.C     0.5 /;
* =============================================================================


*       Use an analytic calibration of the three-factor CES cost
*       function:

ABORT$(CARD(I) NE 3) "Error: not a three-factor model!";

*       Fill in off-diagonals:

AUES(I,J)$AUES(J,I)  =  AUES(J,I);

*       Verify that the cross elasticities are symmetric:

ABORT$SUM((I,J), ABS(AUES(I,J)-AUES(J,I))) " AUES values non-symmetric?";

*       Check that all value shares are positive:

ABORT$(SMIN(I, THETA(I)) LE 0) " Zero value shares are not valid:",THETA;

*       Fill in the elasticity matrices:

AUES(I,I) = 0; AUES(I,I) = -SUM(J, AUES(I,J)*THETA(J))/THETA(I); DISPLAY AUES;

SET     N       Potential nesting /N1*N3/
        K(N)    Nesting aggregates used in the model
        I1(I)   Good fully assigned to first nest
        I2(I)   Good fully assigned to second nest
        I3(I)   Good split between nests;

SCALAR  ASSIGNED /0/;

PARAMETER
        ESUB(*,*)       Alternative calibrated elasticities
        SHR(*,I,N)      Alternative calibrated shares
        SIGMA(N)        Second level elasticities
        S(I,N)          Nesting assignments (in model)
        GAMMA           Top level elasticity (in model);

*       First the Leontief structure:

ESUB("LTF","GAMMA") = SMAX((I,J), AUES(I,J));
ESUB("LTF",N) = 0;
```

```
LOOP((I,J)$((AUES(I,J) EQ ESUB("LTF","GAMMA"))*(NOT ASSIGNED)),
        I1(I) = YES;
        I2(J) = YES;
        ASSIGNED = 1;
);

I3(I) = YES$((NOT I1(I))*(NOT I2(I)));
DISPLAY I1,I2,I3;
LOOP((I1,I2,I3),
        SHR("LTF",I1,"N1") = 1;
        SHR("LTF",I2,"N2") = 1;
        SHR("LTF",I3,"N1") = THETA(I1)*(1-AUES(I1,I3)/AUES(I1,I2)) /
                    ( 1 - THETA(I3) * (1-AUES(I1,I3)/AUES(I1,I2)) );
        SHR("LTF",I3,"N2") = THETA(I2)*(1-AUES(I2,I3)/AUES(I1,I2)) /
                    ( 1 - THETA(I3) * (1-AUES(I2,I3)/AUES(I1,I2)) );
        SHR("LTF",I3,"N3") = 1 - SHR("LTF",I3,"N1") - SHR("LTF",I3,"N2");
);
ABORT$(SMIN((I,N), SHR("LTF",I,N)) LT 0) "Benchmark AUES is indefinite.";

*       Now, the CES function:

ESUB("CES","GAMMA") = SMAX((I,J), AUES(I,J));
ESUB("CES","N1") = 0;
LOOP((I1,I2,I3),
        SHR("CES",I1,"N1") = 1;
        SHR("CES",I2,"N2") = 1;
        ESUB("CES","N2") = (AUES(I1,I2)*AUES(I1,I3)-AUES(I2,I3)*AUES(I1,I1)) /
                           (AUES(I1,I3)-AUES(I1,I1));
        SHR("CES",I3,"N1") =
                (AUES(I1,I2)-AUES(I1,I3)) / (AUES(I1,I2)-AUES(I1,I1));
        SHR("CES",I3,"N2") = 1 - SHR("CES",I3,"N1");
);
ABORT$(SMIN(N, ESUB("CES",N)) LT 0)   "Benchmark AUES is indefinite?";
ABORT$(SMIN((I,N), SHR("CES",I,N)) LT 0) "Benchmark AUES is indefinite?";


PARAMETER       PRICE(I)        PRICE INDICES USING TO VERIFY CALIBRATION
                AUESCHK(*,I,J)  CHECK OF BENCHMARK AUES VALUES;

PRICE(I) = 1;

$ontext

$MODEL:CHKCALIB

$SECTORS:
        Y       ! PRODUCTION FUNCTION
        D(I)

$COMMODITIES:
        PY      ! PRODUCTION FUNCTION OUTPUT
        P(I)    ! FACTORS OF PRODUCTION
        PFX     ! AGGREGATE PRICE LEVEL

$CONSUMERS:
        RA

$PROD:Y  s:GAMMA  K.TL:SIGMA(K)
```

```
        O:PY            Q:1
        I:P(I)#(K)      Q:(THETA(I)*S(I,K))     K.TL:

$PROD:D(I)
        O:P(I)  Q:THETA(I)
        I:PFX   Q:(THETA(I)*PRICE(I))

$DEMAND:RA
        D:PFX
        E:PFX   Q:2
        E:PY    Q:-1
$OFFTEXT
$SYSINCLUDE mpsgeset CHKCALIB

SCALAR DELTA /1.E-5/;

SET     FUNCTION /LTF, CES/;

ALIAS (I,II);

LOOP(FUNCTION,

        K(N) = YES$SUM(I, SHR(FUNCTION,I,N));
        GAMMA = ESUB(FUNCTION,"GAMMA");
        SIGMA(K) = ESUB(FUNCTION,K);
        S(I,K) = SHR(FUNCTION,I,K);

        LOOP(II,

          PRICE(J) = 1; PRICE(II) = 1 + DELTA;

$INCLUDE CHKCALIB.GEN
        SOLVE CHKCALIB USING MCP;

        AUESCHK(FUNCTION,J,II) = (D.L(J)-1) / (DELTA*THETA(II));

));

AUESCHK(FUNCTION,I,J) = AUESCHK(FUNCTION,I,J) - AUES(I,J);
DISPLAY AUESCHK;

*       Evaluate the demand functions:

$LIBINCLUDE qadplot
SET PR Alternative price levels /PR0*PR10/;

PARAMETER
        DEMAND(FUNCTION,I,PR)   Demand functions
        DPLOT(PR,FUNCTION)      Plotting output array;

LOOP(II,
        LOOP(FUNCTION,
          K(N) = YES$SUM(I, SHR(FUNCTION,I,N));
          GAMMA = ESUB(FUNCTION,"GAMMA");
          SIGMA(K) = ESUB(FUNCTION,K);
          S(I,K) = SHR(FUNCTION,I,K);
          LOOP(PR,
            PRICE(J) = 1;
```




```
            PRICE(II) = 0.2 * ORD(PR);
$INCLUDE CHKCALIB.GEN
            SOLVE CHKCALIB USING MCP;
            DEMAND(FUNCTION,II,PR) = D.L(II);
            DPLOT(PR,FUNCTION) = D.L(II);
          );
        );

$LIBINCLUDE qadplot DPLOT PR FUNCTION

);

DISPLAY DEMAND;
```

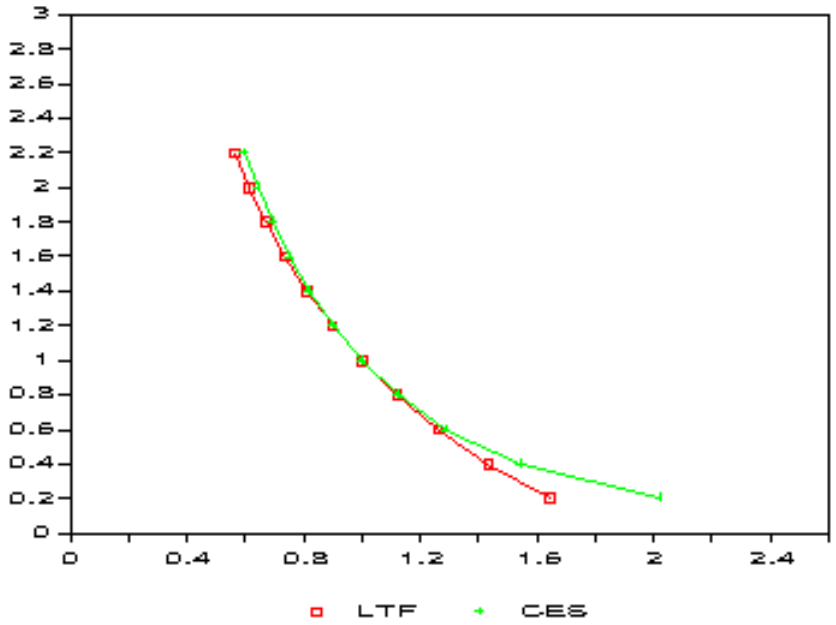
4.53.4.6 A Comparison of Locally-Identical Functions

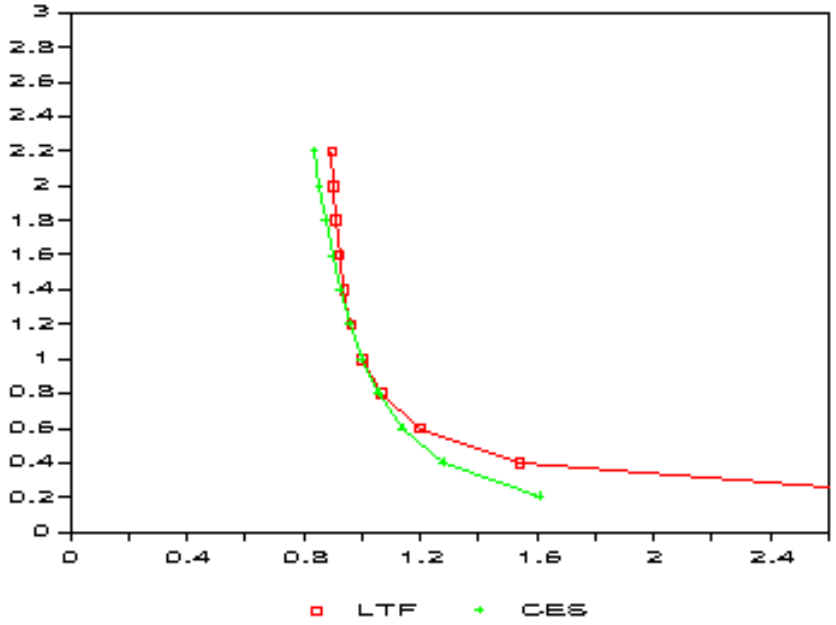
A Comparison of Locally-Identical Functions

Figure 1: Demand Function Comparison – Good A

Figure 2: Demand Function Comparison – Good B

Figure 3: Demand Function Comparison – Good C


3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.4
0.8
1.2
1.6
2
2.4
LTF
CES


3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
0.4
0.8
1.2
1.6
2
2.4
LTF
CES

4.53.4.7 Numerical calibration of NNCES given KLEM elasticities

```
SET  I  Production input aggregates / K, L, E, M/;  ALIAS (I,J);

* =============================================================================
*       Model-specific data defined here:

PARAMETER
    THETA(I)    Benchmark value shares /K 0.2, L 0.4, E 0.05, M 0.35/

    AUES(I,J)   Benchmark cross-elasticities (off-diagonals) /
                        K.L     1
                        K.E     -0.1
                        K.M     0
                        L.E     0.3
                        L.M     0
                        E.M     0.1 /;
* =============================================================================

SCALAR EPSILON          Minimum value share tolerance /0.001/;

*       Fill in off-diagonals:

AUES(I,J)$AUES(J,I)  =  AUES(J,I);

*       Verify that the cross elasticities are symmetric:

ABORT$SUM((I,J), ABS(AUES(I,J)-AUES(J,I))) " AUES values non-symmetric?";

*       Check that all value shares are positive:

ABORT$(SMIN(I, THETA(I)) LE 0) " Zero value shares are not valid:",THETA;

*       Fill in the elasticity matrices:

AUES(I,I) = 0; AUES(I,I) = -SUM(J, AUES(I,J)*THETA(J))/THETA(I); DISPLAY AUES;

* =============================================================================

*       Define variables and equations for NNCES calibration:

SET     N       Nests within the two-level NNCES function /N1*N4/,
        K(N)    Nests which are in use;

VARIABLES
        S(I,N)        Fraction of good I which enters through nest N,
        SHARE(N)        Value share of nest N,
        SIGMA(N)        Elasticity of substitution within nest N,
        GAMMA           Elasticity of substitution at the top level,
        OBJ             Objective function;

POSITIVE VARIABLES S, SHARE, SIGMA, GAMMA;

EQUATIONS
        SDEF(I)         Nest shares must sum to one,
        TDEF(N)         Nest share in total cost,
        ELAST(I,J)      Consistency with given AUES values,
        OBJDEF          Maximize concentration;
```

```
ELAST(I,J)$(ORD(I) GT ORD(J))..

                AUES(I,J) =E=   GAMMA +

                        SUM(K, (SIGMA(K)-GAMMA)*S(I,K)*S(J,K)/SHARE(K));

TDEF(K)..       SHARE(K) =E= SUM(I, THETA(I) * S(I,K));

SDEF(I)..       SUM(N, S(I,N)) =E= 1;

*       Maximize concentration at the same time keeping the elasticities
*       to be reasonable:

OBJDEF..        OBJ =E= SUM((I,K),S(I,K)*S(I,K))

                        - SQR(GAMMA) - SUM(K, SQR(SIGMA(K)));

MODEL   CESCALIB /ELAST, TDEF, SDEF, OBJDEF/;


*       Apply some bounds to avoid divide by zero:

SHARE.LO(N) = EPSILON;

SCALAR  SOLVED  Flag for having solved the calibration problem /0/
        MINSHR  Minimum share in candidate calibration;

SET     TRIES   Counter on the number of attempted calibrations /T1*T10/;

*       We use the random number generator to select starting points,
*       so it is helpful to initialize the seed so that the results
*       will be reproducible:

OPTION SEED=0;

LOOP(TRIES$(NOT SOLVED),

*       Initialize the set of active nests and the bounds:

        K(N) = YES;
        S.LO(I,N) = 0;          S.UP(I,N) = 1;
        SHARE.LO(N) = EPSILON;  SHARE.UP(N) = 1;
        SIGMA.LO(N) = 0;        SIGMA.UP(N) = +INF;

*       Install a starting point:

        SHARE.L(K)  = MAX(UNIFORM(0,1), EPSILON);
        S.L(I,K)    = UNIFORM(0,1);
        GAMMA.L     = UNIFORM(0,1);
        SIGMA.L(K)  = UNIFORM(0,1);

*       Drop any basis information so that we start from scratch:

        SDEF.M(I)     = 0; TDEF.M(K)     = 0; ELAST.M(I,J)  = 0;

        SOLVE CESCALIB USING NLP MAXIMIZING OBJ;
```

```
        SOLVED = 1$(CESCALIB.MODELSTAT LE 2);

*       We have a solution -- now see if it is not on a bound:

        IF (SOLVED,

          MINSHR = SMIN(K, SHARE.L(K)) - EPSILON;

          IF (MINSHR EQ 0,

*       Drop nests which have shares equal to EPSILON in the current
*       solution:

            K(N)$(SHARE.L(N) EQ EPSILON) = NO;

            S.FX(I,N)$(NOT K(N)) = 0;
            SHARE.FX(N)$(NOT K(N)) = 0;
            SIGMA.FX(N)$(NOT K(N)) = 0;

            DISPLAY "Recalibrating with the following nests:",K;

            SOLVE CESCALIB USING NLP MAXIMIZING OBJ;

            IF (CESCALIB.MODELSTAT GT 2, SOLVED = 0;);
            MINSHR = SMIN(K, SHARE.L(K)) - EPSILON;
            IF (MINSHR EQ 0, SOLVED = 0;);
          );
        );
);

IF (SOLVED,
        DISPLAY "Function calibrated:",GAMMA.L,SIGMA.L,SHARE.L,S.L;
ELSE
        DISPLAY "Function calibration fails!";
);

$ONTEXT

*=====================================================================

Solution from MINOS obtained on the second try, following an
ITERATION INTERRUPT on the first:

----    151 Function calibrated:


----    151 VARIABLE  GAMMA.L              =        0.300 Elasticity of
                                                         substitution at the
                                                         top level


----    151 VARIABLE  SIGMA.L       Elasticity of substitution within nest N

N3 7.804


----    151 VARIABLE  SHARE.L       Value share of nest N
```

```
N1 0.604,    N2 0.266,    N3 0.030,    N4 0.100


----    151 VARIABLE  S.L           Fraction of good I which enters through
                                     nest N

           N1          N2          N3          N4

K                   0.797       0.069       0.133
L       0.960                   0.040
E                                           1.000
M       0.630       0.304                   0.067


*=============================================================================
```

The following solution is obtained by CONOPT on the second try, following a LOCALLY INFEASIBLE termination on the first problem. Notice that it is identical to the MINOS solution except that the nesting assigments have been permuted:

```
----    149 Function calibrated:


----    149 VARIABLE  GAMMA.L              =        0.300 Elasticity of
                                                         substitution at the
                                                         top level


----    149 VARIABLE  SIGMA.L       Elasticity of substitution within nest N

N4 7.804


----    149 VARIABLE  SHARE.L       Value share of nest N

N1 0.100,    N2 0.604,    N3 0.266,    N4 0.030


----    149 VARIABLE  S.L           Fraction of good I which enters through
                                     nest N

           N1          N2          N3          N4

K       0.133                   0.797       0.069
L                   0.960                   0.040
E       1.000
M       0.067       0.630       0.304


*=============================================================================

$OFFTEXT


PARAMETER   PRICE(I)    PRICE INDICES USING TO VERIFY CALIBRATION
        AUESCHK(I,J)    CHECK OF BENCHMARK AUES VALUES;

PRICE(I) = 1;
```

```
$ontext

$MODEL:CHKCALIB

$SECTORS:
    Y   ! PRODUCTION FUNCTION
    D(I)

$COMMODITIES:
    PY  ! PRODUCTION FUNCTION OUTPUT
    P(I)    ! FACTORS OF PRODUCTION
    PFX ! AGGREGATE PRICE LEVEL

$CONSUMERS:
    RA

$PROD:Y  s:GAMMA.L  K.TL:SIGMA.L(K)
    O:PY        Q:1
    I:P(I)#(K)  Q:(THETA(I)*S.L(I,K))   K.TL:

$PROD:D(I)
    O:P(I)  Q:THETA(I)
    I:PFX   Q:(THETA(I)*PRICE(I))

$DEMAND:RA
    D:PFX
    E:PFX   Q:2
    E:PY    Q:-1
$OFFTEXT
$SYSINCLUDE mpsgeset CHKCALIB

CHKCALIB.ITERLIM = 0;
$INCLUDE CHKCALIB.GEN
SOLVE CHKCALIB USING MCP;
CHKCALIB.ITERLIM = 2000;

SCALAR DELTA /1.E-5/;

ALIAS (I,II);
LOOP(II,

    PRICE(J) = 1;   PRICE(II) = 1 + DELTA;

$INCLUDE CHKCALIB.GEN
    SOLVE CHKCALIB USING MCP;

    AUESCHK(J,II) = (D.L(J)-1) / (DELTA*THETA(II));

);
DISPLAY AUES, AUESCHK;
```

4.53.4.8 Calibrating Labor Supply and Savings Demand

This material was published in the MUG newsletter, 8/95.

Following Ballard, Fullerton, Shoven and Whalley (BFSW), we consider a representative agent whose utility is based upon current consumption, future consumption and current leisure. Changes in "future

consumption" in this static framework are associated with changes in the level of savings. There are three prices which jointly determine the price index for future consumption. These are:

P_I the composite price index for investment goods

P_K the composite rental price for capital services

P_C the composite price of current consumption.

All of these prices equal unity in the benchmark equilibrium.

Capital income in each future year finances future consumption, which is expected to cost the same as in the current period, PC (static expectations). The consumer demand for savings therefore depends not only on P_I, but also on P_K and P_C, namely:

$$P_S = \frac{P_I P_C}{P_K}$$

The price index for savings is unity in the benchmark period. In a counter-factual equilibrium, however, we would expect generally that $P_S \neq P_I$. When these price indices are not equal, there is a "virtual tax payment" associated with savings demand.

Following BFSW, we adopt a nested constant-elasticity-of-substitution function to represent preferences. In this function, at the top level demand for savings (future consumption) trades off with a second CES aggregate of leisure and current consumption. These preferences can be summarized with the following expenditure function:

$$P_U = \left[\alpha P_H^{1-\sigma_S} + (1-\alpha) P_H^{1-\sigma_S}\right]^{\frac{1}{1-\sigma_S}}$$

Preferences are homothetic, so we have defined P_U as a linearly homogeneous cost index for a unit of utility. We conveniently scale this price index to equal unity in the benchmark. In this definition, α is the benchmark value share for current consumption (goods and leisure). P_H is a compositive price for current consumption defined as:

$$P_H = \left[\beta P_l^{1-\sigma_L} + (1-\beta) P_C^{1-\sigma_L}\right]^{\frac{1}{1-\sigma_L}}$$

in which β is the benchmark value share for leisure within current consumption. Demand functions can be written as follows:

$$S = S_0 \left(\frac{P_U}{P_F}\right)^{\sigma_S} \quad \frac{I}{I_0 P_U},$$

$$C = C_0 \left(\frac{P_H}{P_C}\right)^{\sigma_L} \quad \left(\frac{P_U}{P_H}\right)^{\sigma_S} \quad \frac{I}{I_0 P_U},$$

and

$$\ell = \ell_0 \left(\frac{P_H}{P_L}\right)^{\sigma_L} \quad \left(\frac{P_U}{P_H}\right)^{\sigma_S} \quad \frac{I}{I_0 P_U},$$

Demands are written here in terms of their benchmark values (S_0, C_0 and ℓ_0) and current and benchmark income (I and I_0).

There are four components in income. The first is the value of labor endowment (E), defined inclusive of leisure. The second is the value of capital endowment (K). The third is all other income (M). The fourth is the value of "virtual tax revenue" associated with differences between the shadow price of savings and the cost of investment.

$$I = P_L E + P_K K + M + (P_S - P_I) S$$

The following parameter values are specified exogenously:

1. $\zeta = 1.75$ is the ratio of labor endowment:

$$\zeta \equiv E/L_0$$

where L_0 is the benchmark labor supply. Given ζ and L_0 we have:

$$\ell_0 = L_0(\zeta - 1)$$

2. $\xi = 0.15$ is the uncompensated elasticity of labor supply with respect to the net of tax wage, i.e.

$$\xi = \frac{\delta L}{\delta P_L}\frac{P_L}{L} = \frac{\delta(E-\ell)}{\delta P_L}\frac{P_L}{L} = -\frac{\delta \ell}{\delta P_L}\frac{P_L}{L}$$

3. $\eta = 0.4$ is the elasticity of savings with respect to the return to capital:

$$\eta \equiv \frac{\delta S}{\delta P_K}\frac{S}{P_K}$$

Shephard's lemma applied at benchmark prices provides the following identities which are helpful in deriving expressions for η and ξ :

$$\frac{\delta P_U}{\delta P_H} = \alpha, \frac{\delta P_U}{\delta P_S} = 1-\alpha, \frac{\delta P_H}{\delta P_L} = \beta, \frac{\delta P_H}{\delta P_C} = 1-\beta,$$

It is then a relatively routine application of the chain rule to show that:

$$\xi = (\zeta - 1)\left[\sigma_L + \beta(\sigma_S - \sigma_L) - \alpha\beta(\sigma_S - 1) - \frac{E}{I_0}\right]$$

and

$$\eta = \sigma_S\alpha + \frac{K}{I_0}$$

The expression for *eta* does not involve σ_L , so we may first solve for σ_S and use this value in determining σ_L :

$$\sigma_S = \frac{\eta - \frac{K}{I_0}}{\alpha}$$

and

$$\alpha_L = \frac{\frac{\xi}{\xi-1} - \sigma_S\beta(1-\alpha) - \alpha\beta + \frac{E}{I_0}}{1-\beta}$$

4.53.4.9 A Maquette Illustrating Labor Supply and Savings Demand Calibration

```
*       Exogenous elasticity:

SCALAR  XI      UNCOMPENSATED ELASTICITY OF LABOR SUPPLY /0.15/,
        ETA     ELASTICITY OF SAVINGS WRT RATE OF RETURN /0.40/,
        ZETA    RATIO OF LABOR ENDOWMENT TO LABOR SUPPLY /1.75/;

*       Benchmark data:

SCALAR  C0      CONSUMPTION /2.998845E+2/,
        S0      SAVINGS /70.02698974/,
        LS0     LABOR SUPPLY / 2.317271E+2/,
        K0      CAPITAL INCOME /93.46960577/,
        PL0     MARGINAL WAGE /0.60000000/;

*       Calibrated parameters:

SCALAR  EL0     LABOR ENDOWMENT
```

```
        L0      LEISURE DEMAND
        M0      NON-WAGE INCOME
        I       EXTENDED GROSS INCOME
        ETAMIN  SMALLEST PERMISSIBLE VALUE FOR ETA,
        XIMIN   SMALLEST PERMISSIBLE VALUE FOR XI,
        ALPHA   CURRENT CONSUMPTION VALUE SHARE
        BETA    LEISURE VALUE SHARE IN CURRENT CONSUMPTION
        SIGMA_L ELASTICITY OF SUBSTITUTION WITHIN CURRENT CONSUMPTION
        SIGMA_S ELASTICITY OF SUBSTITUTION - SAVINGS VS CURRENT CONSUMPTION
        TS      SAVINGS PRICE ADJUSTMENT;

*       Convert labor supply into net of tax units:

LS0 = LS0 * PL0;

*       Labor endowment (exogenous):

EL0 = ZETA * LS0;

*       Leisure demand:

L0 = EL0 - LS0;

*       Non-labor, non-capital income:

M0 = C0 + S0 - LS0 - K0;

*       Extended gross income:

I = L0 + C0 + S0;

*       Leisure share of current consumption:

BETA =  L0 / (C0 + L0);

*       Current consumption value share:

ALPHA = (L0 + C0) / I;

*       Calibrated elasticity:

SIGMA_S = (ETA - K0 / I) / ALPHA;
ETAMIN  = K0 / I;
ABORT$(SIGMA_S LT 0) " Error: cannot calibrate SIGMA_S", ETAMIN;

*       Calibrated elasticity of substitution between leisure and consumption:

SIGMA_L = (XI*(LS0/L0)-SIGMA_S*BETA*(1-ALPHA)-ALPHA*BETA+EL0/I)/(1-BETA);
XIMIN = -(L0/LS0) * (- SIGMA_S * BETA * (1-ALPHA) - ALPHA*BETA + EL0/I);
ABORT$(SIGMA_L LT 0) " Error: cannot calibrate SIGMA_L", XIMIN;

DISPLAY "Calibrated elasticities:", SIGMA_S, SIGMA_L;

$ONTEXT

$MODEL:CHKCAL

$COMMODITIES:
```

```
        PL
        PK
        PC
        PS

$SECTORS:
        Y
        S

$CONSUMERS:
        RA

$PROD:Y
        O:PC    Q:(K0+LS0-S0)
        I:PL    Q:(LS0-S0)
        I:PK    Q:K0

$PROD:S
        O:PS    A:RA  T:TS
        I:PL

$DEMAND:RA  s:SIGMA_S a:SIGMA_L
        E:PC    Q:M0
        E:PL    Q:EL0
        E:PK    Q:K0
        D:PS    Q:S0
        D:PC    Q:C0  a:
        D:PL    Q:L0  a:
$OFFTEXT
$SYSINCLUDE mpsgeset CHKCAL

S.L = S0;
TS  = 0;

*       VERIFY THE BENCHMARK:

CHKCAL.ITERLIM = 0;
$INCLUDE CHKCAL.GEN
SOLVE CHKCAL USING MCP;

*       CHECK THE LABOR SUPPLY ELASTICITY:

PL.L = 1.001;

CHKCAL.ITERLIM = 0;
$INCLUDE CHKCAL.GEN
SOLVE CHKCAL USING MCP;
*       Compute induced changes in labor supply using the labor market
*       "marginal", PL.M.  This marginal returns the net excess supply of
*       labor at the given prices.  We started from a balanced benchmark,
*       with no change in labor demand (the iteration limit was zero).
*       Hence, PL.M returns the magnitude of the change in labor supply.
*       We multiply by the benchmark wage (1) and divide by the benchmark
*       labor supply (LS0) to produce a finite difference approximation
*       of the elasticity:

DISPLAY "CALIBRATION CHECK -- THE FOLLOWING VALUES SHOULD BE IDENTICAL:", XI;
XI = (PL.M / 0.001) * (1 / LS0);
```

```
DISPLAY XI;

PL.L = 1.0;

*       CHECK THE ELASTICITY OF SAVINGS WRT RENTAL RATE OF CAPITAL:

PK.L = 1.001;
PS.L = 1 / 1.001;
TS   = 1 / 1.001 - 1;

CHKCAL.ITERLIM = 0;

*       Compute elasticity of savings with respect to the rental rate of
*       capital.  This requires some recursion in order to account for the
*       effect of changes in savings on effective income.  When PK increases,
*       PS declines -- there is an effective "subsidy" for saving, paid from
*       consumer income.  In order to obtain a difference approximation for
*       the elasticity of savings response, we need to make sure the virtual
*       tax payments are properly handled.  In the MPSGE model, this means
*       that the level value for S must be adjusted so that it exactly equals
*       the savings.  We do this recursively:

SET ITER /IT1*IT5/;


PS.M = 1;
LOOP(ITER$(ABS(PS.M) GT 1.0E-8),

$INCLUDE CHKCAL.GEN
        SOLVE CHKCAL USING MCP;
        S.L = S.L - PS.M;
);

DISPLAY "CALIBRATION CHECK -- THE FOLLOWING VALUES SHOULD BE IDENTICAL:", ETA;
ETA = ((S.L - S0) / 0.001) * (1 / S0);
DISPLAY ETA;
```

Chapter 5

Solver Manuals

A large number of solvers for mathematical programming models have been hooked up to GAMS. The tables below provide a brief description of each solver, the `model types` each solver is cabable of solving, and the `platforms` supported by each solver. For general information on using GAMS solvers, see Solver Usage.

Solver	Vendor	Description
ALPHAECP 2.11	Abo University	MINLP solver based on the extended cutting plane (ECP) method
ANTIGONE 1.1	Princeton University	Deterministic global optimization for MINLP
BARON	The Optimization Firm, LLC	Branch-And-Reduce Optimization Navigator for proven global solutions
CBC 2.10	COIN-OR Foundation	High-performance LP/MIP solver
CONOPT 3	ARKI Consulting and Development	Large scale NLP solver
CONOPT 4	ARKI Consulting and Development	Large scale NLP solver
CONVERT	GAMS Development Corp	Framework for translating modes into scalar models of other languages
COPT 7.0	Cardinal Operations	High-performance LP/MIP solver
CPLEX 22.1	IBM ILOG	High-performance LP/MIP solver
DE	GAMS Development Corp	Generates and solves the deterministic equivalent of a stochastic program, included in EMP/SP
DECIS	G. Infanger, Inc.	Large scale stochastic programming solver
DICOPT 2	GAMS Development Corp	Framework for solving MINLP models
EXAMINER	GAMS Development Corp	A tool for examining solution points and assessing their merit
GAMSCHK	Bruce McCarl	A System for Examining the Structure and Solution Properties of Linear Programming Problems Solved using GAMS
GUROBI 11.0	Gurobi Optimization	High performance LP/MIP solver
GUSS	GAMS Development Corp	A framework for solving many instances of related models efficiently (Gather-Update-Solver-Scatter)
HiGHS 1.6	ERGO	High performance LP/MIP solver

Solver	Vendor	Description
IPOPT 3.14	COIN-OR Foundation	Interior Point Optimizer for large scale nonlinear programming
JAMS	GAMS Development Corp	Solver to reformulate extended mathematical programs (incl. LogMIP)
KESTREL	NEOS	Framework for using remote NEOS solvers with a local GAMS system
KNITRO 14.0	Artelys	Large scale NLP and MINLP solver
LINDO 14.0	Lindo Systems Inc.	A stochastic solver from Lindo Systems, Inc. Includes an unrestricted version of LINDOGLOBAL
LINDOGLOBAL 14.0	Lindo Systems Inc.	MINLP solver for proven global solutions
MILES	University of Colorado at Boulder	MCP solver
MINOS 5.6	Stanford University	NLP solver
MOSEK 10	MOSEK ApS	Large scale mixed-integer conic programming solver
NLPEC	GAMS Development Corp	MPEC to NLP translator that uses other GAMS NLP solvers
ODHCPLEX 7	Optimization Direct Inc	ODHeuristic on top of Cplex
PATHNLP	University of Wisconsin - Madison	Large scale NLP solver for convex problems
PATH	University of Wisconsin - Madison	Large scale MCP solver
QUADMINOS 5.6	Stanford University	Quad-precision NLP solver
SBB	ARKI Consulting and Development	Branch-and-Bound algorithm for solving MINLP models
SCIP 8.1	Zuse Institute Berlin et.al.	High-performance Constraint Integer Programming solver
SHOT 1.1	Abo Akademi University	MINLP solver based on the extended supporting hyperplane (ESH) method
SNOPT 7.7	Stanford University	Large scale SQP based NLP solver
SOPLEX 6.0	Zuse Institute Berlin	High-performance LP solver
XPRESS 41.01	FICO	High performance LP/MIP and SLP based MINLP solver

5.1 Model Types

GAMS is able to formulate models in many different types of problem classes or model types. Typically, a solver will be capable of solving (i.e. will accept as input) more than one model type. The solver/model type matrix shows which solver is capable of which model type:

	LP	MIP	NLP	MCP	MPEC	CNS	DNLP	MINLP	QCP	MIQCP	*Stoch.*	*Global*
ALPHAECP								✓		✓		
ANTIGONE			✓			✓	✓	✓	✓	✓		✓*
BARON	✓	✓	✓			✓	✓	✓	✓	✓		✓*
CBC	✓	✓										
CONOPT 3	✓		✓			✓	✓		✓			
CONOPT 4	✓		✓			✓	✓		✓			

	LP	MIP	NLP	MCP	MPEC	CNS	DNLP	MINLP	QCP	MIQCP	*Stoch.*	*Global*
COPT	✓	✓							✓	✓		
CPLEX	✓	✓							✓	✓		
DECIS	✓										✓	
DICOPT								✓		✓		
GUROBI	✓	✓	✓				✓	✓	✓	✓		✓∗
GUSS	✓	✓	✓	✓		✓	✓	✓	✓	✓		
IPOPT	✓		✓			✓	✓		✓			
HiGHS	✓	✓										
KESTREL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		
KNITRO	✓		✓	✓	✓	✓	✓	✓	✓	✓		
LINDO	✓	✓	✓				✓	✓	✓	✓	✓	✓∗
LINDOGLOBAL	✓	✓	✓				✓	✓	✓	✓		✓∗
MILES				✓								
MINOS	✓		✓			✓	✓		✓			
MOSEK	✓	✓	✓				✓	✓	✓	✓		
NLPEC				✓	✓							
ODHCPLEX		✓								✓		
PATH				✓		✓						
QUADMINOS	✓											
SBB								✓		✓		
SCIP		✓	✓			✓	✓	✓	✓	✓		✓∗
SHOT								✓		✓		
SNOPT	✓		✓			✓	✓		✓			
SOPLEX	✓											
XPRESS	✓	✓	✓			✓	✓	✓	✓	✓		

∗ deterministic global solver

When choosing a solver, some judgement should be applied when considering the listed model type capabilities for the solver - the same capability "check boxes" does not imply equality in capacity or suitability. For example, take a hypothetical solver WeOpt designed to solve MINLP models. Since the problem class MINLP includes NLP, MIP, and LP as subclasses, solver WeOpt could include these capabilities also. If WeOpt is also a good performer on NLP models, it would include that capability. But if it does not shine at all as a MIP or LP solver, we would choose not to include MIP and LP in the capability list for WeOpt. In such a case one can always solve using a more general model type (e.g. solve an LP model as NLP so WeOpt can be used), but WeOpt will not advertise itself as an LP solver. Since the WeOpt solver does not even recognize MCP or MPEC models, we don't include those capabilities.

There are two types of non-linear solvers: local and global. A local solver can find a local optimum but generally cannot comment on global optimality of the solution. A local optimum is a point where the objective value is better than its neighboring points but could be worse than a distant point. On the other hand, a global solver can find and prove that the obtained solution is a global optimum i.e., there is no feasible solution that would result in a better objective value. It is worth noting that a local solver can solve a model to global optimality in some special cases. In the table shown, the entries with ∗ in the column Global indicate solvers that can solve a model to global optimality.

There are other differences in solvers that are difficult to quantify or cannot be captured by a capability table like the one shown. For example, for nonconvex NLP or QCP models, one solver could look only for first-order stationary points, another for local solutions, a third for local solutions using a scatter search or similar search heuristic, and a fourth could do a true global search for the global optimum. The relative merits (measured typically by speed alone) of solvers is the subject of considerable benchmarking activity and discussion.

The GAMS sales team can help answer questions you may have about solver capability. We also offer free evaluation licenses to help you decide what solvers are most suitable for your models.

5.2 Supported Platforms

The solver/platform matrix shows which platforms each solver is supported on. In addition, where a vendor has discontinued solver support for a particular platform and we continue to ship the last available supported version, this version number is indicated as well.

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
ALPHAECP 2.11	✓	✓	✓	✓
ANTIGONE 1.1	✓	✓	✓	✓
BARON	✓	✓	✓	✓
CBC 2.10	✓	✓	✓	✓
CONOPT 3	✓	✓	✓	✓
CONOPT 4	✓	✓	✓	✓
COPT 7.0	✓	✓	✓	✓
CPLEX 22.1	✓	✓	✓	✓
DECIS	✓	✓	✓	✓
DICOPT 2	✓	✓	✓	✓
GUROBI 11.0	✓	✓	✓	✓
GUSS	✓	✓	✓	✓
IPOPT 3.14	✓	✓	✓	✓
HiGHS 1.6	✓	✓	✓	✓
KESTREL	✓	✓	✓	✓
KNITRO 14.0	✓	✓	✓	✓
LINDO 14.0	✓	✓	✓	✓
LINDOGLOBAL 14.0	✓	✓	✓	✓
MILES	✓	✓	✓	✓
MINOS 5.6	✓	✓	✓	✓
MOSEK 10	✓	✓	✓	✓
NLPEC	✓	✓	✓	✓
ODHCPLEX 7	✓	✓		
PATH	✓	✓	✓	✓
QUADMINOS 5.6	✓	✓	✓	✓
SBB	✓	✓	✓	✓
SCIP 8.1	✓	✓	✓	✓
SHOT 1.1	✓	✓	✓	✓
SNOPT 7.7	✓	✓	✓	✓
SOPLEX 6.0	✓	✓	✓	✓
XPRESS 41.01	✓	✓	✓	

5.3 AlphaECP

Author

Tapio Westerlund (twesterl@abo.fi) and Toni Lastusilta. Åbo Akademi University, Finland
Ville-Pekka Eronen. University of Turku, Finland

5.3.1 Introduction

AlphaECP is a Mixed-Integer Non-Linear Programming (MINLP) solver based on the extended cutting plane (ECP) method. The solver can be applied to general MINLP problems and it can ensure global

optimal solutions for pseudo-convex MINLP problems. The documentation is written as if the considered problem is a minimization problem, if not otherwise denoted.

The ECP method is an extension of Kelley's cutting plane method, which was originally given for convex NLP problems [105] . The method only requires the solution of a MIP sub-problem in each iteration. The MIP sub-problems can be solved to optimality, to feasibility, or only to an integer relaxed solution in intermediate iterations. This makes the ECP algorithm efficient and easy to implement. Further information about the underlying algorithm can be found in [200] and [145] [174] [199] [201] .

Further developments of the GAMS/AlphaECP algorithm have introduced additional functionality. A NLP solver can now be called at MIP solutions. This improves AlphaECP's ability to find feasible and accurate solutions, especially for MINLP problems which mainly contain continuous variables. Furthermore, a heuristic that reselects cutting planes during the iteration procedure can be used to improve the capability of solving non-convex problems. In addition to the termination based on constraint tolerance the algorithm can terminate also if relative objective gap termination criterion is satisfied. This is only supported for convex problems.

5.3.1.1 Licensing and software requirements

Users need to have a GAMS/AlphaECP license in order to use GAMS/AlphaECP. In addition, a licensed MIP solver is required for solving the mixed integer sub-problem, and a licensed NLP solver is required if the NLP option is used.

5.3.1.2 Running AlphaECP

AlphaECP solves MINLP models. If AlphaECP is not specified as the default solver for these models it can be invoked by issuing the following command before the solve statement:

```
option minlp=alphaecp, miqcp=alphaecp;
```

In principle AlphaECP can also handle NLP models, but it is more suitable for MINLP problems. However, when combined with an NLP solver it can find solutions the NLP solver cannot find by itself. In this case it acts as a good starting point generator. If you want to solve NLPs with AlphaECP you need to *trick* the GAMS system by solving your NLP as an MINLP:

```
solve mynlpmodel minimizing obj using minlp;
```

Constraint violations are reported throughout the progress of AlphaECP and at the end of the algorithm. The violation is reported for the non-linear constraints only. The violation of the linear constraints is subject to the feasibility tolerance of the MIP/NLP solver.

5.3.2 GAMS/AlphaECP Output

The log output below is obtained for the MINLP model fuel.gms from the GAMS model library:

```
--------------------------------------------------------------------------------
                  Welcome to AlphaECP v2.11.01
 MINLP Problem Solver using the Extended Cutting Plane Approach.
 Method development - T.Westerlund, Abo Akademi University, FIN
 Algorithm development - T.Lastusilta, Abo Akademi University, FIN
 Algorithm development - V.-P. Eronen, Turku University, FIN
 Westerlund Tapio and Poern Ray (2002). Optimization & Engineering, 3, 253-280
--------------------------------------------------------------------------------
Minimization problem: "fuel.gms"
The GAMS-model has in total 39 elements of which 15% are non-linear(NL)
included in 16 constraints of which 25% are NL
The NL constraint signs: =E=(3), =G=(1), =L=(0)
The number of variables in NL elements are 6 from a total of 16
variables: Continuous(13), Binary(3), Integer(0)
--------------------------------------------------------------------------------
Using following settings
AlphaECP option file                                        optfile=0
Time limit for AlphaECP (in seconds)                        reslim=10000000000
Solvelink for NLP and MIP sub-solver                        solvelink=5
Solver trace file                                           solvetrace=(Inactive)
Cutting plane strategy (0-3)                                CUTdelcrit=3
Cut generation pace                                         CUTnrcuts=0
Updating multiplier if MIP is infeasible                    ECPbeta=1.3
Write encountered solutions to gdx files                    ECPdumpsol=0
Updating multiplier when verifying solution                 ECPgamma=2
Maximum number of AlphaECP iterations                       ECPiterlim=-1
Level of AlphaECP output to statusfile (0-4)                ECPloglevel=0
Master strategy (0=User 1=Convex)                           ECPmaster=0
Return solution (1.MIP/2.NLP/...)                           ECPretsol=2
User specified start-point (0-3)                            ECPstart=3
AlphaECP strategy (1-5)                                     ECPstrategy=2
AlphaECP termination criterion (1-2)                        ECPtoltype=1
Upper limit of considered MIP solutions per MIP call        MIPnrsols=50
Relative MIP gap in intermediate sub-problems (0->1.0)      MIPoptcr=1.00
Initial iteration limit when MIPoptcr is reduced            MIPoptcrlim=200
Strategy to increase MIPoptcrlim                            MIPoptcrlimtype=0
MIP is solved to optimality with this frequency             MIPoptimaliter=0
Strategy for multiple MIP solutions                         MIPsolstrat=1
MIP solver for sub-problems and . option file number        MIPsolver=cplex.0
NLP strategy. Inactive:0 Active strategy:1-5                NLPcall=5
NLP solver call at next (incremental) iteration             NLPcalliter=0
NLP time limit per call (in seconds or auto=0)              NLPreslim=30
NLP solver for sub-problems and . option file number        NLPsolver=conopt.0
Constraint tolerance                                        TOLepsg=0.001
Distance tolerance for a new linearization                  TOLepsz=0.1
Gradient tolerance                                          TOLgrad=1e-06
Infinity bound (MIP variable bound)                         TOLinfbnd=1e+10
Relative termination tolerance for MINLP                    TOLoptcr=(Inactive)
--------------------------------------------------------------------------------
Itera  Stepcode, Number  Point Alpha OPT Movement Viol  Maximum   MIPobjval
tion   Problems  of Cuts usage Upd.  CR  Norm     Cons  Violation
Start-point: NL constraint (1) infeasibile
0      H         0       0     0     1   0        4     1.8E+03   NA
```

```
1      SAFGI     1       1     1     1   9.3E+03  0     1.1E-13   8566.12
1      FOUND SOLUTION:   8566.12      (NLP) in 1 sec.

2      SAFH      1       1     0     1   6.6E+03  4     1.8E+03   4844.02
3      SAFH      3       2     0     1   8.4E+03  3     1.8E+03   7031.72
4      SAFH      4       3     0     1   1E+03    2     1.8E+03   10157
5      SAH       5       4     0     1   0        1     7E+02     11925
6      SAH       6       5     0     1   1.1E+03  2     3.5E+02   11925
7      SAH       8       6     0     1   1.9E+03  2     5.3E+02   11925
8      SAH       9       7     0     1   8.8E+02  3     2.6E+02   11925
9      SAH       12      8     0     1   8.8E+02  3     1.3E+02   11925
10     SAH       15      9     0     1   4.4E+02  3     66        11925
...
79     SAIJL     101     65    0     1   0        0     0.00067   11925
80     SAJ       100     64    0     0.5 0        0     0.00067   11925
81     SAJ       100     64    0     0.3 0        0     0.00067   11925
...
100    AI        106     39    1     0   0        0     0.00067   11925
101    AI        106     39    1     0   0        0     0.00067   11925
102    AIJ       106     39    0     0   0        0     0.00067   11925
AlphaECP: Iteration procedure terminated normally
--------------------------------------------------------------------------------
Problem                    : fuel.gms
Solver Status              : Normal Completion
Model Status               : Locally Optimal
Exit comment               : No Issues
Final solution             : NLP
Objective value            : 8566.1189616876672517
Max constraint (4)         : 1.1368683772161602974e-13
Alternative solution       : MIP
Alt. objective value       : 8566.1189616876672517
Max constraint (4)         : 1.1368683772161602974e-13
Time used (seconds)        : 0.81
Time limit (seconds)       : 10000000000
Iterations used            : 102
Iteration limit            : -1
Function evaluations       : 496
Gradient evaluations       : 186
Domain violations          : 0
Gradients unusable         : 0
Alphamax bound violations  : 0
ECP time usage             : 3.9 %
NLP time usage             : 3.8 %
MIP time usage             : 92.3 %
Optimal/total MIPs         : 19/102
NLP solver calls           : 8
--------------------------------------------------------------------------------
```

In every iteration, information about the MIP problem and modifications to it is given in 10 columns. Here is a description of the different columns:

Iteration: Iteration identifier.

Stepcode, Problems: Letter for what actions were taken in this iteration, i.e. MIP problem modifications before the next iteration.

A: MIP solver feasible.

B: MIP solver feasible after moving cutting planes, i.e. alpha update.

C: MIP solver feasible after moving cutting planes close to their generation point. The movement is done to make it easier to satisfy nonlinear equality constraints.

D: Line search was successful (in ECPstrategy `3`).

E: Line search failed (in ECPstrategy `3`).

F: A NLP solver was called.

G: Found a MINLP solution.

H: Added linearization(s) to the next MIP problem.

I: Updated alpha values and possibly added linearizations.

J: All cutting planes are valid underestimators for the pseudo-convex constraints, except for the nonlinear objective function constraint.

K: The nonlinear objective function constraint value and MIP solution value differ more than ϵ_f. A linearization was done to reduce the difference (in ECPstrategy `3`).

L: Removed all temporal linearizations.

M: Domain violation(s), some of the constraint could not be evaluated.

N: Some cutting plane(s) could not be generated because of gradient problems.

O: No cutting planes could be generated.

P: Reselecting cuts because cutting planes are repeatedly moved close to their generation point.

Q: Added temporal linearization(s).

R: Failed to add temporal linearization(s).

S: MIP solver strategy to find encountered solutions selected.

T: MIP solver strategy to require MIPnrsols solutions selected.

U: MIP solver strategy to require MIPnrsols solutions with a MIPoptcr <= `0.2` selected.

Number of Cuts: The number of cutting planes the solved MIP problem had.

Point usage: Number of points used to generate the cuts in the solved MIP problem.

Alpha Upd.: The number of times the alpha values has been increased.

OPTCR: Relative MIP gap. Note that this is different from TOLoptcr.

Movement Norm: The Euclidean norm of the current and previous MIP solution.

Viol Cons: Number of unsatisfied (violating) nonlinear constraints.

Maximum Violation: The most violating nonlinear constraint value.

MIPobjval/NLobjval: MIP or nonlinear objective variable value depending on ECPstrategy setting.

The cut reselection heuristic is called in the following cases:

1. If the MIP solver would otherwise return infeasible.
2. When the violation is not reducing, but the cutting planes are repeatedly moved close to their generation point.
3. When the violation is not reducing and domain violations are repeatedly encountered.

The heuristic reselects cutting planes in different ways, but always ensures that the same point cannot be found twice.

```
Pointusage     6/90     Cutusage     15/341    (    0,135  )
```

`Pointusage` informs how many points of all usable points have been used to generate the cutting planes. `Cutusage` tells how many cuts of all usable cuts have been used. The first number in `( 0,135 )` tells how many cuts is required by the user, see CUTnrcuts and the second number gives the sum of added and removed cuts, i.e. a measure of how much the MIP problem has been modified. AlphaECP may fix some cuts and remove points and cuts during the cut reselection procedure in order to save memory.

At the end of each solve AlphaECP gives a summary which includes Problem, Solver Status, Model Status, etc. Note the following lines:

- **Exit comment** may give further information than solverstatus on why the solution procedure stopped.
- **Domain violations** (function evaluation failed) or **Gradients unusable** (all gradients < TOLgrad) might be caused by poor variable bounds.
- **Alphamax bound violations** inform the user how many times an alphamax value was calculated to be $> 10^{154}$ and was reset to 10^{154}.

5.3.3 Notes about Options

To instruct AlphaECP to read an option file you may use **ModelName.OptFile = 1**. The name of the option file is in this case `alphaecp.opt`. For further information, see The Solver Options File. AlphaECP supports the GAMS parameters reslim and optCR, however, other GAMS parameters are passed on to the sub-solvers. Note that optCR is transferred to option TOLoptcr. Furthermore, you may also pass additional sub-solver specific options to the sub-solvers. For example, if you want to use all available threads and sub-solver CPLEX in opportunistic parallel search mode, then you may specify this in a GAMS model, in a similar way, as follows:

```
Model m / all /;
Option threads=0;
m.optfile=1;
$echo MIPsolver cplex.1 > alphaecp.opt
$echo parallelmode -1 > cplex.opt
Solve m using MINLP minimizing objvar;
```

The following information is worth noting when you are interested in AlphaECP options. A linearization of a nonlinear constraint is called a cutting plane or cut. Here a point refers to the variable levels. Global optimality can be guaranteed for pseudo-convex problems. However, if the objective variable is in a nonlinear constraint and pseudo-convex, then ECPstrategy >= 3 needs to be used to guarantee global optimality (because one non-linear equality constraint makes a problem non-pseudo-convex, and hence also non-convex). The basic options might significantly impact the solution procedure, and the best values are likely to be problem specific. The user is therefore encouraged to try different values for the basic options.

5.3.4 Summary of AlphaECP Options

5.3.4.1 Basic options

Option	Description	Default
CUTnrcuts	Cut generation pace	0
ECPmaster	Master strategy (0=User 1=Convex)	0
MIPnrsols	Upper limit of considered MIP solutions per MIP call	50
MIPsolstrat	MIP solution collection strategy	1
MIPsolver	MIP solver for sub-problems and . option file number	GAMS MIP solver
NLPsolver	NLP solver for sub-problems and . option file number	GAMS NLP solver
reslim	Time limit for AlphaECP (in seconds)	GAMS reslim

5.3.4.2 Algorithmic options for advanced users

Option	Description	Default
CUTdelcrit	Cutting plane strategy	3
ECPbeta	Updating multiplier if MIP is infeasible	1.3
ECPdumpsol	Write encountered solutions to gdx files	0
ECPgamma	Updating multiplier when verifying solution	2.0
ECPiterlim	Maximum number of AlphaECP iterations	-1
ECPloglevel	Level of AlphaECP output to statusfile	0
ECPpcostrategy	Pseudo-convex objective function strategy	3
ECPretsol	Return solution (1.MIP/2.NLP/3.QUALITY/4.PERFORMANCE)	2
ECPstart	User specified start-point	3
ECPstrategy	AlphaECP strategy	2
ECPtoltype	AlphaECP termination criterion	1
solvelink	Solvelink for NLP and MIP sub-solver	5
solvetrace	Filename of solvetrace file	
solvetracetime	Time interval when a trace record is written	1
TOLepsf	Pseudo-convex objective function termination tolerance	1e-3
TOLepsg	Constraint tolerance	1e-3
TOLepsz	Distance tolerance for a new linearization	1e-1
TOLgrad	Gradient tolerance	1e-6
TOLinfbnd	Infinity bound (MIP variable bound)	1e10
TOLoptcr	Relative termination tolerance for MINLP	GAMS optCR

5.3.4.3 MIP Solver related options

Option	Description	Default
MIPloglevel	Level of MIP solver output	0
MIPoptcr	Relative MIP gap in intermediate sub-problems	1.0
MIPoptcrlim	Initial iteration limit when MIPoptcr is reduced	200
MIPoptcrlimtype	Strategy to increase MIPoptcrlim	0
MIPoptimaliter	MIP is solved to optimality with this frequency	0

5.3.4.4 NLP Solver related options

Option	Description	Default
NLPcall	NLP strategy	5
NLPcalliter	NLP solver call at next (incremental) iteration	0
NLPlimsameint	NLP call after a number of recurring integer solutions	5
NLPloglevel	Level of NLP solver output	0
NLPreslim	NLP time limit per call	0

5.3.5 Detailed Descriptions of AlphaECP Options

CUTdelcrit *(integer)*: Cutting plane strategy ↩

Default: 3

value	meaning
0	Do not remove any valid cuts.
1	As 0 and allow temporary cuts at semirandom points if normal cuts can not be made.
2	Allow temporary cuts and cut reselection, and use memory to save points and cuts.
3	As 2 and call the reselection heuristic before termination to improve the solution.

CUTnrcuts *(real)*: Cut generation pace ↩

The number of linearizations that are generated during an iteration can be chosen by AlphaECP, proportional to the number of violating constraints, or can be determined by a fixed amount. Furthermore, the cut reselection CUTdelcrit >=2 adds cuts to the problem so that the requested cut generation pace is taken into consideration.

Default: 0

value	meaning
0	Let AlphaECP decide.
0<n<1	Number of linearizations = n∗ the number of linearizations that is possible to generate.
>1	Specifies the number of linearizations to generate.

ECPbeta *(real)*: Updating multiplier if MIP is infeasible ↩

In case of an infeasible MIP solution, the invalid cuts are updated with the ECPbeta multiplier.

Range: [1.001, ∞]

Default: 1.3

ECPdumpsol *(integer)*: Write encountered solutions to gdx files ↩

Default: 0

value	meaning
0	No.
1	Solutions that the NLP solver found.
2	Solutions that the NLP or MIP solver found.

ECPgamma *(real)*: Updating multiplier when verifying solution ↩

If a MINLP solution is obtained but some cuts are not valid underestimators they are updated with the ECPgamma multiplier in order to make them into valid underestimators.

Range: [1.001, ∞]

Default: 2.0

ECPiterlim *(integer)*: Maximum number of AlphaECP iterations ↩

This is the maximum number of iterations given to AlphaECP to perform the optimization. Value -1 deactivates the AlphaECP iteration limit.

Default: -1

value	meaning
-1	No limit.
>=0	Specifies an iteration limit.

ECPloglevel *(integer)*: Level of AlphaECP output to statusfile ↩

Default: 0

value	meaning
0	No additional output to statusfile.
1	Report solutions. Report all encountered solutions with their corresponding variable levels.
2	Report main actions at iteration level (available for minimization problems).
3	Report main actions at linearization level (available for minimization problems).
4	Full reporting. Report the main actions taken, the linearizations, function values, and solution points for every iteration and line search details (available for minimization problems).

ECPmaster *(integer)*: Master strategy (0=User 1=Convex) ↩

The master strategy sets some options in order to solve a model with specific characteristics more efficiently. The affected options are noted in the log output. The set options takes precedence over the value set by the user for the affected options.

Default: `0`

value	meaning
0	Use only user defined options.
1	The model is convex. Set option ECPstrategy, CUTdelcrit and ECPtoltype.

ECPpcostrategy *(integer)*: Pseudo-convex objective function strategy ↩

Default: `3`

value	meaning
1	Remove support. Remove old support planes when a new pseudo-convex problem is formed.
2	Replace support. Replace old support planes with linearizations of the reduction constraint when a new pseudo-convex problem is formed.
3	Remove support and line search. Remove old support planes when a new pseudo-convex problem is formed and perform a line search when it is possible.
4	Replace support and line search. Replace old support planes with linearizations of the reduction constraint when a new pseudo-convex problem is formed and perform a line search when it is possible.

ECPretsol *(integer)*: Return solution (1.MIP/2.NLP/3.QUALITY/4.PERFORMANCE) ↩

The reported solution can be extracted from either the MIP or NLP solver result. If the MIP solution is returned only the primal values are available.

Default: `2`

value	meaning
1	Choose MIP solution if it is available.
2	Choose NLP solution if it is available.
3	Choose the solution with the best tolerance.
4	Choose the solution with the best objective value.

ECPstart *(integer)*: User specified start-point ↩

Define which variable levels are used when the optimization is started.

Default: `3`

value	meaning
0	Do not use a start-point; start the algorithm by solving the linear part (MIP) of the problem.

value	meaning
1	Use the user specified startpoint, but adjust the variable levels with a small value.
2	Use the exact start-point set by the user.
3	Use the exact start-point if linearly feasible; else adjust variable levels with a small value.

ECPstrategy *(integer)*: AlphaECP strategy ↩

Default: 2

value	meaning
1	Convex strategy. Ensures global optimality for problems with convex objective function and convex constraints.
2	Pseudo-convex constraints. Ensures global optimality for problems with convex objective function and pseudo-convex constraints.
3	Pseudo-convex objective. Ensures global optimality for problems with pseudo-convex objective function and pseudo-convex constraints. The reformulation of a non-linear objective function into a constraint must be done in a specific way. The requirement is that the objective variable must be in a linear part of the non-linear function. Assuming that the minimized or maximized variable is called objvar, the reformulation can be done as follows: (objective function expression) - objvar =E= 0. Furthermore, this strategy can effectively use a feasible start-point.
4	Pseudo-convex objective, but first complete with ECPstrategy 2. (Only the necessary linearizations are removed when the ECPstrategy is changed.)
5	Pseudo-convex objective, but find the first solution with ECPstrategy 2. (Only the necessary linearizations are removed when the ECPstrategy is changed.)

ECPtoltype *(integer)*: AlphaECP termination criterion ↩

Default: 1

value	meaning
1	Use termination based on tolerances TOLepsg and TOLepsf.
2	Terminate also if relative objective gap is satisfied according to TOLoptcr. The relative objective gap termination criterion also called relative termination criterion is only supported for convex problems when using ECPstrategy=1. The relative termination criterion can be used for non-convex problems, as well as, with ECPstrategy=2, however, in this case it may not work properly because the lower bound may be invalid.

MIPloglevel *(boolean)*: Level of MIP solver output ↩

By default the detailed log of the MIP solver is suppressed in the AlphaECP log stream. If this option is turned on the MIP log will be merged into the AlphaECP log.

Default: 0

value	meaning
0	No output.
1	MIP solver log goes to GAMS log.

MIPnrsols *(integer)*: Upper limit of considered MIP solutions per MIP call ↩

When the MIP solver returns several solutions the most suitable solution is chosen. Many times the solutions from the MIP solver are similar and a larger number might help to find a feasible MINLP solution if the constraints are almost satisfied. See MIPsolstrat to change the solution collection strategy.

Range: {1, ..., ∞}

Default: 50

MIPoptcr *(real)*: Relative MIP gap in intermediate sub-problems ↩

The relative stopping tolerance sent to the MIP solver for intermediate MIP problems. Note that the MIPoptcr value is decreased automatically to zero during the optimization.

Range: [0, 1]

Default: 1.0

MIPoptcrlim *(integer)*: Initial iteration limit when MIPoptcr is reduced ↩

The MIPoptcr parameter is reduced in steps: From 1 to 0.5 to 0.3 to 0.2 to 0.1 to 0.0. The first reduction is at iteration `MIPoptcrlim`. The following reductions occur also at iteration `MIPoptcrlim` but it is updated with a strategy defined by parameter MIPoptcrlimtype. Note that a step reduction may happen for other reasons.

Range: {1, ..., ∞}

Default: 200

MIPoptcrlimtype *(integer)*: Strategy to increase MIPoptcrlim ↩

Default: 0

value	meaning
0	Exponential increase of MIPoptcrlim. The parameter MIPoptcrlim is increased by multiplying it by 2. The increase occurs at iteration "MIPoptcrlim". If MIPoptcrlim=200 it will have values 200,400,800 etc.
1	Linear increase of MIPoptcrlim. The parameter MIPoptcrlim is increased by adding the original value of MIPoptcrlim to it. If MIPoptcrlim=200 it will have values 200,400,600 etc.

MIPoptimaliter *(integer)*: MIP is solved to optimality with this frequency ↩

Defines the frequency to solve the MIP problem to optimum before the algorithm sets MIPoptcr=0 and, hence, solves MIP problems to optimum. For example if `MIPoptimaliter`=n, then at least every nth iteration the MIP problem is solved to the optimum. Solving MIP problem to optimum yields a lower bound for a convex MINLP problem which can help to terminate the algorithm faster if ECPtoltype=2 is used. Non-default value is not recommended for non-convex problems nor if ECPstrategy >1.

Default: 0

value	meaning
0	Never.
n>0	Every nth iteration, where n is the given value.

MIPsolstrat *(integer)*: MIP solution collection strategy ↩

Default: 1

value	meaning
0	Instruct MIP solver to return only one solution.
1	Instruct MIP solver to return any solutions encountered during MIP procedure.
2	Instruct MIP solver to search for solutions to obtain requested number MIPnrsols solutions.
3	As 2, but also require the solutions to fulfill MIPoptcr >= 0.2.
4	Let AlphaECP decide.

MIPsolver *(string)*: MIP solver for sub-problems and . option file number ↩

`solver[.n]` Solver is the name of the GAMS MIP solver and `n` is the integer corresponding to optfile. The option file is appended to the option file, that is written by AlphaECP. Hence, the specified options take precedence over the options set by AlphaECP. If `.n` is missing, the optfile is treated as zero i.e. the MIP solver will not look for a options file specified by the user. This option can be used to overwrite the default that uses the MIP solver specified with an `Option NLP = solver;` statement or the default GAMS solver for NLP.

Default: `GAMS MIP solver`

NLPcall *(integer)*: NLP strategy ↩

Determine when the NLP solver is called.

Default: 5

value	meaning
0	No output.
1	Call the NLP solver at end of AlphaECP algorithm.
2	As 1 and when a better solution is found.
3	As 2 and when the same integer solution is encountered NLPlimsameint times.
4	Let AlphaECP decide.
5	Let AlphaECP decide and add noise to the variable levels before call.

NLPcalliter *(integer)*: NLP solver call at next (incremental) iteration ↩

Specify an iteration interval for the NLP solver calls.

Default: 0

NLPlimsameint *(integer)*: NLP call after a number of recurring integer solutions ↩

If the same integer solution is encountered `NLPlimsameint` times in a row then the NLP solver is called. The counter is reset after the NLP solver is called.

Range: {1, ..., ∞}

Default: 5

NLPloglevel *(boolean)*: Level of NLP solver output ↩

By default the detailed log of the NLP solver is suppressed in the AlphaECP log stream. If this option is turned on the NLP log will be merged into the AlphaECP log.

Default: `0`

value	meaning
`0`	No output.
`1`	NLP solver log goes to GAMS log.

NLPreslim *(real)*: NLP time limit per call ↩

The time limit (in seconds) given to the chosen NLP solver at each NLP solver call. Setting this option to 0 calculates a time limit which is relative to the problem size.

Default: `0`

NLPsolver *(string)*: NLP solver for sub-problems and . option file number ↩

`solver[.n]` Solver is the name of the GAMS NLP solver that should be used in the root node, and `n` is the integer corresponding to optfile. If `.n` is missing, the optfile is treated as zero, i.e., the NLP solver will not look for an options file. This option can be used to overwrite the default that uses the NLP solver specified with an `Option NLP = solver;` statement or the default GAMS solver for NLP.

Default: `GAMS NLP solver`

reslim *(real)*: Time limit for AlphaECP (in seconds) ↩

Default: `GAMS reslim`

solvelink *(integer)*: Solvelink for NLP and MIP sub-solver ↩

Default: `5`

value	meaning
`1`	Call NLP and MIP solver via script.
`2`	Call NLP and MIP solver via module.
`5`	Call NLP and MIP solver in memory.

solvetrace *(string)*: Filename of solvetrace file ↩

solvetracetime *(real)*: Time interval when a trace record is written ↩

Default: `1`

TOLepsf *(real)*: Pseudo-convex objective function termination tolerance ↩

Maximum allowed absolute difference between the nonlinear and the MIP objective function value (used only in ECPstrategy 3).

Range: [`1e-20`, `1`]

Default: `1e-3`

TOLepsg *(real)*: Constraint tolerance ↩

The nonlinear constraint tolerance defines the maximum value that a nonlinear constraint may violate. For example, a constraint required to be zero may hold a value +/- `TOLepsg` at a solution.

Range: [`1e-20`, `1`]

Default: `1e-3`

TOLepsz *(real)*: Distance tolerance for a new linearization ↩

The maximum perpendicular distance between a valid cutting plane and its generation point (MIP solution).

Range: [`1e-20`, `1`]

Default: `1e-1`

TOLgrad *(real)*: Gradient tolerance ↩

The absolute value of a gradient's partial derivative must be above `TOLgrad` value in order for it to be considered nonzero.

Range: [`1e-20`, `1`]

Default: `1e-6`

TOLinfbnd *(real)*: Infinity bound (MIP variable bound) ↩

All variables must have a positive and a negative finite bound in order to ensure a bounded MIP problem. The finite bound value `TOLinfbnd` will be applied to single or double unbounded variables.

Default: `1e10`

TOLoptcr *(real)*: Relative termination tolerance for MINLP ↩

The relative objective gap termination criterion is satisfied if |UB-LB|/ (10E-12+max(|LB|,|UB|)) < `TOLoptcr`, where UB is the current best upper bound and LB is the current best lower bound. Upper bounds are obtained from solving NLP problems and lower bounds are obtained from lower bounds of MIP problems. If the inequality holds true and ECPtoltype=2, the algorithm terminates.

Range: [`1e-20`, ∞]

Default: `GAMS optCR`

5.3.6 FAQ

- What are good settings to solve a convex problem?

  Use ECPmaster `1`.

- What are good settings if the solution speed is essential?

  Try ECPstrategy `1` and CUTdelcrit `1` to see if using multiple threads for the MIP solver improves the solution speed. However, there is considerable chance that a feasible solution for a non-convex problem with nonlinear equality constraints cannot be found.

- What are good settings when the solution quality is essential?

  Use NLPcalliter `1` and MIPsolstrat `4` or `3`, and also try different values, for CUTnrcuts option, for example `0.1`.

- The objective function is non-linear, should the default ECPstrategy be used?

  If the objective function constraint can be written in the required form of ECPstrategy `3` then this strategy may find a better solution. If the constraints and the objective function are pseudo-convex the global optimal solution will be found.

5.4 ANTIGONE

Author

Christodoulos A. Floudas, floudas@titan.princeton.edu; Computer-Aided Systems Laboratory; Department of Chemical and Biological Engineering; Princeton University

Ruth Misener, r.misener@imperial.ac.uk; Centre for Process Systems Engineering; Imperial College London

Date

16 April 2013

5.4.1 Introduction

ANTIGONE, Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations, is a deterministic general mixed-integer nonlinear global optimization framework [134] [135] [137] [136] [138] .

MINLP is defined:

$$\begin{aligned} \min \quad & f_0(\boldsymbol{x},\, \boldsymbol{y},\, \boldsymbol{z}) \\ \text{s.t.} \quad & b_m^{\text{LO}} \leq f_m(\boldsymbol{x},\, \boldsymbol{y},\, \boldsymbol{z}) \leq b_m^{\text{UP}} \quad \forall\, m \in \{1,\, \ldots,\, M\} \\ & \boldsymbol{x} \in \mathbb{R}^C;\; \boldsymbol{y} \in \{0,\, 1\}^B;\; \boldsymbol{z} \in \mathbb{Z}^I \end{aligned} \tag{MINLP}$$

where C, B, I, and M represent the number of continuous variables, binary variables, integer variables, and constraints, respectively. Parameters vectors $\boldsymbol{b_m^{\text{LO}}}$ and $\boldsymbol{b_m^{\text{UP}}}$ bound the constraints. We assume that it is possible to infer finite bounds $\left[x_i^L,\, x_i^U\right]$ on the variables participating in nonlinear terms f_m and that the image of f_m is finite on $\boldsymbol{x}$. Typical expressions for $f_0(\boldsymbol{x},\, \boldsymbol{y},\, \boldsymbol{z})$ and $f_m(\boldsymbol{x},\, \boldsymbol{y},\, \boldsymbol{z})$ are:

$$\begin{aligned} f_m(\boldsymbol{x},\, \boldsymbol{y},\, \boldsymbol{z}) = c_m + a_m^T\, [\boldsymbol{x};\; \boldsymbol{y};\; \boldsymbol{z}] + [\boldsymbol{x};\; \boldsymbol{y};\; \boldsymbol{z}]^T\, Q_m\, [\boldsymbol{x};\; \boldsymbol{y};\; \boldsymbol{z}] \\ + \sum_{s=1}^{S_m} c_{s_m} \cdot \prod_{c=1}^{C} x_c^{p_{s_m,\,c}} + \sum_{e=1}^{E_m} c_{e_m} \cdot e^x + \sum_{\ell=1}^{L_m} c_{\ell_m} \cdot \log x \end{aligned}$$

where the powers $p_{s_m,\,c}$ are constant reals; c_m, a_m, Q_m, c_{s_m}, c_{e_m}, c_{ℓ_m} are constant coefficients; S_m, E_m, L_m are the number of signomial, exponential, and logarithmic terms, respectively.

As illustrated in Figure 5.1, ANTIGONE responds dynamically to exploit special structure within (MINLP). ANTIGONE falls broadly into the category of branch-and-bound global optimization because it: generates and solves convex relaxations of the nonconvex MINLP that rigorously bound the global solution; finds feasible solutions via local optimization; divides and conquers the feasible set to generate a sequence of convex relaxations converging to the global optimum [68] [67] .

5.4.1.1 Licensing and software requirements

Using GAMS/ANTIGONE requires

1. an ANTIGONE license,
2. a CPLEX license, and
3. a CONOPT or SNOPT license.

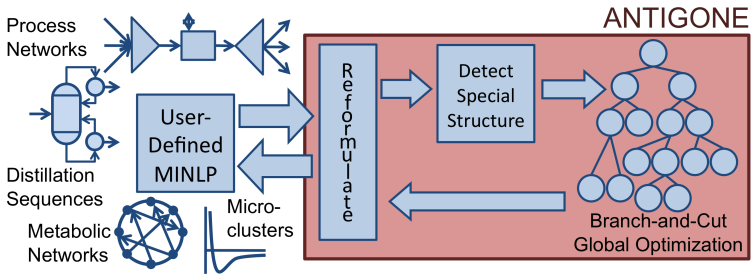


Figure 5.1 Given an MINLP optimization problem, ANTIGONE reformulates the model, detects special structure in the reformulated MINLP, solves the optimization problem, and returns the model with respect to the original problem variables

5.4.1.2 Running GAMS/ANTIGONE

GAMS/ANTIGONE solves: `NLP`; `MINLP`; `RMINLP`; `QCP`; `MIQCP`; `RMIQCP`; `CNS`. If GAMS/ANTIGONE is not the default solver for these models, it can be called using the following command before the `solve` statement:

```
option nlp=antigone, minlp=antigone, rminlp=antigone;
```

5.4.2 GAMS/ANTIGONE Output

The log output shown below is generated using the MINLP model `cecil_13` from the MINLPLib.

```
--------------------------------------------------------------------------------
ANTIGONE: Algorithms for coNTinuous/Integer Global Optimization; Version 1.0

          Ruth Misener and Christodoulos A. Floudas

          Computer-Aided Systems Laboratory (CASL)
          Department of Chemical & Biological Engineering; Princeton University
--------------------------------------------------------------------------------

Before Pre-processing:
          840 Variables
                 660 Continuous
                 180 Binary
          929 Equations

After Pre-processing:
          520 Variables
                 418 Continuous
                 102 Binary
          499 Equations
                 291 Linear
                 208 Nonconvex nonlinear
          232 Nonlinear Terms
                 232 Signomial
          730 Possible Reformulation Linearization Technique (RLT) equations
             34 RLT Equations Added Outright to Formulation

Constituent Libraries:
```

```
            CPLEX  Solving relaxations
            CONOPT Finding feasible points
            LAPACK Addressing linear systems
            Boost  Bounding Intervals

-------------------------------------------------------------------------------
Time (s) Nodes explored Nodes remaining Best possible   Best found Relative Gap
-------------------------------------------------------------------------------

      65              1               1    -1.158e+05   -1.157e+05   +1.032e-03
     134              1               1    -1.157e+05   -1.157e+05   +4.337e-04
     202              1               1    -1.157e+05   -1.157e+05   +4.334e-04
     258              1               1    -1.157e+05   -1.157e+05   +4.140e-06
     Solving MILP relaxation at tree level 0 ----------------------------------
     341              1               1    -1.157e+05   -1.157e+05   +4.091e-06
     413              1               1    -1.157e+05   -1.157e+05   +4.011e-06
     Solving MILP relaxation at tree level 0 ----------------------------------
     483              1               1    -1.157e+05   -1.157e+05   +4.001e-06
     571              1               1    -1.157e+05   -1.157e+05   +3.959e-06
     Solving MILP relaxation at tree level 0 ----------------------------------
     640              1               1    -1.157e+05   -1.157e+05   +3.880e-06
     Solving MILP relaxation at tree level 0 ----------------------------------
     702              1               0    -1.157e+05   -1.157e+05   +1.000e-06

-------------------------------------------------------------------------------
Termination Status : Global minimum
Best Feasible Point: -1.156565e+05
Best Possible Point: -1.156566e+05
       Relative Gap: +1.000000e-06
Algorithm analysis :

                  0 Nodes explored
                  0 Nodes remaining

                  0 Maximum tree depth

                183 Cutting Planes (183 Globally Valid)
                      183 Signomial

              702.38 Total time (CPU s)
                       0.07 Pre-processing
                     698.56 Solving MILP relaxations
                       0.95 Searching for feasible solutions
                       2.79 Variable bounds tightening
                              2.31 OBBT
                              1.48 FBBT (0.13 EC; 0.92 RLT; 0.00 Factoring)
                       0.00 Branching
                              0.00 Reliability branching
-------------------------------------------------------------------------------
```

5.4.3 Summary of ANTIGONE Options

5.4.3.1 General Options

Option	Description	Default
abs_opt_tol	absolute stopping tolerance	GAMS optca
dumpsolutions	name of solutions index gdx file for writing alternate solutions	
max_number_nodes	node limit	GAMS nodlim
max_time	resource limit	GAMS reslim
readparams	read secondary option file in ANTIGONE syntax	
rel_opt_tol	relative stopping tolerance	GAMS optcr
trydual	call CONOPT or SNOPT to produce duals	5

5.4.3.2 Options for Solving the MILP Relaxations

Option	Description	Default
cplex_optfile	read a secondary GAMS/CPLEX options file that will be applied to every LP and MILP subsolve	
cut_generation_epsilon	absolute violation threshold for separating hyperplanes	1e-4
nominal_time_limit	nominal time limit for solving MILP subproblems	100
populate_solution_pool	emphasis on generating starting points	3

5.4.3.3 Options for Finding Feasible Solutions

Option	Description	Default
conopt_optfile	read a secondary GAMS/CONOPT options file that will be applied to every NLP subsolve	
feas_soln_time_limit	time limit (s) for an NLP solve	30
feas_tolerance	absolute feasibility tolerance	1e-6
nlp_solver	use CONOPT or SNOPT to find feasible solutions	conopt

5.4.3.4 Options for Branching

Option	Description	Default
branching_bounds_push_away	branch a minimum fraction away from the variable bounds	0.1
branching_weight	branch on a convex combination of midpoint and solution	0.25
num_reliability_tests	number of strong branching initialization tests	8
reliability_branching	heuristic choice for building reliable pseudocosts	error
reliability_branching_mu	score parameter for building reliability	0.15
use_reliability_branching	use reliability branching?	1

5.4.3.5 Options for Bounding

Option	Description	Default
fbbt_improvement_bound	bounds reduction improvement threshold needed to exit FBBT loop	0.999
max_fbbt_iterations	maximum number of FBBT iterations	50
max_obbt_iterations	maximum number of OBBT iterations	30
max_time_each_obbt	time limit (s) for each OBBT LP	10
obbt_improvement_bound	bounds reduction improvement threshold	0.95
use_obbt	use optimality-based bounds tightening?	1

5.4.3.6 Options for Logging to the Console

Option	Description	Default
logging_freq	how often should we log progress to the console?	5
logging_level	logging information level	-1
print_options	print the option parameter choices used in a single run?	1

5.4.3.7 Options for Addressing Special Structure

Option	Description	Default
adaptive_add_rlt	use the dynamic approach to adaptively determine deep RLT cuts?	1
adaptive_add_rlt_tree_depth	tree depth for heuristic that adaptively determines deep RLT cuts	3
add_bilinear_terms	allow addition of nonconvex bilinear terms to generate deep RLT cuts	1
convexity_cuts	derive convexity-based separating cuts for multivariable terms?	1
dominant_ec_only	add only the low-dimension edge-concave aggregations introducing dominant cuts into relaxations?	1
eigenvector_projections	use eigenvector projections as additional cuts?	1
eigenvector_projection_partitioning	allow partitioning on eigenvector projections?	1
low_dim_edge_concave_agg	use low-dimension edge-concave aggregations?	1
max_partitioned_quantities	number of partitioned quantities	0
max_rlt_cuts	maximum number of violated RLT cuts to add before resolving the relaxation?	100
naive_add_ec	naively integrate all low-dimension edge-concave aggregations into relaxations?	0
naive_add_rlt	naively add all RLT cuts to the relaxations?	0
number_of_partitions	how many partitions per variable?	1

Option	Description	Default
partitioning_scheme	Partitioning scheme can be linear or logarithmic	linear
piecewise_linear_partitions	use piecewise-linear partitioning?	0
rlt	find RLT variable/equation and equation/equation pairs?	1
use_alpha_bb	apply globally-valid alphaBB cuts to tighten a node relaxation	1
use_edge_concave_dynamic	apply locally-valid edge-concave cuts to tighten a node relaxation	1

In addition, GAMS option threads specifies the number of processors to use for linear algebra routines, e.g., when computing eigenvalues of a quadratic coefficients matrix. By default, all available processors are used.

5.4.4 Detailed Descriptions of ANTIGONE Options

abs_opt_tol *(real)*: absolute stopping tolerance ↩

Default: `GAMS optca`

adaptive_add_rlt *(boolean)*: use the dynamic approach to adaptively determine deep RLT cuts? ↩

In the first few levels of the branch-and-bound tree, query the RLT equations after solving an initial relaxation. Add violated equations to the relaxation and resolve. Track the most commonly-violated equations and include those cuts in later nodes.

Default: `1`

adaptive_add_rlt_tree_depth *(integer)*: tree depth for heuristic that adaptively determines deep RLT cuts ↩

To the specified tree depth, solve the relaxation of a node twice if RLT equations are violated. After this depth, automatically add the most commonly violated cuts to the solution of each node

Range: {`1`, ..., `100`}

Default: `3`

add_bilinear_terms *(boolean)*: allow addition of nonconvex bilinear terms to generate deep RLT cuts ↩

Default: `1`

branching_bounds_push_away *(real)*: branch a minimum fraction away from the variable bounds ↩

Range: [`0`, `0.5`]

Default: `0.1`

branching_weight *(real)*: branch on a convex combination of midpoint and solution ↩

The branching weight specifies the emphasis on the midpoint of a variable, so larger branching weights imply branching closer to the center of a variable range.

Range: [0, 1]

Default: 0.25

conopt_optfile *(string)*: read a secondary GAMS/CONOPT options file that will be applied to every NLP subsolve ↩

Gain direct access to the GAMS/CONOPT options. The value of the string should match the name of the GAMS/CONOPT options file.

convexity_cuts *(boolean)*: derive convexity-based separating cuts for multivariable terms? ↩

Default: 1

cplex_optfile *(string)*: read a secondary GAMS/CPLEX options file that will be applied to every LP and MILP subsolve ↩

Gain direct access to the GAMS/CPLEX options. Specifying an options file allows, for example, the possibility of running the CPLEX subsolver with multiple threads. The value of the string should match the name of the GAMS/CPLEX options file.

cut_generation_epsilon *(real)*: absolute violation threshold for separating hyperplanes ↩

Absolute violation threshold to generate separating hyperplanes for convex multivariable terms

Range: [1e-7, 10]

Default: 1e-4

dominant_ec_only *(boolean)*: add only the low-dimension edge-concave aggregations introducing dominant cuts into relaxations? ↩

Default: 1

dumpsolutions *(string)*: name of solutions index gdx file for writing alternate solutions ↩

The GDX file specified by this option will contain a set call `index` that contains the names of GDX files with the individual solutions. For details see example model **dumpsol** in the GAMS Test Library.

eigenvector_projections *(boolean)*: use eigenvector projections as additional cuts? ↩

Default: 1

eigenvector_projection_partitioning *(boolean)*: allow partitioning on eigenvector projections? ↩

Default: 1

fbbt_improvement_bound *(real)*: bounds reduction improvement threshold needed to exit FBBT loop ↩

Range: [0, 1]

Default: 0.999

feas_soln_time_limit *(real)*: time limit (s) for an NLP solve ↩

Range: [1, ∞]

Default: `30`

feas_tolerance *(real)*: absolute feasibility tolerance ↩

Default: `1e-6`

logging_freq *(real)*: how often should we log progress to the console? ↩

Wait at least the specified time in seconds before next output to the console

Range: [1, ∞]

Default: `5`

logging_level *(integer)*: logging information level ↩

Log to the console at the specified level (-1: default; 0: minimal logging; 3: extensive logging)

Default: `-1`

value	meaning
-1	minimal plus warnings
0	minimal
1	entering info
2	updating info
3	includes Cplex updates

low_dim_edge_concave_agg *(boolean)*: use low-dimension edge-concave aggregations? ↩

Default: 1

max_fbbt_iterations *(integer)*: maximum number of FBBT iterations ↩

Range: {1, ..., 100}

Default: 50

max_number_nodes *(integer)*: node limit ↩

Default: GAMS nodlim

max_obbt_iterations *(integer)*: maximum number of OBBT iterations ↩

Range: {1, ..., 100}

Default: 30

max_partitioned_quantities *(integer)*: number of partitioned quantities ↩

Range: {0, ..., 50}

Default: 0

max_rlt_cuts *(integer)*: maximum number of violated RLT cuts to add before resolving the relaxation? ↩

Range: {1, ..., 1000}

Default: 100

max_time *(real)*: resource limit ↩

Default: GAMS reslim

max_time_each_obbt *(real)*: time limit (s) for each OBBT LP ↩

Range: [1, 100]

Default: 10

naive_add_ec *(boolean)*: naively integrate all low-dimension edge-concave aggregations into relaxations? ↩

Default: 0

naive_add_rlt *(boolean)*: naively add all RLT cuts to the relaxations? ↩

Default: 0

nlp_solver *(string)*: use CONOPT or SNOPT to find feasible solutions ↩

Note, that independent of the setting for this option, for the initial NLP solve from the user provided starting point, always CONOPT is used, if available. Further, for the final NLP solve (see trydual), always CONOPT is used, if available, otherwise SNOPT is used.

Default: conopt

value	meaning
conopt	Conopt
snopt	Snopt

nominal_time_limit *(real)*: nominal time limit for solving MILP subproblems ↩

Nominal time limit for solving MILP subproblems. Terminate long-running MILP subproblems over this time limit once they reach an integer feasible point

Range: [0.1, 1000]

Default: 100

number_of_partitions *(integer)*: how many partitions per variable? ↩

Range: {0, ..., 16}

Default: 1

num_reliability_tests *(integer)*: number of strong branching initialization tests ↩

Range: {1, ..., 100}

Default: 8

obbt_improvement_bound *(real)*: bounds reduction improvement threshold ↩

Bounds reduction improvement threshold needed to exit OBBT loop This parameter also determines whether to continue obbt in child; if the parent bound improvement is less than this threshold, then child node won't try OBBT

Range: [0, 1]

Default: 0.95

partitioning_scheme *(string)*: Partitioning scheme can be linear or logarithmic ↩

Linear partitioning uses a number of binary variables linear in the number of partitions while logarithmic partitioning uses a number of binary variables logarithmic in the number of breakpoints. Linear partitioning tends to be numerically favorable for a few breakpoints while logarithmic partitioning is better for a larger number of breakpoints.

Default: linear

value	meaning
linear	Linear partitioning
logarithmic	Logarithmic partitioning

piecewise_linear_partitions *(boolean)*: use piecewise-linear partitioning? ↩

Default: 0

populate_solution_pool *(integer)*: emphasis on generating starting points ↩

Emphasis on generating many starting points for NLP solves using the CPLEX solution pool feature. Larger number implies more starting points.

Range: {0, ..., 4}

Default: 3

print_options *(boolean)*: print the option parameter choices used in a single run? ↩

Default: 1

readparams *(string)*: read secondary option file in ANTIGONE syntax ↩

reliability_branching *(string)*: heuristic choice for building reliable pseudocosts ↩

Default: error

value	meaning
error	Max Error Branching
forward	Forward branching
reverse	Reverse branching

reliability_branching_mu *(real)*: score parameter for building reliability ↩

Range: [0, 1]

Default: 0.15

rel_opt_tol *(real)*: relative stopping tolerance ↩

Default: GAMS optcr

rlt *(boolean)*: find RLT variable/equation and equation/equation pairs? ↩

Default: 1

trydual *(real)*: call CONOPT or SNOPT to produce duals ↩

Spend the specified amount of time in seconds or less in producing a dual solution by calling CONOPT or SNOPT.

Range: [0, ∞]

Default: 5

use_alpha_bb *(boolean)*: apply globally-valid alphaBB cuts to tighten a node relaxation ↩

Default: 1

use_edge_concave_dynamic *(boolean)*: apply locally-valid edge-concave cuts to tighten a node relaxation ↩

Default: 1

use_obbt *(boolean)*: use optimality-based bounds tightening? ↩

Default: 1

use_reliability_branching *(boolean)*: use reliability branching? ↩

Default: 1

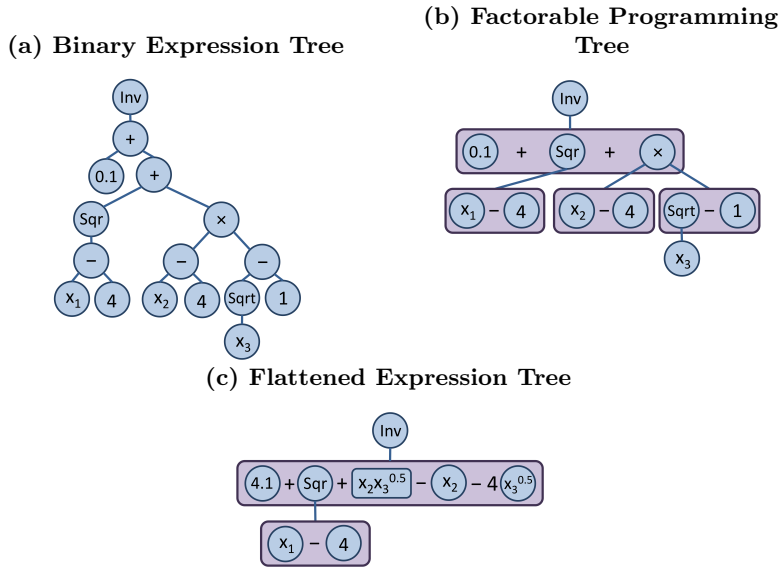


Figure 5.2 (a) A binary tree represents algebraic expressions; (b) A factorable programming tree employs operator-based relaxations; (c) The ANTIGONE transformation to a flattened expression tree allows access to term-based underestimators; Convex form $(x_1 - 4)^2$ is not expanded.

5.4.5 ANTIGONE Algorithmic Features

As illustrated in Figure 5.1, the primary algorithmic features in ANTIGONE are reformulating model input, elucidating special structure, and branch-and-bound global optimization [134] [135] [137] [136] [138] .

5.4.5.1 Reformulating Model Input

As illustrated in Figure 5.2, ANTIGONE transforms a factorable programming tree into a flattened expression tree to capitalize on the development of tight convex underestimators for specific classes of nonlinear terms. ANTIGONE extends the efficacy of hybrid strategies by meaningfully integrating mutually reinforcing operator- and term-based strategies [27] [76] [136] . This approach reformulates towards multivariable terms with specialized underestimators while maintaining a tree-like representation of powers that cannot be distributed and convex operators that can be exploited by dynamic cut generation.

5.4.5.2 Elucidating Special Structure

After reformulating the user-defined MINLP, ANTIGONE detects special mathematical structure that it will exploit in the branch-and-cut phase (Section Branch-and-Bound Global Optimization). The types of special structure that ANTIGONE considers are: reformulation-linearization technique (RLT) equations; convexity/concavity; edge-convexity/edge-concavity; αBB relaxations; term-specific underestimators [134] [135] [137] [136] [138] .

- **RLT** multiplies every pairwise combination of: variables; nonlinear terms; equations [11] [119] [135] [136] [138] [169] [170] [171] [172] [173] . ANTIGONE saves the combinations that do *not* introduce new terms into the model formulation and updates these equations at every node of the branch-and-cut tree. Special RLT equations are added directly to the model formulation; other RLT equations are used as cutting planes and integrated into the feasibility-based bounds tightening routines.

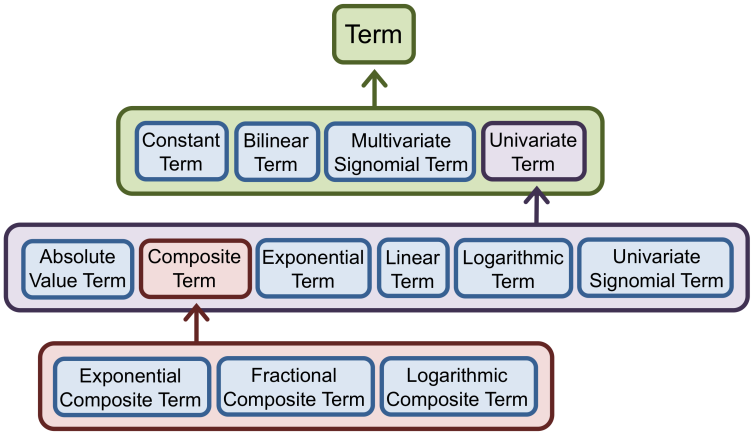


Figure 5.3 Inheritance Structure of Base Class Term

- **Convexity/Concavity** permits the easy generation of a cutting plane at a point $\hat{x}$:

$$f(x) \geq f(\hat{x}) + f'(x) \cdot (x - \hat{x}) \quad \text{(convex)}$$
$$f(x) \leq f(\hat{x}) + f'(x) \cdot (x - \hat{x}) \quad \text{(concave)}$$

  Based on interval arithmetic, terms and multi-term expressions are labeled as always/sometimes/never convex/ concave; this information is used in the branch-and-cut phase.

- **Edge-Convexity/Edge-Concavity** implies a vertex polyhedral envelope; ANTIGONE labels terms and multi-term expressions as always/sometimes/never edge-convex/edge-concave with a simple interval arithmetic test [133] [175] [176] [177] .

- **αBB** underestimators convexify an expression with univariate quadratics [5] [6] [10] [66] [128]; ANTIGONE uses αBB to relax aggregates of bilinear terms.

- **Term-Specific Underestimators** are diagrammed in Figure 5.3; their implementation is based on work available in the open literature [53] [52] [65] [77] [118] [125] [123] [124] [129] [133] [134] [146] [175] [176] [178] .

5.4.5.3 Branch-and-Bound Global Optimization

After the reformulation and special structure detection phases, ANTIGONE initiates a branch-and-cut global optimization algorithm that generates tight convex underestimators, dynamically generates separating hyperplanes, bounds the variables [6] [10] [5] [11] [18] [49] [50] [116] [157] [158] [172] [173] [191]; branches on the search space [3] [18], and finds feasible solutions.

5.5 BARON

Author

Nick Sahinidis, The Optimization Firm, LLC, niksah@minlp.com, http://www.minlp.com

Date

5 June 2015

5.5.1 Introduction

The Branch-And-Reduce Optimization Navigator (BARON) is a GAMS solver for the *global solution* of nonlinear (NLP) and mixed-integer nonlinear programs (MINLP).

While traditional NLP and MINLP algorithms are guaranteed to converge only under certain convexity assumptions, BARON implements deterministic global optimization algorithms of the branch-and-bound type that are *guaranteed to provide global optima* under fairly general assumptions. These include the existence of finite lower and upper bounds on nonlinear expressions in the NLP or MINLP to be solved.

BARON implements algorithms of the branch-and-bound type enhanced with a variety of constraint propagation, interval analysis, and duality techniques for reducing ranges of variables in the course of the algorithm. Rigorous relaxations are constructed by enlarging the feasible region and/or underestimating the objective function.

Parts of the BARON software were created at the University of Illinois at Urbana-Champaign. The algorithms implemented in the software, the theory behind them, and some related applications are (partly) described in [157] [51] [158] [161] [93] [120] [78] [187] [168] [188] [163] [79] [159] [178] [179] [180] [183] [184] [165] [160] [162] [7] [181] [164] [182] [36] [12] [149] [13] [106] [107] [208] [209] [108] [210] [14] .

5.5.1.1 Licensing and software requirements

In order to use GAMS/BARON, users will need to have a GAMS/BARON license. BARON comes with several embedded LP/MIP/QP and NLP solvers (CBC; IPOPT, FilterSD, FilterSQP). Additionally, GAMS/BARON users can expedite convergence by accessing CPLEX and XPRESS to solve BARON's LP/MIP/QP subproblems and MINOS, SNOPT, and any GAMS NLP solver, such as CONOPT, to solve BARON's NLP subproblems. These solvers need to be licensed with GAMS separately.

By default, GAMS/BARON will attempt to use CPLEX as the LP solver and select an NLP solver automatically. The user can use the options LPSol and NLPSol to specify the LP/MIP/QP and NLP solver, respectively. If the user does not have a license for the user-specified solver, BARON will automatically select a licensed one and may default to CLP/CBC and IPOPT if no other LP/MIP and NLP solver is available, respectively. BARON can be used without a local NLP solver by setting DoLocal and NumLoc to 0.

5.5.1.2 Running GAMS/BARON

BARON is capable of solving models of the following types: LP, MIP, RMIP, NLP, DNLP, RMINLP, and MINLP. If BARON is not specified as the default solver for these models, it can be invoked by issuing the following command before the solve statement:

```
option <modeltype>=baron;
```

where <`modeltype`> stands for `LP`, `MIP`, `RMIP`, `QCP`, `MIQCP`, `RMIQCP`, `CNS`, `NLP`, `DNLP`, `MINLP`, or `RMINLP`.

5.5.2 Model requirements

In order to achieve convergence to global optimality, additional model constraints may be required. The additional constraints may speed up solver time and increase the probability of success.

5.5.2.1 Variable and expression bounds

All nonlinear expressions in the mathematical program to be solved must be bounded below and/or above. It is important that finite lower and upper bounds be provided by the user on all problem variables. Note that providing finite bounds on variables alone may not suffice to guarantee finite bounds on nonlinear expressions arising in the model.

For example, consider the term `1/x` for $\texttt{x} \in \texttt{[0,1]}$, which has finite variable bounds, but is unbounded. It is important to provide bounds for problem variables that guarantee that the problem functions are finitely-valued. If the user model does not include variable bounds that guarantee that all nonlinear expressions are finitely-valued, BARON will attempt to infer appropriate bounds from problem constraints. If this step fails, global optimality of the solutions provided is not guaranteed. Occasionally, because of the lack of bounds no numerically stable lower bounding problems can be constructed, in which case BARON may terminate.

See section Some BARON features on how to specify variable bounds.

5.5.2.2 Allowable nonlinear functions

In addition to multiplication and division, GAMS/BARON can handle nonlinear functions that involve `exp(x)`, `ln(x)`, $\texttt{x}^{\alpha}$ for real α, $\beta^{\texttt{x}}$ for real β, $\texttt{x}^{\texttt{y}}$, and $|\texttt{x}|$. Currently, there is no support for other functions, including the trigonometric functions `sin(x)`, `cos(x)`, etc.

5.5.3 BARON output

5.5.3.1 BARON log output

The log output below is obtained for the MINLP model `gear.gms` from the GAMS model library using absolute and relative optimality tolerances of 1e-6.

```
================================================================================
 BARON version 15.6.5. Built: LNX-64 Fri Jun 5 08:34:09 EDT 2015

 If you use this software, please cite:
 Tawarmalani, M. and N. V. Sahinidis, A polyhedral
 branch-and-cut approach to global optimization,
 Mathematical Programming, 103(2), 225-249, 2005.

 BARON is a product of The Optimization Firm, LLC. http://www.minlp.com/
 Parts of the BARON software were created at the
 University of Illinois at Urbana-Champaign.
================================================================================
 This BARON run may utilize the following subsolver(s)
 For LP/MIP: ILOG CPLEX
 For NLP: MINOS, SNOPT, GAMS external NLP, COIN IPOPT with MUMPS and METIS, FILTERSD
================================================================================
 Starting solution is feasible with a value of     36.1767610000
 Doing local search
 Solving bounding LP
 Starting multi-start local search
 Preprocessing found feasible solution with value  4.23791612465
 Done with local search
================================================================================
```

```
  Iteration    Open nodes         Time (s)    Lower bound      Upper bound
          1             1             0.02     1.00000          4.23792
*         1+            1             0.02     1.00000          3.29321
*         1+            1             0.02     1.00000          2.20487
          1             1             0.02     1.00000          2.20487
*         2             2             0.02     1.00000          1.06987
*         3+            1             0.03     1.00000          1.01273
*         4+            2             0.03     1.00000          1.00117
*         4+            2             0.03     1.00000          1.00018
*         4             2             0.03     1.00000          1.00004
*        14             8             0.04     1.00000          1.00001
*        30             0             0.05     1.00000          1.00000
         30             0             0.05     1.00000          1.00000

 Cleaning up

                        *** Normal completion ***

 Wall clock time:                    0.05
 Total CPU time used:                0.05

 Total no. of BaR iterations:      30
 Best solution found at node:      30
 Max. no. of nodes in memory:      11

 All done
==============================================================================
```

The solver first tests feasibility of the user-supplied starting point. This point is found to be feasible with an objective function value of 36.1767610000. BARON subsequently performs a randomized local search procedure and, eventually, finds a feasible solution with an objective of 4.23791612465. Then, the iteration log provides information every 1,000,000 branch-and-bound iterations and every 30 seconds. Additionally, information is printed at the end of the root node, whenever the value of the incumbent is improved by at least 10^{-5}, and at the end of the search. A star ($*$) in the first position of a line indicates that a better feasible solution was found that improves the value of previous best known solution by at least 10^{-5}. The log fields include the iteration number, number of open branch-and-bound nodes, the CPU time taken thus far, the lower bound, and the upper bound for the problem. The log output fields are summarized below:

Field	Description
`Itn. no.`	The number of the current iteration. A plus (`+`) following the iteration number denotes reporting while solving a probing (as opposed to a relaxation) subproblem of the corresponding node.
`Open Nodes`	Number of open nodes in branch-and-reduce tree.
`Time`	Current computational time in seconds. CPU time is reported for single-threaded jobs and wall clock time is reported for multi-threaded jobs.
`Lower Bound`	Current lower bound on the model.
`Upper Bound`	Current upper bound on the model.

Once the branch-and-reduce tree is searched, the best solution is isolated and a corresponding dual solution is calculated. Finally, the total number of branch-and-reduce iterations (number of search tree nodes) is reported, followed by the node where the best solution was identified (a `-1` indicates preprocessing as explained in the next section on termination messages).

5.5.3.2 Termination messages, model and solver statuses

Upon normal termination, BARON will report the node where the optimal solution was found. We refer to this node as `nodeopt`. Associated with this node is a return code indicating the status of the solution found at `nodeopt`. The return code is given in the log line:

```
Best solution found at node:  (return code)
```

The return codes have the following interpretation:

-3 : no feasible solution found

-2 : the best solution found was the user-supplied

-1 : the best solution was found during preprocessing

`i` : the best solution was found in the `i`-th node of the tree

In addition to reporting `nodeopt`, upon termination, BARON will issue one of the following statements:

`*** Normal completion ***`

This is the most desirable termination status. The problem has been solved within tolerances in this case. If BARON returns a code of `-3`, then no feasible solution exists.

`*** Heuristic termination ***`

While global optimality is not guaranteed in this case, BARON will terminate with this message when (a) a feasible solution has been found and (b) the progress of lower/upper bounds satisfies the heuristic termination criterion set by the user through BARON's DeltaTerm option.

`*** User did not provide appropriate variable bounds ***`

The user will need to read the BARON output (in file `sum.dat` in the scratch directory, use GAMS parameter `keep=1` to prevent the automatic removal of this directory) for pointers to variables and expressions with missing bounds. The model should be modified in order to provide bounds for variables and intermediate expressions that make it possible for BARON to construct reliable relaxations. Even though relaxation bounds are printed on the screen to give the user a feeling for convergence, these bounds may not be valid for the problem at hand. This message is followed by one of the following two messages:

`***Infeasibility is therefore not guaranteed ***`

This indicates that, because of missing bounds, no feasible solution was found, but model infeasibility was not proven.

`*** Globality is therefore not guaranteed ***`

This indicates that, because of missing bounds, a feasible solution was found, but global optimality was not proven.

`*** Max.  allowable nodes in memory reached ***`

The user will need to increase the physical memory of the computer or change algorithmic options, such as branching and node selection rules, to reduce the size of the search tree and memory required for storage.

*** Max. allowable BaR iterations reached ***

The user will need to increase the maximum number of allowable iterations. The BARON option is MaxIter. To specify this in GAMS, one can use the NodLim option. We remark that the BARON option MaxIter overrides NodLim.

*** Max. allowable CPU time exceeded ***

The user will need to increase the maximum of allowable CPU time. The BARON option is MaxTime. To specify this in GAMS, one can use the ResLim option. We remark that the BARON option MaxTime overrides ResLim.

*** Problem is numerically sensitive ***

BARON is designed to automatically handle problems with numerical difficulties. However, for certain problems, the global optimum is numerically sensitive. This occurs, for instance, when the objective function value varies significantly over small neighborhoods of points that are strictly outside the feasible region but nonetheless feasible within numerical tolerances. When BARON returns this message, the "Best possible" reported on the objective is likely correct.

*** Search interrupted by user ***

The run was interrupted by the user (Ctrl-C).

*** Insufficient memory for data structures ***

More memory is needed to set up the problem data structures. The user will need to increase the physical memory available on the computer in order to accommodate problems of this size.

*** A potentially catastrophic access violation just took place ***

In the unlikely event of a memory access violation, BARON will terminate the search and return the best known solution. Please report problems that lead to this condition to Nick Sahinidis (niksah@minlp.com).

5.5.4 Some BARON features

The features described in this section rely on options that are further detailed in the next section. For details of the algorithmic implementations, the user may wish to consult publications cited at the end of this document.

5.5.4.1 No starting point is required

In contrast to many NLP algorithms that require a feasible starting point, a starting point is not required for BARON. A user may optionally provide a starting point for all or even some of the problem variables. BARON will judiciously initialize any variables that are not initialized by the user. Even when the problem functions cannot be evaluated at a user-provided starting point, BARON is still capable of carrying out its global search.

For problems for which GAMS compilation is aborted because the nonlinear functions cannot be evaluated at the starting point, the user can use the following commands before the SOLVE statement:

```
MaxExecError = 100000;
option sys12 = 1;
```

The first command asks GAMS to continue compilation for as many as MaxExecError execution errors. The sys12 option will pass the model to the BARON despite the execution errors. Even though the starting point is bad in this case, BARON is capable of carrying out its global search.

5.5.4.2 Finding a few of the best or all feasible solutions

BARON offers a facility, through its NumSol option, to find the best few, or even all feasible, solutions to a model. The development of this facility was motivated by combinatorial optimization problems but the facility is applicable to continuous problems as well. Even for combinatorial problems, BARON does not rely on integer cuts to find multiple solutions. Instead, it utilizes a single search tree, thus providing a computationally efficient method for finding multiple solutions. Furthermore, because BARON's approach applies to integer as well as continuous programs, it can be used to find *all* feasible solutions to a system of nonlinear equality and inequality constraints.

Once a model is solved by BARON with the NumSol option, the solutions found can be recovered using the GAMS GDX facility. An example is provided below.

```
$eolcom !
$Ontext
  Purpose: demonstrate use of BARON option 'numsol' to obtain the best
  numsol solutions of an optimization problem in a single branch-and-bound
  search tree.

  The model solved here is a linear general integer problem with 18 feasible
  solutions.  BARON is run with a request to find up to 20 solutions.  The
  model solved is the same as the one solved in gamslib/icut.gms.
$Offtext

set i index of integer variables  / 1 * 4 /

variables x(i) variables
          z    objective variable

integer variable x;

x.lo(i) = 2; x.up(i) = 4; x.fx('2') = 3;   ! fix one variable
x.up('4') = 3;   ! only two values

equation obj obj definition;

* pick an objective function which will order the solutions

obj .. z =e= sum(i, power(10,card(i)-ord(i))*x(i));

model enum / all /;

* instruct BARON to return numsol solutions
$onecho > baron.opt
numsol 20
gdxout multsol
$offecho

enum.optfile=1; option mip=baron, limrow=0, limcol=0, optca=1e-5,
optcr=1e-5; solve enum minimizing z using mip;

* recover BARON solutions through GDX
set sol /multsol1*multsol100/; variables xsol(sol,i), zsol(sol);

execute 'gdxmerge multsol*.gdx >  %gams.scrdir%merge.%gams.scrext%';
execute_load 'merged.gdx', xsol=x, zsol=z;
```

```
option decimals=8;

display xsol.l, zsol.l;
```

5.5.4.3 Using BARON as a multi-start heuristic solver

To gain insight into the difficulty of a nonlinear program, especially with regard to existence of multiple local solutions, modelers often make use of multiple local searches from randomly generated starting points. This can be easily done with BARON's NumLoc option, which determines the number of local searches to be done by BARON's preprocessor. BARON can be forced to terminate after preprocessing by setting the number of iterations to 0 through the MaxIter option. In addition to local search, BARON's preprocessor performs extensive reduction of variable ranges. To sample the search space for local minima without range reduction, one would have to set to 0 the range reduction options TDo, MDo, LBTTDo, and OBTTDo. On the other hand, leaving these options to their default values increases the likelihood of finding high quality local optima during preprocessing. If NumLoc is set to -1, local searches in preprocessing will be done from randomly generated starting points until global optimality is proved or MaxTime seconds have elapsed.

5.5.4.4 Systematic treatment of unbounded problems

If BARON declares a problem as unbounded, it will search for and may report a vertex and direction of an unbounded ray. In addition, BARON will report the best solution found. This will be a feasible point that is as far as possible along an unbounded ray while avoiding numerical errors due to floating point arithmetic.

5.5.4.5 Systematic treatment of infeasible problems

If BARON declares a problem as infeasible, it has the capability to identify a subset of the constraints that are infeasible and become feasible once any one of them is eliminated. This, so-called, *irreducibly inconsistent system* (IIS) can be obtained by BARON for all types of problems handled by BARON, including linear and nonlinear, continuous and integer, convex and nonconvex, and problems with complementarity constraints. BARON's CompIIS option can be used to identify an IIS.

As an example, consider the problem of minimizing the nonconvex function x_1x_3 over the following nonconvex constrained set:

$$
\begin{aligned}
e_1: \quad & 85 + 0.006x_2x_5 + 0.0006x_1x_4 - 0.002x_3x_5 <= 92 \\
e_2: \quad & 0.8x_2x_5 + 0.003x_1x_2 + 0.002x_3^2 = 110 \\
e_3: \quad & 9 + 0.005x_3x_5 + 0.001x_1x_3 + 0.002x_3x_4 <= 25 \\
& 78 \leq x_1 \leq 102 \\
& 33 \leq x_2 \leq 45 \\
& 27 \leq x_i \leq 45, \quad i = 3, \ldots, 5
\end{aligned}
$$

When this problem is solved with CompIIS equal to 1, BARON provides the following result in the listing file:

```
Number of equations in the IIS: 1.
Upper: e_2 <= 110

Number of variables in the IIS: 3.
Lower: x_1 >= 78
Lower: x_2 >= 33
Lower: x_5 >= 27
```

The IIS consists of the lower bounds of variables x_1, x_2, and x_5, along with the $\leq$ part of the equality constraint e_2. This suggests that constraint e_2 and the entire model can be made feasible by lowering the lower bound of any of the three variables that are part of the IIS, whereas modifying the bounds of x_3 would not make the model feasible.

If a problem is known to be infeasible and the user desires to identify an IIS, it may be beneficial to set BARON's NumLoc option to zero. Doing so will deactivate BARON's initial upper bounding search, which involves multiple local searches. On the other hand, DoLocal should be nonzero in order to permit local search during the solution of certain subproblems that BARON solves while searching for an IIS.

5.5.4.6 Handling of complementarity constraints

Complementarity relationships of the type `f(x)g(x) = 0` are automatically recognized and exploited algorithmically by BARON. The functions `f` and `g` may be univariate or multivariate, linear or nonlinear, convex or nonconvex, in terms of continuous and/or integer variables, and may be subject to additional constraints in the model. These complementarity relationships can be inferred by BARON even when implied by problem constraints and variable bounds. As a result, BARON can solve general mathematical programs with equilibrium constraints (MPECs). This class of problems includes the classical linear complementarity problem

(LCP): Find $z \geq 0$ and q such that

$$\begin{aligned} Mz + q &\geq 0, \\ z^t(Mz + q) &= 0 \end{aligned}$$

as well as the more general mixed complementarity problem

(MCP): Given a function $f : \mathbb{R}^n \to \mathbb{R}^n$ and bounds $l, u \in \mathbb{R}^n$ with $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$, find $z \in \mathbb{R}^n$ and $w, v \in \mathbb{R}^n_+$ such that

$$\begin{aligned} f(z) &= w - v, \\ l \leq z &\leq u, \\ (z - l)^t w &= 0, \\ (u - z)^t v &= 0. \end{aligned}$$

Both problems are automatically recognized and exploited by BARON without the user having to mark complementarities in any special way. In GAMS, all these problems can be solved by BARON when declared as NLP or MINLP models.

5.5.4.7 Parallel capabilities

For difficult problems with integer variables, most of BARON's time is spent on solving MIP relaxations. Hence, considerable speedups may be obtained via parallel solution of the MIP subproblems. For this purpose, the option threads may be used to specify the number of cores that BARON's MIP subsolver is allowed to use. By default, this option has the value of 1, meaning that a single core will be utilized.

5.5.5 The BARON options

The BARON options allow the user to control termination tolerances, branching and relaxation strategies, heuristic local search options, and output options as detailed in this section.

Many options can also be set in the GAMS model. The most relevant GAMS options are ResLim, NodLim, OptCA, OptCR, OptFile, and CutOff. The IterLim option is not implemented. specify BARON iterations, the user can set the MaxIter option, which is equivalent to the GAMS option NodLim.

Additionally, a BARON Options file can be provided. See section The Solver Options File for general use of solver option files.

For branching, users can specify separate branching priorities for any discrete and continuous variables using Dot Options. To specify variable branching priorities, one specifies

(variable).`prior` *(value)*

in the BARON options file, where *(value)* can be any non-negative real value. The lower the value, the higher the priority for branching, see also Setting Priorities for Branching. Specifying `maxdouble` for *(value)* translates to passing 0 as branching priority for *(variable)*.

5.5.5.1 Termination options

Option	Description	Default
AbsConFeasTol	Absolute constraint feasibility tolerance This tolerance is used for general constraints and variable bounds. `AbsConFeasTol` must be $\geq$ 1e-12. A point is considered feasible for a constraint/bound if the absolute or relative constraint feasibility tolerance is satisfied. Range: [`1e-12`, ∞]	`1e-5`
AbsIntFeasTol	Absolute integer feasibility tolerance All integer variable values must satisfy this tolerance. `AbsIntFeasTol` must be $\geq$ 1e-12. A point is considered integer feasible for a variable if integrality is satisfied using the absolute or relative integer feasibility tolerance. Range: [`1e-12`, ∞]	`1e-5`
BoxTol	Box elimination tolerance. Boxes will be eliminated if smaller than this tolerance. `BoxTol` must be $\geq$ 1e-12.	`1e-8`
CutOff	Eliminate solutions that are no better than this value Can also be used with the GAMS model attribute option CutOff.	`GAMS CutOff`
DeltaA	Absolute improvement for insufficient progress termination `DeltaA` (δ_a) must be $\geq$ 1e-12. Range: [`1e-12`, ∞]	∞

Option	Description	Default
DeltaR	Relative improvement for insufficient progress termination `DeltaR` (δ_r) must be $\geq$ 1e-12. Range: [`1e-12`, ∞]	`1`
DeltaT	Time interval for insufficient progress termination If DeltaTerm is set to 1, BARON will terminate if insufficient progress is made over `DeltaT` (δ_t) consecutive seconds. If δ_t is set to a non-positive quantity, BARON will automatically set δ_t equal to -δ_t times the CPU time taken till the end of root node processing. `DeltaT` can take any real value. Range: [-∞, ∞]	`-100`
DeltaTerm	Indicates whether insufficient progress termination is on or off Users have the option to request BARON to terminate if insufficient progress is made over DeltaT (δ_t) consecutive seconds. Progress is measured using the absolute and relative improvement thresholds DeltaA (δ_a) and DeltaR (δ_r). Termination will occur if, over a period of δ_t consecutive seconds, the value of the best solution found by BARON is not improved by at least an absolute amount δ_a or an amount equal to δ_r times the value of the incumbent at time `t`-δ_t. This termination condition is enforced after processing the root node and only after a feasible solution has been obtained. Because it relies on CPU time measurements, which may depend on machine load, this option may result in nondeterministic behavior. `0`: do not enforce this termination condition `1`: terminate if progress is insufficient	`0`
EpsA	Absolute termination tolerance. BARON terminates if |`U-L`| $\leq$ `EpsA`, where `U` and `L` are the upper and lower bound, respectively, on the optimal value of the optimization problem at the current iteration. `EpsA` must be $\geq$ 1e-12.	GAMS OptCA
EpsR	Relative termination tolerance. BARON terminates if |`U-L`| $\leq$ `EpsR`|`U`|, where `U` and `L` are the objective function value of the incumbent and the lower bound (if minimizing, otherwise upper bound) on the optimal value, respectively, for the optimization problem at the current iteration.	GAMS OptCR
FirstFeas	Changes the search for first numsol solutions If set to 1, BARON will terminate once it finds NumSol feasible solutions, irrespective of solution quality. `0`: search for the *best* NumSol feasible solutions `1`: find NumSol solutions irrespective of solution quality	`0`
FirstLoc	Terminate the search as soon as a local optimum is found	`0`
ISolTol	Solution Distance Separation distance between solutions. This option is used in conjunction with NumSol. For combinatorial optimization problems, feasible solutions are isolated. For continuous problems, feasible solution points within an l_∞ distance that does not exceed `IsolTol` will be treated as identical by BARON. `IsolTol` must be $\geq$ 1e-12.	`1e-4`
MaxIter	Maximum number of branch-and-reduce iterations allowed Setting `MaxIter` to 0 will force BARON to terminate after root node preprocessing. Setting `MaxIter` to 1 will result in termination after the solution of the root node. `MaxIter` must be $\geq$ -1, where -1 implies unlimited. Range: {`-1`, ..., ∞}	GAMS NodeLim

Option	Description	Default
MaxTime	Maximum time allowed (sec) `MaxTime` must be -1 or > 0, where -1 implies unlimited. For single-threaded jobs, i.e., when threads equals 1, this limit is enforced on CPU time consumed by the job. For multi-threaded jobs, the limit is enforced on wall clock time.	GAMS ResLim
NumSol	Number of feasible solutions to be found Solutions found will be listed in the `results` file. As long as `NumSol` ≠ -1, these solutions will be sorted from best to worse. If `NumSol` is set to -1, BARON will search for all feasible solutions to the given model and print them, in the order in which they are found, in the `results` file. `NumSol` must be ≥ -1. Range: {-1, ..., ∞}	1
RelConFeasTol	Relative constraint feasibility tolerance This tolerance is used for general constraints and variable bounds. `RelConFeasTol` must be ≥ 0. A point is considered feasible for a constraint/bound if the absolute or relative constraint feasibility tolerance is satisfied. Range: [0, 0.1]	0
RelIntFeasTol	Relative integer feasibility tolerance All integer variable values must satisfy this tolerance. `RelIntFeasTol` must be ≥ 0. A point is considered integer feasible for a variable if integrality is satisfied using the absolute or relative integer feasibility tolerance. Range: [0, 0.1]	0

5.5.5.2 Relaxation options

Option	Description	Default
Nouter1	Number of outer approximators of convex univariant functions `NOuter1` must be ≥ 0.	4
NoutIter	Number of rounds of cutting plane generation at node relaxation `NOutIter` must be a ≥ 0.	4
NoutPerVar	Number of outer approximations per variable for convex multivariate functions `NOutPerVar` must be ≥ 0.	4
OutGrid	Number of grid points per variable for convex multivariate approximators of BARON's `CONVEX_EQUATIONS`. `OutGrid` must be a ≥ 0.	20
Threads	Number of cores used for solution of MIP subproblems The value of this option is passed to CBC, CPLEX, and XPRESS. GAMS option threads set to 0 corresponds to `Threads=1`. Range: {1, ..., ∞}	GAMS Threads

5.5.5.3 Range reduction options

Option	Description	Default
LBTTDo	Linear-feasibility-based range reduction option (poor man's LPs) `0`: no range reduction based on feasibility. `1`: range reduction done based on feasibility.	1
MDo	Marginals-based reduction option `0`: no range reduction based on marginals. `1`: range reduction done based on marginals.	1

Option	Description	Default
OBTTDo	Optimality based tightening option 0: no range reduction based on optimality. 1: range reduction done based on optimality.	1
PDo	Number of probing problems allowed -2: automatically decided by BARON. -1: probing on all variables. 0: no range reduction by probing. n: probing on n variables.	-2
TDo	Nonlinear-feasibility-based range reduction option (poor man's NLPs) 0: no bounds tightening is performed. 1: bounds tightening is performed.	1

5.5.5.4 Tree management options

Option	Description	Default
BrPtStra	Branching point selection strategy 0: BARONs dynamic strategy 1: w-branching 2: bisection-branching 3: convex combination of 1 and 2	0
BrVarStra	Branching variable selection strategy 0: BARONs dynamic strategy 1: largest violation 2: longest edge	0
NodeSel	Specifies the node selection rule to be used for exploring the search tree 0: BARONs mixed selection scheme 1: best bound 2: last in first out [LIFO] 3: minimum infeasibility	0

5.5.5.5 Local search options

Option	Description	Default
DoLocal	Local search option for upper bounding 0: no local search is done during upper bounding 1: BARON's dynamic local search decision rule	1
NumLoc	Number of local searches done in preprocessing The first local search begins with the user-specified starting point. Subsequent local searches are done from judiciously chosen starting points. If NumLoc is set to -1, local searches in preprocessing will be done until proof of globality or MaxTime is reached. If NumLoc is set to -2, BARON decides the number of local searches in preprocessing based on problem and NLP solver characteristics. NumLoc must be $\geq$ -2. Range: {-2, ..., ∞}	-2

5.5.5.6 Output Options

Option	Description	Default
LocRes	Option to control output from local search 0: no local search output 1: detailed results from local search will be printed to res.dat file	0
prfreq	Log output frequency in number of nodes	1000000
prlevel	Defines the level of log output printed. 0: all log output is suppressed 1: print log output	1
prtimefreq	Log output frequency in number of seconds	30

5.5.5.7 Subsolver Options

Option	Description	Default
AllowCbc	Indicator for use of CLP/CBC with automatic LP/MIP/QP solver selection	1
AllowCplex	Indicator for use of CPLEX with automatic LP/MIP/QP solver selection	1
AllowExternal	Indicator for use of External NLP solver with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of external GAMS NLP solver as an NLP subsolver. 0: do not use the GAMS external NLP solver for local search 1: consider the GAMS external NLP solver for local search	1
AllowFilterSD	Indicator for use of FILTERSD with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of FILTERSD as an NLP subsolver. 0: do not use FILTERSD for local search 1: consider FILTERSD for local search	1
AllowFilterSQP	Indicator for use of FILTERSQP with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of FILTERSQP as an NLP subsolver. 0: do not use FILTERSQP for local search 1: consider FILTERSQP for local search	1
AllowHsl	Indicator for use of HSL LA04 with automatic LP solver selection	1
AllowIpopt	Indicator for use of IPOPT with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of IPOPT as an NLP subsolver. 0: do not use IPOPT for local search 1: consider IPOPT for local search	0 on macOS/ARM64, 1 otherwise

Option	Description	Default
AllowMinos	Indicator for use of MINOS with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of MINOS as an NLP subsolver. 0: do not use MINOS for local search 1: consider MINOS for local search	1
AllowSnopt	Indicator for use of SNOPT with automatic NLP solver selection In case of automatic NLP solver selection, this option can be used to selectively permit or disallow the use of SNOPT as an NLP subsolver. 0: do not use SNOPT for local search 1: consider SNOPT for local search	1
AllowXpress	Indicator for use of FICO XPRESS with automatic LP/MIP/QP solver selection	0 on macOS/ARM64, 1 otherwise
ExtNLPsolver	External GAMS NLP solver and option file (e.g. conopt.1) Specifies the GAMS NLP solver to be used when NLPSol is set to -1 or 6. All GAMS NLP solvers are available through this option. If a non-existing solver is specified or the solver chosen cannot solve NLPs, NLPSol will be reset to its default. A GAMS solver options file can be specified for the GAMS NLP solver by adding a dot followed by the options file number to the solver name, e.g., setting ExtNLPSolver to CONOPT.42 would instruct GAMS/CONOPT to use options file conopt.o42.	conopt
LPAlg	Specifies the LP algorithm to be used (available only with CPLEX as the LP solver) 0: automatic selection of LP algorithm 1: primal simplex 2: dual simplex 3: barrier	0
LPSol	Specifies the LP/MIP/QP Solver to be used By default, BARON will select the LP solver and may switch between different LP solvers during the search according to problem characteristics and solver performance. The solvers CPLEX, if available, and CBC will be used for this purpose. A single specific LP solver can be specified by setting this option to a value other than the default. If the specified solver is not licensed, BARON will default to automatic solver selection. -1: Automatic LP/MIP/QP solver selection and switching strategy 3: CPLEX 7: XPRESS 8: CLP/CBC 15: HSL LA04	-1

Option	Description	Default
NLPSol	Specifies the NLP solver to be used By default, BARON will select the NLP solver and may switch between different NLP solvers during the search, based on problem characteristics and solver performance. Any combination of licensed NLP solvers may be used in that case. A single specific NLP solver can be specified by setting this option to a value other than the default. If the specified solver is not licensed, BARON will default to automatic solver selection. -1: Automatic NLP solver selection and switching strategy 0: Local search based on function evaluations alone with no calls to local solvers 2: MINOS 4: SNOPT 6: GAMS NLP solver (see ExtNLPsolver) 9: IPOPT 10: FILTERSD 14: FILTERSQP	-1

5.5.5.8 Other Options

Option	Description	Default
CompIIS	Request the computation of an Irreducible Inconsistent Set (IIS) In case of an infeasible problem, this option can be used to search for an IIS. Setting this option 1, works very well for most problems. 0: do not search for an IIS 1: the search for an IIS is based on a fast heuristic 2: an IIS is obtained using a deletion filtering algorithm 3: an IIS is obtained using an addition filtering algorithm 4: an IIS is obtained using an addition-deletion filtering algorithm 5: an IIS is obtained using a depth-first search algorithm	0
IISInt	Indicates whether general integers should be considered as potential members of the IIS When search for an IIS is requested through CompIIS, BARON assumes that the model is unlikely to include an error in terms of binaries, i.e., the binary definitions are assumed correct and the IIS output should be interpreted with respect to binary definitions. General integer bounds may be assumed as correct or can be questioned using the option `IISint`. Integrality is enforced in both cases. 0: do not consider general integers as part of an IIS, assume them to be correct 1: consider general integers (but not binaries) as part of an IIS	0
IISOrder	Order in which constraints are considered in the search for an IIS -1: auto set to aim for a small IIS depending on the value of CompIIS 1: arrange constraints in problem order 2: arrange constraints in ascending order of degree 3: arrange constraints in descending order of degree >3: random order using `IISorder` as seed	-1
WantDual	whether to try to provide dual solution values 0: BARON may or may not return a dual solution. 1: BARON will return a dual solution using an inexpensive technique to solve a KKT system corresponding to the best primal solution identified	1

5.5.5.9 Interface and Conversion

Option	Description	Default
BarName	Name of BARON problem file to be written	225a/mybaron.dat
ClockType	Type of clock to use when reporting solving time back to GAMS wall: report time according to "clock on the wall" (as used by most GAMS solver links) cpu: report time used by CPU (summed up over all cores) baron: report same time as used by BARON ("cpu" if one thread, "wall" if multiple threads)	wall
.EquClass	Equation Classification Specifies nature of named constraint in the users model. This is a Dot Option. Slices like supply.EquClass("new-york") 1 are allowed. 0: Regular constraint. 1: Relaxation-only constraint. These constraints are provided to BARON as RELAXATION_ONLY_EQUATIONS and used to help strengthen the relaxation bound but are not considered as part of the user model and thus not used for feasibility testing of solutions or local search. Adding, for instance, the first-order optimality conditions as relaxation-only constraints often expedites convergence. 2: Convex constraint. These constraints are provided to BARON as CONVEX_EQUATIONS and used to generate cutting planes from the set of outer approximating supporting hyperplanes of the convex constraint set. 3: Convex constraint that is relaxation-only.	0
GDXOut	Prefix for GDX file names for multiple solutions if NumSol > 1.	
InfBnd	infinity value to be used on bounds If set to 0, then no bounds are used.	0

5.6 CBC

CBC (COIN-OR Branch and Cut) is an open-source mixed integer programming solver working with the COIN-OR LP solver CLP and the COIN-OR Cut generator library Cgl. The code has been written primarily by John J. Forrest. Most of the CBC documentation in the section was copied from the help in the CBC standalone version.

The CBC link in GAMS supports continuous, binary, integer, semicontinuous, semiinteger variables, special ordered sets of type 1 and 2, and branching priorities.

5.6.1 Usage

The following statement can be used inside your GAMS program to specify using CBC

```
Option LP = CBC;      { or MIP or RMIP }
```

The above statement should appear before the Solve statement. If CBC was specified as the default solver during GAMS installation, the above statement is not necessary.

For usage and syntax of solver options file, see Section The Solver Option File. Following is an example options file cbc.opt.

```
cuts root
perturbation off
```

It will cause CBC to use cut generators only in the root node and turns off the perturbation of the LP relaxation.

GAMS/CBC currently does not support the GAMS Branch-and-Cut-and-Heuristic (BCH) Facility. If you need to use GAMS/CBC with BCH, please consider to use a GAMS system of version $\leq$ 23.3.

The following GAMS parameters are currently supported by GAMS/CBC: reslim, iterlim, nodlim, optca, optcr, cheat, cutoff, and threads.

5.6.2 List of Options

There are many parameters which can affect the performance the CBCs Branch and Cut Algorithm. First just try with default settings and look carefully at the log file. Did cuts help? Did they take too long? Look at the output to see which cuts were effective and then do some tuning (see the option cuts). If the preprocessing reduced the size of the problem or strengthened many coefficients then it is probably wise to leave it on. Switch off heuristics which did not provide solutions. The other major area to look at is the search. Hopefully good solutions were obtained fairly early in the search so the important point is to select the best variable to branch on. See whether strong branching did a good job – or did it just take a lot of iterations? Adjust the options strongBranching and trustpseudocosts.

In the following, we summarize all available CBC options.

5.6.2.1 General Options

Option	Description	Default
clocktype	type of clock for time measurement	wall
reslim	maximum seconds	GAMS reslim
special	options passed unseen to CBC	
writemps	create MPS file for problem	

5.6.2.2 LP Options

Option	Description	Default
autoScale	Whether to scale objective, rhs and bounds of problem if they look odd (experimental)	0
biasLU	Whether factorization biased towards U	LX
bscale	Whether to scale in barrier (and ordering speed)	off
crash	Whether to create basis for problem	off
crossover	Whether to get a basic solution with the simplex algorithm after the barrier algorithm finished	on
denseThreshold	Threshold for using dense factorization	-1
dualPivot	Dual pivot choice algorithm	automatic

Option	Description	Default
factorization	Which factorization to use	normal
gamma	Whether to regularize barrier	off
idiotCrash	Whether to try idiot crash	-1
iterlim	Maximum number of iterations before stopping	GAMS iterlim
KKT	Whether to use KKT factorization in barrier	0
maxFactor	Maximum number of iterations between refactorizations	200
passPresolve	How many passes in presolve	5
perturbation	Whether to perturb the problem	1
presolve	Whether to presolve problem	on
primalPivot	Primal pivot choice algorithm	automatic
primalWeight	Initially algorithm acts as if it costs this much to be infeasible	1e+10
psi	Two-dimension pricing factor for Positive Edge criterion	-0.5
randomSeedClp	Random seed for Clp	1234567
scaling	Whether to scale problem	automatic
smallFactorization	Threshold for using small factorization	-1
sparseFactor	Whether factorization treated as sparse	1
sprintCrash	Whether to try sprint crash	-1
startalg	LP solver for root node	dual
substitution	How long a column to substitute for in presolve	3
tol_dual	For an optimal solution no dual infeasibility may exceed this value	1e-07
tol_presolve	Tolerance to use in presolve	1e-08
tol_primal	For a feasible solution no primal infeasibility, i.e., constraint violation, may exceed this value	1e-07

5.6.2.3 MIP Options

Option	Description	Default
costStrategy	How to use costs for branching priorities	off
cutoff	Bound on the objective value for all solutions	GAMS cutoff
cutoffConstraint	Whether to use cutoff as constraint	off
dumpsolutions	name of solutions index gdx file for writing alternate solutions	
dumpsolutionsmerged	name of gdx file for writing all alternate solutions	
expensiveStrong	Whether to do even more strong branching	0
extraVariables	Allow creation of extra integer variables	0

Option	Description	Default
fixOnDj	Try heuristic based on fixing variables with reduced costs greater than this	-1
increment	A valid solution must be at least this much better than last integer solution	GAMS cheat
infeasibilityWeight	Each integer infeasibility is expected to cost this much	0
loglevel	amount of output printed by CBC	1
maxsol	Maximum number of solutions to save	100
mipstart	whether it should be tried to use the initial variable levels as initial MIP solution	0
multipleRootPasses	Do multiple root passes to collect cuts and solutions	0
nodeStrategy	What strategy to use to select the next node from the branch and cut tree	fewest
nodlim	node limit	GAMS nodlim
optca	Stop when gap between best possible and best less than this	GAMS optca
optcr	Stop when gap between best possible and best known is less than this fraction of larger of two	GAMS optcr
OrbitalBranching	Whether to try orbital branching	off
parallelmode	whether to run opportunistic or deterministic	deterministic
preprocess	Whether to use integer preprocessing	sos
printfrequency	frequency of status prints	0
randomSeedCbc	Random seed for Cbc	-1
sollim	Maximum number of feasible solutions to get	∞
solvefinal	final solve of MIP with fixed discrete variables	1
solvetrace	name of trace file for solving information	
solvetracenodefreq	frequency in number of nodes for writing to solve trace file	100
solvetracetimefreq	frequency in seconds for writing to solve trace file	5
sosPrioritize	How to deal with SOS priorities	off
strategy	Switches on groups of features	1
strongBranching	Number of variables to look at in strong branching	5
threads	Number of threads to try and use	GAMS threads
tol_integer	For a feasible solution no integer variable may be more than this away from an integer value	1e-07
trustPseudoCosts	Number of branches before we trust pseudocosts	10

5.6.2.4 MIP Options for Cutting Plane Generators

Option	Description	Default
cliqueCuts	Whether to use Clique cuts	ifmove
conflictcuts	Conflict Cuts	0

Option	Description	Default
cut_passes_root	Number of rounds that cut generators are applied in the root node	20 or 100
cut_passes_slow	Maximum number of rounds for slower cut generators	10
cut_passes_tree	Number of rounds that cut generators are applied in the tree	10
cutDepth	Depth in tree at which to do cuts	-1
cutLength	Length of a cut	-1
cuts	Switches all cut generators on or off	on
flowCoverCuts	Whether to use Flow Cover cuts	ifmove
gomoryCuts	Whether to use Gomory cuts	ifmove
gomorycuts2	Whether to use alternative Gomory cuts	off
knapsackCuts	Whether to use Knapsack cuts	ifmove
lagomoryCuts	Whether to use Lagrangean Gomory cuts	off
latwomirCuts	Whether to use Lagrangean TwoMir cuts	off
liftAndProjectCuts	Whether to use Lift and Project cuts	off
mirCuts	Whether to use Mixed Integer Rounding cuts	ifmove
probingCuts	Whether to use Probing cuts	ifmove
reduceAndSplitCuts	Whether to use Reduce-and-Split cuts	off
reduceAndSplitCuts2	Whether to use Reduce-and-Split cuts - style 2	off
residualCapacityCuts	Whether to use Residual Capacity cuts	off
twoMirCuts	Whether to use Two phase Mixed Integer Rounding cuts	root
zeroHalfCuts	Whether to use zero half cuts	ifmove

5.6.2.5 MIP Options for Primal Heuristics

Option	Description	Default
combine2Solutions	Whether to use crossover solution heuristic	off
combineSolutions	Whether to use combine solution heuristic	off
Dins	Whether to try Distance Induced Neighborhood Search	off
diveSolves	Diving solve option	100
DivingCoefficient	Whether to try Coefficient diving heuristic	off
DivingFractional	Whether to try Fractional diving heuristic	off
DivingGuided	Whether to try Guided diving heuristic	off
DivingLineSearch	Whether to try Linesearch diving heuristic	off
DivingPseudoCost	Whether to try Pseudocost diving heuristic	off
DivingRandom	Whether to try Diving heuristics	off

Option	Description	Default
DivingVectorLength	Whether to try Vectorlength diving heuristic	off
dwHeuristic	Whether to try Dantzig Wolfe heuristic	off
feaspump	Whether to try the Feasibility Pump heuristic	on
feaspump_artcost	Costs ≥ this treated as artificials in feasibility pump	0
feaspump_cutoff	Fake cutoff for use in feasibility pump	0
feaspump_fracbab	Fraction in feasibility pump	0.5
feaspump_increment	Fake increment for use in feasibility pump	0
feaspump_passes	How many passes to do in the Feasibility Pump heuristic	20
greedyHeuristic	Whether to use a greedy heuristic	on
heuristics	Switches most primal heuristics on or off	1
hOptions	Heuristic options	0
localTreeSearch	Whether to use local tree search when a solution is found	0
naiveHeuristics	Whether to try some stupid heuristic	off
pivotAndComplement	Whether to try Pivot and Complement heuristic	off
pivotAndFix	Whether to try Pivot and Fix heuristic	off
proximitySearch	Whether to do proximity search heuristic	off
randomizedRounding	Whether to try randomized rounding heuristic	off
Rens	Whether to try Relaxation Enforced Neighborhood Search	off
Rins	Whether to try Relaxed Induced Neighborhood Search	off
roundingHeuristic	Whether to use simple (but effective) Rounding heuristic	on
VndVariableNeighborhoodSearch	Whether to try Variable Neighborhood Search	off
vubheuristic	Type of VUB heuristic	-1

5.6.3 Detailed Options Description

In the following, we give a detailed description of all available CBC options.

autoScale: Whether to scale objective, rhs and bounds of problem if they look odd (experimental) ↩

Range: boolean

Default: 0

biasLU: Whether factorization biased towards U ↩

Range: UU, UX, LX, LL

Default: LX

bscale: Whether to scale in barrier (and ordering speed) ↩

Range: off, on, off1, on1, off2, on2

Default: off

cliqueCuts: Whether to use Clique cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglClique

Range: off, on, root, ifmove, forceOn, onglobal

Default: ifmove

clocktype: type of clock for time measurement ↩

value	meaning
cpu	CPU clock
wall	Wall clock

Default: wall

combine2Solutions: Whether to use crossover solution heuristic ↩

This heuristic does branch and cut on the problem given by fixing variables which have the same value in two or more solutions. It obviously only tries after two or more solutions. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before

Default: off

combineSolutions: Whether to use combine solution heuristic ↩

This heuristic does branch and cut on given problem by just using variables which have appeared in one or more solutions. It is obviously only tried after two or more solutions have been found. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
onquick	
bothquick	
beforequick	
~~0~~	~~Same as off. This is a deprecated setting.~~
1	Same as on. This is a deprecated setting.

Default: off

conflictcuts: Conflict Cuts ↩

Equivalent to setting cutoffconstraint=conflict

Range: boolean

Default: 0

costStrategy: How to use costs for branching priorities ↩

Value 'priorities' assigns highest priority to variables with largest absolute cost. This primitive strategy can be surprisingly effective. Value 'columnorder' assigns the priorities 1, 2, 3, ... with respect to the column ordering. Value '01first' ('01last') assignes two sets of priorities such that binary variables get high (low) priority. Value 'length' assigns high priority to variables that occur in many equations.

value	meaning
off	
priorities	
columnOrder	
01first	
binaryfirst	This is a deprecated setting. Please use 01first.
01last	
binarylast	This is a deprecated setting. Please use 01last.
length	
singletons	
nonzero	
generalForce	

Default: off

crash: Whether to create basis for problem ↩

If crash is set to 'on' and there is an all slack basis then Clp will flip or put structural variables into the basis with the aim of getting dual feasible. On average, dual simplex seems to perform better without it and there are alternative types of 'crash' for primal simplex, e.g. 'idiot' or 'sprint'. A variant due to Solow and Halim which is as 'on' but just flips is also available.

Range: off, on, solow_halim, lots, idiot1, idiot2, idiot3, idiot4, idiot5, idiot6, idiot7

Default: off

crossover: Whether to get a basic solution with the simplex algorithm after the barrier algorithm finished ↩

Interior point algorithms do not obtain a basic solution. This option will crossover to a basic solution suitable for ranging or branch and cut.

value	meaning
on	
off	
presolve	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: on

cut_passes_root: Number of rounds that cut generators are applied in the root node ↩

The default is to do 100 passes if the problem has less than 500 columns, 100 passes (but stop if the drop in the objective function value is small) if the problem has less than 5000 columns, and 20 passes otherwise. A negative value -n means that n passes are also applied if the objective does not drop.

Range: {-∞, ..., ∞}

Default: 20 or 100

Synonym: passCuts

cut_passes_slow: Maximum number of rounds for slower cut generators ↩

Some cut generators are fairly slow - this limits the number of times they are tried. The cut generators identified as 'may be slow' at present are Lift and project cuts and both versions of Reduce and Split cuts.

Range: {-1, ..., ∞}

Default: 10

Synonym: slowcutpasses

cut_passes_tree: Number of rounds that cut generators are applied in the tree ↩

The default is to do one pass. A negative value -n means that n passes are also applied if the objective does not drop.

Range: {-∞, ..., ∞}

Default: 10

Synonym: passTreeCuts

cutDepth: Depth in tree at which to do cuts ↩

Cut generators may be off, on, on only at the root node, or on if they look useful. Setting this option to a positive value K let CBC call a cutgenerator on a node whenever the depth in the tree is a multiple of K. The default of -1 lets CBC decide.

Range: {-1, ..., ∞}

Default: -1

cutLength: Length of a cut ↩

At present this only applies to Gomory cuts. -1 (default) leaves as is. Any value >0 says that all cuts ≤ this length can be generated both at root node and in tree. 0 says to use some dynamic lengths. If value ≥10,000,000 then the length in tree is value%10000000 - so 10000100 means unlimited length at root and 100 in tree.

Range: {-1, ..., ∞}

Default: -1

cutoff: Bound on the objective value for all solutions ↩

All solutions must have a better objective value than the value of this option. CBC also updates this value whenever it obtains a solution to the value of the objective function of the solution minus the cutoff increment.

Range: real

Default: GAMS cutoff

cutoffConstraint: Whether to use cutoff as constraint ↩

For some problems, cut generators and general branching work better if the problem would be infeasible if the cost is too high. If this option is enabled, the objective function is added as a constraint which right hand side is set to the current cutoff value (objective value of best known solution)

value	meaning
off	
on	
variable	
forcevariable	
conflict	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

Synonym: constraintfromCutoff

cuts: Switches all cut generators on or off ↩

This can be used to switch on or off all cut generators (apart from Reduce and Split). Then one can turn individual ones off or on. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node.

Range: off, on, root, ifmove, forceOn

Default: on

Synonym: cutsOnOff

denseThreshold: Threshold for using dense factorization ↩

If processed problem $\leq$ this use dense factorization

Range: {-1, ..., 10000}

Default: -1

Dins: Whether to try Distance Induced Neighborhood Search ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
often	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

diveSolves: Diving solve option ↩

If >0 then do up to this many solves. However, the last digit is ignored and used for extra options: 1-3 enables fixing of satisfied integer variables (but not at bound), where 1 switches this off for that dive if the dive goes infeasible, and 2 switches it off permanently if the dive goes infeasible.

Range: {-1, ..., 200000}

Default: 100

DivingCoefficient: Whether to try Coefficient diving heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

DivingFractional: Whether to try Fractional diving heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

DivingGuided: Whether to try Guided diving heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

DivingLineSearch: Whether to try Linesearch diving heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

DivingPseudoCost: Whether to try Pseudocost diving heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

DivingRandom: Whether to try Diving heuristics ↩

This switches on a random diving heuristic at various times. One may prefer to individually turn diving heuristics on or off. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

Synonym: DivingSome

DivingVectorLength: Whether to try Vectorlength diving heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

dualPivot: Dual pivot choice algorithm ↩

The Dantzig method is simple but its use is deprecated. Steepest is the method of choice and there are two variants which keep all weights updated but only scan a subset each iteration. Partial switches this on while automatic decides at each iteration based on information about the factorization. The PE variants add the Positive Edge criterion. This selects incoming variables to try to avoid degenerate moves. See also option psi.

value	meaning
automatic	
dantzig	
partial	
steepest	
PEsteepest	
PEdantzig	
auto	Same as automatic. This is a deprecated setting.

Default: automatic

dumpsolutions: name of solutions index gdx file for writing alternate solutions ↩

The name of a solutions index gdx file for writing alternate solutions found by CBC. The GDX file specified by this option will contain a set called index that contains the names of GDX files with the individual solutions.

Range: string

Default: *empty*

dumpsolutionsmerged: name of gdx file for writing all alternate solutions ↩

Range: string

Default: *empty*

dwHeuristic: Whether to try Dantzig Wolfe heuristic ↩

This heuristic is very very compute intensive. It tries to find a Dantzig Wolfe structure and use that. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before

Default: off

expensiveStrong: Whether to do even more strong branching ↩

Strategy for extra strong branching. 0 is normal strong branching. 1, 2, 4, and 6 does strong branching on all fractional variables if at the root node (1), at depth less than modifier (2), objective equals best possible (4), or at depth less than modifier and objective equals best possible (6). 11, 12, 14, and 16 are like 1, 2, 4, and 6, respecitively, but do strong branching on all integer (incl. non-fractional) variables. Values ≥ 100 are used to specify a depth limit (value/100), otherwise 5 is used. If the values ≥ 100, then above rules are applied to value%100.

Range: {0, ..., ∞}

Default: 0

extraVariables: Allow creation of extra integer variables ↩

Switches on a trivial re-formulation that introduces extra integer variables to group together variables with same cost.

Range: {-∞, ..., ∞}

Default: 0

factorization: Which factorization to use ↩

The default is to use the normal CoinFactorization, but other choices are a dense one, OSL's, or one designed for small problems.

Range: normal, dense, simple, osl

Default: normal

feaspump: Whether to try the Feasibility Pump heuristic ↩

This heuristic is due to Fischetti, Glover, and Lodi and uses a sequence of LPs to try and get an integer feasible solution. Some fine tuning is available by options passFeasibilityPump and pumpTune. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: on

Synonym: feasibilityPump

feaspump_artcost: Costs ≥ this treated as artificials in feasibility pump ↩

A value of 0.0 means off. Otherwise, variables with costs ≥ this are treated as artificial variables and fixed to lower bound in feasibility pump.

Range: [0, ∞]

Default: 0

Synonym: artificialCost

feaspump_cutoff: Fake cutoff for use in feasibility pump ↩

A value of 0.0 means off. Otherwise, add a constraint forcing objective below this value in feasibility pump

Range: real

Default: 0

Synonym: pumpCutoff

feaspump_fracbab: Fraction in feasibility pump ↩

After a pass in the feasibility pump, variables which have not moved about are fixed and if the preprocessed model is smaller than this fraction of the original problem, a few nodes of branch and bound are done on the reduced problem.

Range: [1e-05, 1.1]

Default: 0.5

Synonym: fractionforBAB

feaspump_increment: Fake increment for use in feasibility pump ↩

A value of 0.0 means off. Otherwise use as absolute increment to cutoff when solution found in feasibility pump

Range: real

Default: 0

Synonym: pumpIncrement

feaspump_passes: How many passes to do in the Feasibility Pump heuristic ↩

Range: {0, ..., 10000}

Default: 20

Synonym: passFeasibilityPump

fixOnDj: Try heuristic based on fixing variables with reduced costs greater than this ↩

If this is set integer variables with reduced costs greater than this will be fixed before branch and bound - use with extreme caution!

Range: real

Default: -1

flowCoverCuts: Whether to use Flow Cover cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglFlowCover

Range: off, on, root, ifmove, forceOn, onglobal

Default: ifmove

gamma: Whether to regularize barrier ↩

Range: off, on, gamma, delta, onstrong, gammastrong, deltastrong

Default: off

gomoryCuts: Whether to use Gomory cuts ↩

The original cuts - beware of imitations! Having gone out of favor, they are now more fashionable as LP solvers are more robust and they interact well with other cuts. They will almost always give cuts (although in this executable they are limited as to number of variables in cut). However the cuts may be dense so it is worth experimenting (Long allows any length). Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglGomory

Range: off, on, root, ifmove, forceOn, onglobal, forceandglobal, forceLongOn, long

Default: ifmove

gomorycuts2: Whether to use alternative Gomory cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. This version is by Giacomo Nannicini and may be more robust than gomoryCuts.

Range: off, on, root, ifmove, forceOn, endonly, long, longroot, longifmove, forceLongOn, longendonly

Default: off

Synonym: GMICuts

greedyHeuristic: Whether to use a greedy heuristic ↩

This heuristic tries to obtain a feasible solution by just fixing a percentage of variables and then try a small branch and cut run. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before

Default: on

heuristics: Switches most primal heuristics on or off ↩

This option can be used to switch on or off all heuristics that search for feasible solutions, except for the local tree search, as it dramatically alters the search. Then individual heuristics can be turned off or on.

Range: boolean

Default: 1

Synonym: heuristicsOnOff

hOptions: Heuristic options ↩

Value 1 stops heuristics immediately if the allowable gap has been reached. Other values are for the feasibility pump - 2 says do exact number of passes given, 4 only applies if an initial cutoff has been given and says relax after 50 passes, while 8 will adapt the cutoff rhs after the first solution if it looks as if the code is stalling.

Range: {-∞, ..., ∞}

Default: 0

idiotCrash: Whether to try idiot crash ↩

This is a type of 'crash' which works well on some homogeneous problems. It works best on problems with unit elements and rhs but will do something to any model. It should only be used before the primal simplex algorithm. It can be set to -1 when the code decides for itself whether to use it, 0 to switch off, or n > 0 to do n passes.

Range: {-1, ..., ∞}

Default: -1

increment: A valid solution must be at least this much better than last integer solution ↩

Whenever a solution is found the bound on the objective value for new solutions is set to the objective function of the found solution (in a minimization sense) plus this. If it is not set then CBC will try and work one out, e.g. if all objective coefficients are multiples of 0.01 and only integer variables have entries in the objective function, then the increment can be set to 0.01. Be careful if setting this to a negative value!

Range: real

Default: GAMS cheat

infeasibilityWeight: Each integer infeasibility is expected to cost this much ↩

A primitive way of deciding which node to explore next. Satisfying each integer infeasibility is expected to cost this much.

Range: [0, ∞]

Default: 0

iterlim: Maximum number of iterations before stopping ↩

For an LP, this is the maximum number of iterations to solve the LP. For a MIP, this option is ignored.

Range: {0, ..., ∞}

Default: GAMS iterlim

Synonym: maxIterations

KKT: Whether to use KKT factorization in barrier ↩

Range: boolean

Default: 0

knapsackCuts: Whether to use Knapsack cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglKnapsackCover

Range: off, on, root, ifmove, forceOn, onglobal, forceandglobal

Default: ifmove

lagomoryCuts: Whether to use Lagrangean Gomory cuts ↩

This is a gross simplification of 'A Relax-and-Cut Framework for Gomory's Mixed-Integer Cuts' by Matteo Fischetti & Domenico Salvagnin. This simplification just uses original constraints while modifying objective using other cuts. So you don't use messy constraints generated by Gomory etc. A variant is to allow non messy cuts e.g. clique cuts. So 'only' does this while 'clean' also allows integral valued cuts. 'End' is recommended and waits until other cuts have finished before it does a few passes. The length options for gomory cuts are used.

Range: off, endonlyroot, endcleanroot, root, endonly, endclean, endboth, onlyaswell, cleanaswell, bothaswell, onlyinstead, cleaninstead, bothinstead, onlyaswellroot, cleanaswellroot, bothaswellroot

Default: off

latwomirCuts: Whether to use Lagrangean TwoMir cuts ↩

This is a Lagrangean relaxation for TwoMir cuts. See lagomoryCuts for description of options.

Range: off, endonlyroot, endcleanroot, endbothroot, endonly, endclean, endboth, onlyaswell, cleanaswell, bothaswell, onlyinstead, cleaninstead, bothinstead

Default: off

liftAndProjectCuts: Whether to use Lift and Project cuts ↩

These cuts may be expensive to compute. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglLandP

Range: off, on, root, ifmove, forceOn

Default: off

localTreeSearch: Whether to use local tree search when a solution is found ↩

The heuristic is from Fischetti and Lodi and is not really a heuristic although it can be used as one (with limited functionality). It is not switched on when heuristics are switched on.

Range: boolean

Default: 0

loglevel: amount of output printed by CBC ↩

Range: {0, ..., ∞}

Default: 1

maxFactor: Maximum number of iterations between refactorizations ↩

If this is left at its default value of 200 then CLP will guess a value to use. CLP may decide to re-factorize earlier for accuracy.

Range: {1, ..., ∞}

Default: 200

maxsol: Maximum number of solutions to save ↩

Maximal number of solutions to store during search and to dump into gdx files if dumpsolutions options is set.

Range: {0, ..., ∞}

Default: 100

Synonym: maxSavedSolutions

mipstart: whether it should be tried to use the initial variable levels as initial MIP solution ↩

This option controls the use of advanced starting values for mixed integer programs. A setting of 1 indicates that the variable level values should be checked to see if they provide an integer feasible solution before starting optimization.

Range: boolean

Default: 0

mirCuts: Whether to use Mixed Integer Rounding cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglMixedIntegerRounding2

Range: off, on, root, ifmove, forceOn, onglobal

Default: ifmove

Synonym: mixedIntegerRoundingCuts

multipleRootPasses: Do multiple root passes to collect cuts and solutions ↩

Solve (in parallel, if enabled) the root phase this number of times, each with its own different seed, and collect all solutions and cuts generated. The actual format is aabbcc where aa is the number of extra passes; if bb is non zero, then it is number of threads to use (otherwise uses threads setting); and cc is the number of times to do root phase. The solvers do not interact with each other. However if extra passes are specified then cuts are collected and used in later passes - so there is interaction there. Some parts of this implementation have their origin in idea of Andrea Lodi, Matteo Fischetti, Michele Monaci, Domenico Salvagnin, and Andrea Tramontani.

Range: {0, ..., ∞}

Default: 0

naiveHeuristics: Whether to try some stupid heuristic ↩

This is naive heuristics which, e.g., fix all integers with costs to zero!. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

nodeStrategy: What strategy to use to select the next node from the branch and cut tree ↩

Normally before a feasible solution is found, CBC will choose a node with fewest infeasibilities. Alternatively, one may choose tree-depth as the criterion. This requires the minimal amount of memory, but may take a long time to find the best solution. Additionally, one may specify whether up or down branches must be selected first (the up-down choice will carry on after a first solution has been bound). The default choice 'hybrid' does breadth first on small depth nodes and then switches to 'fewest'.

Range: hybrid, fewest, depth, upfewest, downfewest, updepth, downdepth

Default: fewest

nodlim: node limit ↩

Maximum number of nodes that are enumerated in the Branch and Bound tree search.

Range: {-1, ..., ∞}

Default: GAMS nodlim

Synonyms: maxNodes nodelim

optca: Stop when gap between best possible and best less than this ↩

If the gap between best known solution and the best possible solution is less than this value, then the search will be terminated. Also see ratioGap.

Range: [0, ∞]

Default: GAMS optca

Synonym: allowableGap

optcr: Stop when gap between best possible and best known is less than this fraction of larger of two ↩

If the gap between the best known solution and the best possible solution is less than this fraction of the objective value at the root node then the search will terminate. See 'allowableGap' for a way of using absolute value rather than fraction.

Range: [0, ∞]

Default: GAMS optcr

Synonym: ratioGap

OrbitalBranching: Whether to try orbital branching ↩

This switches on Orbital branching. Value 'on' just adds orbital, 'strong' tries extra fixing in strong branching. 'cuts' just adds global cuts to break symmetry. 'fastish' only computes symmetry at root. 'lightweight' is as on where computation seems cheap

Range: off, on, strong, force, simple, fastish, lightweight, moreprinting, cuts, cutslight

Default: off

parallelmode: whether to run opportunistic or deterministic ↩

Determines whether a parallel MIP search (threads > 1) should be done in a deterministic (i.e., reproducible) way or in a possibly faster but not necessarily reproducible way

Range: opportunistic, deterministic

Default: deterministic

passPresolve: How many passes in presolve ↩

Normally Presolve does 10 passes but you may want to do less to make it more lightweight or do more if improvements are still being made. As Presolve will return if nothing is being taken out, you should not normally need to use this fine tuning.

Range: {-200, ..., 100}

Default: 5

perturbation: Whether to perturb the problem ↩

Perturbation helps to stop cycling, but CLP uses other measures for this. However, large problems and especially ones with unit elements and unit right hand sides or costs benefit from perturbation. Normally CLP tries to be intelligent, but one can switch this off.

Range: boolean

Default: 1

pivotAndComplement: Whether to try Pivot and Complement heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before

Default: off

pivotAndFix: Whether to try Pivot and Fix heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

preprocess: Whether to use integer preprocessing ↩

This tries to reduce size of model in a similar way to presolve and it also tries to strengthen the model - this can be very useful and is worth trying. Value 'save' saves the presolved problem to a file presolved.mps. Value 'equal' will turn inequality-cliques into equalities. Value 'sos' lets CBC search for rows with upper bound 1 and where all nonzero coefficients are 1 and creates special ordered sets if the sets are not overlapping and all integer variables (except for at most one) are in the sets. Value 'trysos' is same as 'sos', but allows any number of integer variables outside of sets. Value 'equalall' lets CBC turn all valid inequalities into equalities by adding integer slack variables.

Range: off, on, save, equal, sos, trysos, equalall, strategy, aggregate, forcesos, stopaftersaving

Default: sos

presolve: Whether to presolve problem ↩

Presolve analyzes the model to find such things as redundant equations, equations which fix some variables, equations which can be transformed into bounds, etc. For the initial solve of any problem this is worth doing unless one knows that it will have no effect. Option 'on' will normally do 5 passes, while using 'more' will do 10.

value	meaning
on	
off	
more	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: on

primalPivot: Primal pivot choice algorithm ↩

The Dantzig method is simple but its use is deprecated. Exact devex is the method of choice and there are two variants which keep all weights updated but only scan a subset each iteration. Partial switches this on while 'change' initially does 'dantzig' until the factorization becomes denser. This is still a work in progress. The PE variants add the Positive Edge criterion. This selects incoming variables to try to avoid degenerate moves. See also Towhidi, M., Desrosiers, J., Soumis, F., The positive edge criterion within COIN-OR's CLP; Omer, J., Towhidi, M., Soumis, F., The positive edge pricing rule for the dual simplex.

value	meaning
automatic	
exact	
dantzig	
partial	
steepest	
change	
sprint	
PEsteepest	
PEdantzig	
auto	Same as automatic. This is a deprecated setting.

Default: automatic

primalWeight: Initially algorithm acts as if it costs this much to be infeasible ↩

The primal algorithm in Clp is a single phase algorithm as opposed to a two phase algorithm where you first get feasible then optimal. So Clp is minimizing this weight times the sum of primal infeasibilities plus the true objective function (in minimization sense). Too high a value may mean more iterations, while too low a value means the algorithm may iterate into the wrong directory for long and then has to increase the weight in order to get feasible.

Range: [1e-20, ∞]

Default: 1e+10

printfrequency: frequency of status prints ↩

Controls the number of nodes that are evaluated between status prints.

Range: {0, ..., ∞}

Default: 0

probingCuts: Whether to use Probing cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Value 'forceOnBut' turns on probing and forces CBC to do probing at every node, but does only probing, not strengthening etc. Value 'strong' forces CBC to strongly do probing at every node, that is, also when CBC would usually turn it off because it hasn't found something. Value 'forceonbutstrong' is like 'forceonstrong', but does only probing (column fixing) and turns off row strengthening, so the matrix will not change inside the branch and bound. Reference: https://github.com/coin-or/Cgl/wiki/CglProbing

Range: off, on, root, ifmove, forceOn, onglobal, forceonglobal, forceOnBut, forceOnStrong, forceOnButStrong, strongRoot

Default: ifmove

proximitySearch: Whether to do proximity search heuristic ↩

This heuristic looks for a solution close to the incumbent solution (Fischetti and Monaci, 2012). The idea is to define a sub-MIP without additional constraints but with a modified objective function intended to attract the search in the proximity of the incumbent. The approach works well for 0-1 MIPs whose solution landscape is not too irregular (meaning the there is reasonable probability of finding an improved solution by flipping a small number of binary variables), in particular when it is applied to the first heuristic solutions found at the root node. Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
10	
100	
300	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

psi: Two-dimension pricing factor for Positive Edge criterion ↩

The Positive Edge criterion has been added to select incoming variables to try and avoid degenerate moves. Variables not in the promising set have their infeasibility weight multiplied by psi, so 0.01 would mean that if there were any promising variables, then they would always be chosen, while 1.0 effectively switches the algorithm off. There are two ways of switching this feature on. One way is to set psi to a positive value and then the Positive Edge criterion will be used for both primal and dual simplex. The other way is to select PEsteepest in dualpivot choice (for example), then the absolute value of psi is used. Code donated by Jeremy Omer. See Towhidi, M., Desrosiers, J., Soumis, F., The positive edge criterion within COIN-OR's CLP; Omer, J., Towhidi, M., Soumis, F., The positive edge pricing rule for the dual simplex.

Range: [-1.1, 1.1]

Default: -0.5

randomizedRounding: Whether to try randomized rounding heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

randomSeedCbc: Random seed for Cbc ↩

Allows initialization of the random seed for pseudo-random numbers used in heuristics such as the Feasibility Pump to decide whether to round up or down. The special value of 0 lets Cbc use the time of the day for the initial seed.

Range: {-1, ..., ∞}

Default: -1

Synonym: randomCbcSeed

randomSeedClp: Random seed for Clp ↩

Initialization of the random seed for pseudo-random numbers used to break ties in degenerate problems. This may yield a different continuous optimum and, in the context of Cbc, different cuts and heuristic solutions. The special value of 0 lets CLP use the time of the day for the initial seed.

Range: {0, ..., ∞}

Default: 1234567

Synonym: randomSeed

reduceAndSplitCuts: Whether to use Reduce-and-Split cuts ↩

These cuts may be expensive to generate. Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglRedSplit

Range: off, on, root, ifmove, forceOn

Default: off

reduceAndSplitCuts2: Whether to use Reduce-and-Split cuts - style 2 ↩

This switches on reduce and split cuts (either at root or in entire tree). This version is by Giacomo Nannicini based on Francois Margot's version. Standard setting only uses rows in tableau ≤ 256, long uses all. These cuts may be expensive to generate. See option cuts for more information on the possible values.

Range: off, on, root, longOn, longRoot

Default: off

Synonym: reduce2AndSplitCuts

Rens: Whether to try Relaxation Enforced Neighborhood Search ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve. Value 'on' just does 50 nodes. 200, 1000, and 10000 does that many nodes.

value	meaning
off	
on	
both	
before	
200	
1000	
10000	
dj	
djbefore	
usesolution	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

residualCapacityCuts: Whether to use Residual Capacity cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglResidualCapacity

Range: off, on, root, ifmove, forceOn

Default: off

reslim: maximum seconds ↩

Range: [-1, ∞]

Default: GAMS reslim

Synonym: seconds

Rins: Whether to try Relaxed Induced Neighborhood Search ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
often	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: off

roundingHeuristic: Whether to use simple (but effective) Rounding heuristic ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

value	meaning
off	
on	
both	
before	
0	Same as off. This is a deprecated setting.
1	Same as on. This is a deprecated setting.

Default: on

scaling: Whether to scale problem ↩

Scaling can help in solving problems which might otherwise fail because of lack of accuracy. It can also reduce the number of iterations. It is not applied if the range of elements is small. When the solution is evaluated in the unscaled problem, it is possible that small primal and/or dual infeasibilities occur. Option 'equilibrium' uses the largest element for scaling. Option 'geometric' uses the squareroot of the product of largest and smallest element. Option 'auto' let CLP choose a method that gives the best ratio of the largest element to the smallest one.

value	meaning
off	
equilibrium	
geometric	
automatic	
dynamic	
rowsonly	
auto	Same as automatic. This is a deprecated setting.

Default: automatic

smallFactorization: Threshold for using small factorization ↩

If processed problem $\leq$ this use small factorization

Range: {-1, ..., 10000}

Default: -1

sollim: Maximum number of feasible solutions to get ↩

Range: {1, ..., ∞}

Default: ∞

Synonym: maxSolutions

solvefinal: final solve of MIP with fixed discrete variables ↩

Whether the MIP with discrete variables fixed to solution values should be solved after CBC finished.

Range: boolean

Default: 1

solvetrace: name of trace file for solving information ↩

Name of file for writing solving progress information during solve.

Range: string

Default: *empty*

solvetracenodefreq: frequency in number of nodes for writing to solve trace file ↩

Range: {0, ..., ∞}

Default: 100

solvetracetimefreq: frequency in seconds for writing to solve trace file ↩

Range: [0, ∞]

Default: 5

sosPrioritize: How to deal with SOS priorities ↩

This sets priorities for SOS. Values 'high' and 'low' just set a priority relative to the for integer variables. Value 'orderhigh' gives first highest priority to the first SOS and integer variables a low priority. Value 'orderlow' gives integer variables a high priority then SOS in order.

Range: off, high, low, orderhigh, orderlow

Default: off

sparseFactor: Whether factorization treated as sparse ↩

Range: boolean

Default: 1

special: options passed unseen to CBC ↩

This parameter let you specify CBC options which are not supported by the GAMS/CBC interface. The string value given to this parameter is split up into parts at each space and added to the array of parameters given to CBC (in front of the -solve command). Hence, you can use it like the command line parameters for the CBC standalone version.

Range: string

Default: *empty*

sprintCrash: Whether to try sprint crash ↩

For long and thin problems this method may solve a series of small problems created by taking a subset of the columns. The idea as 'Sprint' was introduced by J. Forrest after an LP code of that name of the 60's which tried the same tactic (not totally successfully). CPLEX calls it 'sifting'. -1 lets CLP automatically choose the number of passes, 0 is off, n is number of passes

Range: {-1, ..., ∞}

Default: -1

Synonym: sifting

startalg: LP solver for root node ↩

Determines the algorithm to use for an LP or the initial LP relaxation if the problem is a MIP.

value	meaning
primal	Primal Simplex algorithm
dual	Dual Simplex algorithm
barrier	Primal-dual predictor-corrector algorithm

Default: dual

strategy: Switches on groups of features ↩

This turns on newer features. Use 0 for easy problems, 1 is default, 2 is aggressive. 1 uses Gomory cuts with a tolerance of 0.01 at the root node, does a possible restart after 100 nodes if many variables could be fixed, activates a diving and RINS heuristic, and makes the feasibility pump more aggressive.

Range: {0, ..., 2}

Default: 1

strongBranching: Number of variables to look at in strong branching ↩

In order to decide which variable to branch on, the code will choose up to this number of unsatisfied variables to try minimal up and down branches on. Then the most effective one is chosen. If a variable is branched on many times then the previous average up and down costs may be used - see also option trustPseudoCosts.

Range: {0, ..., ∞}

Default: 5

substitution: How long a column to substitute for in presolve ↩

Normally Presolve gets rid of 'free' variables when there are no more than 3 coefficients in a row. If you increase this, the number of rows may decrease but the number of coefficients may increase.

Range: {0, ..., 10000}

Default: 3

threads: Number of threads to try and use ↩

If set to 0, then multithreading of Cbc itself is disabled, but all available processors are made available for linear algebra subroutines.

Range: {0, ..., 99}

Default: GAMS threads

tol_dual: For an optimal solution no dual infeasibility may exceed this value ↩

Normally the default tolerance is fine, but one may want to increase it a bit if the dual simplex algorithm seems to be having a hard time. One method which can be faster is to use a large tolerance e.g. 1.0e-4 and the dual simplex algorithm and then to clean up the problem using the primal simplex algorithm with the correct tolerance (remembering to switch off presolve for this final short clean up phase).

Range: [1e-20, ∞]

Default: 1e-07

Synonym: dualTolerance

tol_integer: For a feasible solution no integer variable may be more than this away from an integer value ↩

Beware of setting this smaller than the primal feasibility tolerance.

Range: [1e-20, 0.5]

Default: 1e-07

Synonym: integerTolerance

tol_presolve: Tolerance to use in presolve ↩

One may want to increase this tolerance if presolve says the problem is infeasible and one has awkward numbers and is sure that the problem is really feasible.

Range: [1e-20, ∞]

Default: 1e-08

Synonym: preTolerance

tol_primal: For a feasible solution no primal infeasibility, i.e., constraint violation, may exceed this value ↩

Normally the default tolerance is fine, but one may want to increase it a bit if the primal simplex algorithm seems to be having a hard time.

Range: [1e-20, ∞]

Default: 1e-07

Synonym: primalTolerance

trustPseudoCosts: Number of branches before we trust pseudocosts ↩

Using strong branching computes pseudo-costs. This parameter determines after how many branches for a variable we just trust the pseudo costs and do not do any more strong branching.

Range: {-3, ..., ∞}

Default: 10

twoMirCuts: Whether to use Two phase Mixed Integer Rounding cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. Reference: https://github.com/coin-or/Cgl/wiki/CglTwomir

Range: off, on, root, ifmove, forceOn, onglobal, forceandglobal, forceLongOn

Default: root

VndVariableNeighborhoodSearch: Whether to try Variable Neighborhood Search ↩

Value 'on' means to use the heuristic in each node of the tree, i.e. after preprocessing. Value 'before' means use the heuristic only if option doHeuristics is used. Value 'both' means to use the heuristic if option doHeuristics is used and during solve.

Range: off, on, both, before, intree

Default: off

vubheuristic: Type of VUB heuristic ↩

This heuristic tries and fix some integer variables.

Range: {-2, ..., 20}

Default: -1

writemps: create MPS file for problem ↩

Write the problem formulation in MPS format. The parameter value is the name of the MPS file.

Range: string

Default: *empty*

zeroHalfCuts: Whether to use zero half cuts ↩

Value 'on' enables the cut generator and CBC will try it in the branch and cut tree (see cutDepth on how to fine tune the behavior). Value 'root' lets CBC run the cut generator generate only at the root node. Value 'ifmove' lets CBC use the cut generator in the tree if it looks as if it is doing some good and moves the objective value. Value 'forceon' turns on the cut generator and forces CBC to use it at every node. This implementation was written by Alberto Caprara.

Range: off, on, root, ifmove, forceOn, onglobal

Default: ifmove

5.7 CONOPT 3

Author

Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

5.7.1 Introduction

This documentation is for CONOPT3. We will refer to CONOPT3 as CONOPT in the following. Note that there is also the improved version CONOPT4.

Nonlinear models created with GAMS must be solved with a nonlinear programming (NLP) algorithm. Currently, there is a large number of different solvers available and the number is growing.

The most important distinction between the solvers is whether they attempt to find a local or a global solution. Solvers that attempt to find a global solution (so called Global Solvers) can usually not solve very large models. As a contrast most Local Solvers can work with much larger models, and models with over 10,000 variables and constraints are not unusual. If the model has the right mathematical properties, e.g. is convex, then Local Solvers will find a global optimum. Unfortunately, the mathematical machinery for testing whether a general NLP model is convex or not has not yet been developed (and is expected to be in the class or hard problems).

It is almost impossible to predict how difficult it is to solve a particular model with a particular algorithm, especially for NLP models, so GAMS cannot select the best algorithm for you automatically. When GAMS is installed you must select one of the nonlinear programming algorithms as the default solver for NLP models. If you want to switch between algorithms for a particular model you may add the statement `Option NLP = <solvername>`, in your GAMS source file before the `Solve` statement, you may add `NLP = <solvername>` on the GAMS command line or by rerunning the `gamsinst` program.

The only reliable way to find which solver to use for a particular class of models is so far to experiment. However, there are a few rules of thumb:

CONOPT is well suited for models with very nonlinear constraints. If you experience that a solver has problems maintaining feasibility during the optimization you should try CONOPT. On the other hand, if you have a model with few nonlinearities outside the objective function then other solvers could be the best solver.

CONOPT has a fast method for finding a first feasible solution that is particularly well suited for models with few degrees of freedom. If you have a model with roughly the same number of constraints as variable you should try CONOPT. CONOPT can also be used to solve square systems of equations without an objective function corresponding to the GAMS model class CNS - Constrained Nonlinear System.

CONOPT can use second derivatives. If the number of variables is much larger than the number of constraints CONOPT will use second derivatives and overall progress can be considerably faster than for MINOS or SNOPT. IPOPT and KNITRO will also use second derivatives, but the method is very different and it is not possible to predict which solver will be better.

CONOPT has a preprocessing step in which recursive equations and variables are solved and removed from the model. If you have a model where many equations can be solved one by one then CONOPT will take advantage of this property. Similarly, intermediate variables only used to define objective terms are eliminated from the model and the constraints are moved into the objective function.

CONOPT has many built-in tests and messages, and many models that can and should be improved by the modeler are rejected with a constructive message. CONOPT is therefore also a helpful debugging tool during model development. The best solver for the final, debugged model may or may not be CONOPT.

CONOPT has been designed for large and sparse models. This means that both the number of variables and equations can be large. Indeed, NLP models with over 100,000 equations and variables have been solved successfully, and CNS models with over 1,000,000 equations and variables have also been solved. The components used to build CONOPT have been selected under the assumptions that the model is sparse, i.e. that most functions only depend on a small number of variables. CONOPT can also be used for denser models, but the performance will suffer significantly.

CONOPT is designed for models with smooth functions, but it can also be applied to models that do not have differentiable functions, in GAMS called DNLP models. However, CONOPT will use the same algorithm used for a real NLP model and it will search for a point that satisfies standard first-order optimality conditions without taking into account that parts of the model could be non-smooth or non-differentiable. The lack of smoothness may confuse the algorithm in CONOPT causing slow convergence, and a point that satisfies standard first-order optimality conditions may not even exist. There are therefore no guaranties whatsoever for this class of models. If CONOPT terminates with a locally optimal solution then the solution will indeed be locally optimal. However, you will sometimes get termination messages like "Convergence too slow" or "No change in objective although the reduced gradient is greater than the tolerance" that indicate unsuccessful termination. The final point may or may not be locally optimal. If possible, you should try to reformulate a DNLP model to an equivalent or approximately equivalent form as described in section NLP and DNLP Models .

Most modelers should not be concerned with algorithmic details such as choice of algorithmic sub-components or tolerances. CONOPT has considerable build-in logic that selects a solution approach that seems to be best suited for the type of model at hand, and the approach is adjusted dynamically as information about the behavior of the model is collected and updated. The description of the CONOPT algorithm has therefore been moved to an appendix (Appendix A) and most modelers can skip it. However, if you are solving very large or complex models or if you are experiencing solution difficulties you may benefit from using non-standard tolerances or options, in which case you will need some understanding of what CONOPT is doing to your model. Some guidelines for selecting options can be found at the end of Appendix A and a list of all options and tolerances is shown in Appendix B.

The main text of this User's Guide will give a short overview over the iteration output you will see on the screen (section Iteration Output), and explain the termination messages (section CONOPT Termination Messages). We will then discuss function evaluation errors (section Function Evaluation Errors), the use of options (section The CONOPT Options File), and give a CONOPT perspective on good model formulation including topics such as initial values and bounds, simplification of expressions, and scaling (section Hints on Good Model Formulation). Finally, we will discuss the difference between NLP and DNLP models (section NLP and DNLP Models).

5.7.2 Iteration Output

On most machines you will by default get a logline on your screen or terminal at regular intervals. The iteration log may look something like this:

```
CONOPT 3         Jul  4, 2012 23.9.4 WEX 35892.35906 WEI x86_64/MS Windows


    C O N O P T 3   version 3.15G
    Copyright (C)   ARKI Consulting and Development A/S
                    Bagsvaerdvej 246 A
                    DK-2880 Bagsvaerd, Denmark


   Iter Phase Ninf   Infeasibility   RGmax    NSB   Step InItr MX OK
      0   0        1.6354151782E+01 (Input point)
                                 Pre-triangular equations:        2
                                 Post-triangular equations:       1
```

```
     1   0        1.5354151782E+01 (After pre-processing)
     2   0        3.0983571843E+00 (After scaling)
    10   0    12  3.0814290456E+00               0.0E+00      T  T
    20   0    12  3.0814290456E+00               0.0E+00      T  T
    30   0    13  3.0814290456E+00               0.0E+00      F  F
    40   0    18  2.3738740159E+00               2.3E-02      T  T
    50   0    23  2.1776589484E+00               0.0E+00      F  F

  Iter Phase Ninf   Infeasibility   RGmax    NSB   Step InItr MX OK
    60   0    33  2.1776589484E+00               0.0E+00      T  T
    70   0    43  2.1776589484E+00               0.0E+00      F  F
    80   0    53  2.1776589484E+00               0.0E+00      F  F
    90   0    63  2.1776589484E+00               0.0E+00      F  F
   100   0    73  2.1776589484E+00               0.0E+00      F  F
   110   0    83  2.1776589484E+00               0.0E+00      F  F
   120   0    93  2.1776589484E+00               0.0E+00      F  F
   130   0   103  2.1776589484E+00               0.0E+00      F  F
   140   0   113  2.1776589484E+00               0.0E+00      T  T
   150   0   119  8.7534351971E-01               0.0E+00      F  F

  Iter Phase Ninf   Infeasibility   RGmax    NSB   Step InItr MX OK
   160   0   124  9.5022881759E-01               0.0E+00      F  F
   170   0   134  9.5022881759E-01               0.0E+00      F  F
   180   0   144  9.5022881759E-01               0.0E+00      F  F
   190   0   154  9.5022881759E-01               0.0E+00      F  F
   201   1   160  9.4182618946E-01 4.3E+01   134 2.4E-06      T  T
   206   1   130  8.2388503304E-01 9.5E+01   138 1.0E+00   13 T  T
   211   1    50  1.0242911941E-01 6.9E+00    84 7.2E-01   24 T  T
   216   1    16  2.6057507770E-02 1.3E+00    52 6.1E-01   17 T  T
   221   1     5  7.2858773666E-04 6.1E-03    38 6.0E-01    7 F  F

 ** Feasible solution. Value of objective =    1.00525015566

  Iter Phase Ninf     Objective     RGmax    NSB   Step InItr MX OK
   226   3        1.0092586645E+00 4.4E-04    38 1.0E+00    3 T  T
   231   3        1.0121749760E+00 1.4E+00    24 4.8E-01    9 T  T
   236   3        1.0128148550E+00 4.8E-06    13 5.8E-02   12 F  T
   241   3        1.0128161551E+00 2.5E-06    12 9.1E+03      F  T
   246   4        1.0128171043E+00 1.2E-07    13 1.0E+00    3 F  T
   247   4        1.0128171043E+00 5.7E-08    13

 ** Optimal solution. Reduced gradient less than tolerance.
```

The first few lines identify the version of CONOPT you use.

The first iterations have a special interpretation: iteration 0 represents the initial point exactly as received from GAMS, iteration 1 represent the point that is the result of CONOPT's pre-processing, and iteration 2 represents the same point after scaling (even if scaling is turned off).

The remaining iterations are characterized by the value of "Phase" in column 2. The model is infeasible during Phase 0, 1, and 2 and the Sum of Infeasibilities in column 4 (labeled "Infeasibility") is being minimized; the model is feasible during Phase 3 and 4 and the actual objective function, also shown in column 4 (now labeled "Objective"), is minimized or maximized. Phase 0 iterations are Newton-like iterations. They are very cheap so you should not be concerned if there are many of these Phase 0 iterations. During Phase 1 and 3 the model behaves almost linearly and CONOPT applies special linear iterations that take advantage of the linearity. These iterations are sometimes augmented with some inner "Sequential Linear Programming" (SLP) iterations, indicated by a number of inner SLP iterations in the "InItr" column. During Phase 2 and 4 the model behaves more nonlinear and most aspects of the

iterations are therefore changed: the line search is more elaborate, and CONOPT needs second order information to improve the convergence. For small and simple models CONOPT will approximate second order information as a byproduct of the line searches. For larger and more complex models CONOPT will use some inner "Sequential Quadratic Programming" (SQP) iterations based on exact second derivatives. These SQP iterations are identified by the number of inner SQP iterations in the "InItr" column.

The column "NSB" for Number of SuperBasics defines the degree of freedom or the dimension of the current search space, and "rgmax" measures the largest reduced gradient among the non-optimal variables. Rgmax should eventually converge towards zero, but convergence is not expected to be monotone. The last two columns labeled "MX" and "OK" gives information about the line search: OK = T means that the line search was well-behaved, and OK = F means that the line search was terminated before an optimal step length was found because it was not possible to find a feasible solution for large step lengths. MX = T means that the line search was terminated by a variable reaching a bound (always combined with OK = T), and MX = F means that the step length was determined by nonlinearities. If OK = T then the step length was optimal; if OK = F then the constraints were too nonlinear to allow CONOPT to make a full optimal step.

5.7.3 CONOPT Termination Messages

CONOPT may terminate in a number of ways. This section will show most of the termination messages and explain their meaning. It will also show the Model Status returned to GAMS in <model>.modelStat, where <model> represents the name of the GAMS model. The Solver Status returned in <model>.solveStat will be given if it is different from 1 (Normal Completion). We will in all cases first show the message from CONOPT followed by a short explanation. The first 4 messages are used for optimal solutions and CONOPT will return modelStat = 2 (Locally Optimal), except as noted below:

```
** Optimal solution. There are no superbasic variables.
```

The solution is a locally optimal corner solution. The solution is determined by constraints only, and it is usually very accurate. In some cases CONOPT can determine that the solution is globally optimal and it will return modelStat = 1 (Optimal).

```
** Optimal solution. Reduced gradient less than tolerance.
```

The solution is a locally optimal interior solution. The largest component of the reduced gradient is less than the tolerance `rtredg` with default value 1.e-7. The value of the objective function is very accurate while the values of the variables are less accurate due to a flat objective function in the interior of the feasible area.

```
** Optimal solution. The error on the optimal objective function
   value estimated from the reduced gradient and the estimated
   Hessian is less than the minimal tolerance on the objective.
```

The solution is a locally optimal interior solution. The largest component of the reduced gradient is larger than the tolerance `rtredg`. However, when the reduced gradient is scaled with information from the estimated Hessian of the reduced objective function the solution seems optimal. For this to happen the objective must be large or the reduced objective must have large second derivatives so it is advisable to scale the model. See the sections on "Scaling" and "Using the Scale Option in GAMS" for details on how to scale a model.

```
** Optimal solution. Convergence too slow. The change in
   objective has been less than xx.xx for xx consecutive
   iterations.
```

CONOPT stops with a solution that seems optimal. The solution process is stopped because of slow progress. The largest component of the reduced gradient is greater than the optimality tolerance `rtredg`, but less than `rtredg` multiplied by the largest Jacobian element divided by 100. The model must have large derivatives so it is advisable to scale it.

The four messages above all exist in versions where "Optimal" is replaced by "Infeasible" and modelStat will be 5 (Locally Infeasible) or 4 (Infeasible). The infeasible messages indicate that a Sum of Infeasibility objective function is locally minimal, but positive. If the model is convex it does not have a feasible solution; if the model is non-convex it may have a feasible solution in a different region. See the section on "Initial Values" for hints on what to do.

```
** Feasible solution. Convergence too slow. The change in
   objective has been less than xx.xx for xx consecutive
   iterations.

** Feasible solution. The tolerances are minimal and
   there is no change in objective although the reduced
   gradient is greater than the tolerance.
```

The two messages above tell that CONOPT stops with a feasible solution. In the first case the solution process is very slow and in the second there is no progress at all. However, the optimality criteria have not been satisfied. These messages are accompanied by modelStat = 7 (Feasible Solution) and solveStat = 4 (Terminated by Solver). The problem can be caused by discontinuities if the model is of type DNLP; in this case you should consider alternative, smooth formulations as discussed in section NLP and DNLP Models . The problem can also be caused by a poorly scaled model. See section Scaling for hints on model scaling. Finally, it can be caused by stalling as described in section Stalling in Appendix A. The two messages also exist in a version where "Feasible" is replaced by "Infeasible". modelStat is in this case 6 (Intermediate Infeasible) and solveStat is still 4 (Terminated by Solver); these versions tell that CONOPT cannot make progress towards feasibility, but the Sum of Infeasibility objective function does not have a well defined local minimum.

```
   <var>: The variable has reached infinity

** Unbounded solution. A variable has reached 'infinity'.
   Largest legal value (Rtmaxv) is xx.xx
```

CONOPT considers a solution to be unbounded if a variable exceeds the indicated value of `rtmaxv` (default 1.e10) and it returns with modelStat = 3 (Unbounded). The check for unboundedness is done at every iteration which means that CONOPT will stop if an intermediate solution has a variable that is very large, even if none of the variables in the optimal solution have large values. Check whether the solution appears unbounded or the problem is caused by the scaling of the unbounded variable "<var>" mentioned in the first line of the message. If the model seems correct you are advised to scale it. There is also a lazy solution: you can increase the largest legal value, `rtmaxv`, as mentioned in the section on options. However, you will pay through reduced reliability or increased solution times. Unlike LP models, where an unbounded model is recognized by an unbounded ray and the iterations are stopped far from "infinity", CONOPT will actually have to make a line search and move to a region with large values of the variables. This may lead to bad scaling and to many different kinds of tolerance and roundoff problems, including problems of determining whether a solution is feasible or not.

The message above exists in a version where "Unbounded" is replaced by "Infeasible" and modelStat is 5 (Locally Infeasible). You may also see a message like

```
   <var>: Free variable becomes too large

** Infeasible solution. A free variable exceeds the allowable
   range.  Current value is 1.02E+10 and current upper bound
   (Rtmaxv) is 1.00E+10
```

These two messages indicate that some variables become very large before a feasible solution has been found. You should again check whether the problem is caused by the scaling of the unbounded variable "<var>" mentioned in the first line of the message. If the model seems correct you should scale it.

```
** The time limit has been reached.
```

The time or resource limit defined in GAMS, either by default (usually 1000 seconds) or by `Option ResLim = xx;` or `<model>.ResLim = xx;` statements, has been reached. CONOPT will return with solveStat = 3 (Resource Interrupt) and modelStat either 6 (Locally Infeasible) or 7 (Feasible Solution).

```
** The iteration limit has been reached.
```

The iteration limit defined in GAMS, either by default (usually 2000000000 iterations) or by `Option IterLim = xx;` or `<model>.IterLim = xx;` statements, has been reached. CONOPT will return with solveStat = 2 (Iteration Interrupt) and modelStat either 6 (Locally Infeasible) or 7 (Feasible Solution).

```
** Domain errors in nonlinear functions.
   Check bounds on variables.
```

The number of function evaluation errors has reached the limit defined in GAMS by `Option DomLim = xx;` or `<model>.DomLim = xx;` statements or the default limit of 0 function evaluation errors. CONOPT will return with solveStat = 5 (Evaluation Error Limit) and modelStat either 6 (Locally Infeasible) or 7 (Feasible Solution). See more details in section Function Evaluation Errors .

```
** An initial derivative is too large (larger than Rtmaxj= xx.xx)
   Scale the variables and/or equations or add bounds.

   <var> appearing in
   <equ>: Initial Jacobian element too large = xx.xx
```

and

```
** A derivative is too large (larger than Rtmaxj= xx.xx).
   Scale the variables and/or equations or add bounds.

   <var> appearing in
   <equ>: Jacobian element too large = xx.xx
```

These two messages appear if a derivative or Jacobian element is very large, either in the initial point or in a later intermediate point. The relevant variable and equation pair(s) will show you where to look. A large derivative means that the function changes very rapidly even after a very small change in the variable and it will most likely create numerical problems for many parts of the optimization algorithm. Instead of attempting to solve a model that most likely will fail, CONOPT will stop and you are advised to adjust the model if at all possible.

If the offending derivative is associated with a Log(x) or 1/x term you may try to increase the lower bound on x. If the offending derivative is associated with an Exp(x) term you must decrease the upper bound on x. You may also try to scale the model, either manually or using the variable.Scale and/or equation.Scale option in GAMS as described in section Scaling There is also in this case a lazy solution: increase the limit on Jacobian elements, `rtmaxj`; however, you will pay through reduced reliability or longer solution times.

In addition to the messages shown above you may see messages like

```
** An equation in the pre-triangular part of the model cannot be
   solved because the critical variable is at a bound.

** An equation in the pre-triangular part of the model cannot be
   solved because of too small pivot.
```

or

```
** An equation is inconsistent with other equations in the
   pre-triangular part of the model.
```

These messages containing the word "Pre-triangular" are all related to infeasibilities identified by CONOPT's pre-processing stage and they are explained in detail in section Iteration 1: Preprocessing in Appendix A.

Usually, CONOPT will be able to estimate the amount of memory needed for the model based on statistics provided by GAMS. However, in some cases with unusual models, e.g. very dense models or very large models, the estimate will be too small and you must request more memory yourself using a statement like <model>.WorkFactor = x.x; or <model>.WorkSpace = xx; in GAMS or by adding WorkFactor=xx to the command line call of GAMS. The message you will see is similar to the following:

```
** FATAL ERROR **  Insufficient memory to continue the
                   optimization.

                   You must request more memory.
                   Current   CONOPT space =  0.29 Mbytes
                   Estimated CONOPT space =  0.64 Mbytes
                   Minimum   CONOPT space =  0.33 Mbytes


CONOPT time Total                          0.109 seconds
  of which: Function evaluations           0.000 =  0.0%
            1st derivative evaluations     0.000 =  0.0%
```

The text after "Insufficient memory to" may be different; it says something about where CONOPT ran out of memory. If the memory problem appears during model setup the message will be accompanied by solveStat = 13 (System Failure) and modelStat = 13 (Error No Solution) and CONOPT will not return any values. If the memory problem appears later during the optimization solveStat will be 11 (Internal Solver Failure) and modelStat will be either 6 (Intermediate Infeasible) or 7 (Feasible Solution) and CONOPT will return primal solution values. The marginals of both equations and variables will be zero or EPS.

It is recommended that you use the WorkFactor option if you must change the amount of memory. The same number will usually work for a whole family of models. If you prefer to use WorkSpace, the GAMS WorkSpace option corresponds to the amount of memory, measured in Mbytes.

A new termination message has been added from version 3.16C:

```
** Feasible solution. The solution process has been terminated
   because intermediate results have become NaN (Not A Number).
```

and similar with Infeasible. To prevent non-sensible results and/or infinite loops in special degenerate cases CONOPT has added checks for internal intermediate results being NaN (Not A Number) or very large. If this happens CONOPT will try to change some tolerances and try to continue the optimization. If this attempt fails CONOPT will stop and return the message above. The solver status will return 4 "Terminated by Solver" and model status 6 or 7, "Intermediate Infeasible" or "Intermediate Feasible." Section Overflow and NaN (Not A Number) in Appendix A has more details on the sources of NaN and the actions that can be taken by the user and by CONOPT.

5.7.4 Function Evaluation Errors

Many of the nonlinear functions available with GAMS are not defined for all values of their arguments. Log is not defined for negative arguments, Exp overflows for large arguments, and division by zero is illegal. To avoid evaluating functions outside their domain of definition you should add reasonable variable bounds. CONOPT will in return guarantee that the nonlinear functions never are evaluated with variables outside their bounds.

In some cases bounds are not sufficient, e.g. in the expression Log(Sum(i, x(i))): in some models each individual x should be allowed to become zero, but the Sum should not. In this case you should introduce an intermediate variable and an extra equation, e.g. xSumDef .. xSum =E= sum(i,x(i)); add a lower bound on xSum; and use xSum as the argument to the Log function. See section Simple Expressions for additional comments on this topic.

Whenever a nonlinear function is called outside its domain of definition, GAMS' function evaluator will intercept the function evaluation error and prevent the system to crash. GAMS will replace the undefined result by some appropriate real number, and it will make sure the error is reported to the modeler as part of the standard solution output in the GAMS listing file. GAMS will also report the error to CONOPT, so CONOPT can try to correct the problem by backtracking to a safe point. Finally, CONOPT will be instructed to stop after `DomLim` errors.

During Phase 0, 1, and 3 CONOPT will often use large steps as the initial step in a line search and functions will very likely be called with some of the variables at their lower or upper bound. You are therefore likely to get a division-by-zero error if your model contains a division by x and x has a lower bound of zero. And you are likely to get an exponentiation overflow error if your model contains Exp(x) and x has no upper bound. However, CONOPT will usually not get trapped in a point outside the domain of definition for the model. When GAMS' function evaluator reports that a point is "bad", CONOPT will decrease the step length, and it will for most models be able to recover and continue to an optimal solution. It is therefore safe to use a large value for `DomLim` instead of GAMS default value of 0.

CONOPT may get stuck in some cases, for example because there is no previous point to backtrack to, because "bad" points are very close to "reasonable" feasible points, or because the derivatives are not defined in a feasible point. The more common messages are:

```
** Fatal Error **  Function error in initial point in Phase 0
                   procedure.

** Fatal Error **  Function error after small step in Phase 0
                   procedure.

** Fatal Error **  Function error very close to a feasible point.

** Fatal Error **  Function error while reducing tolerances.

** Fatal Error **  Function error in Pre-triangular equations.

** Fatal Error **  Function error after Preprocessing.

** Fatal Error **  Function error in Post-triangular equation.
```

In the first four cases you must either add better bounds or define better initial values. If the problem is related to a pre- or post-triangular equation as shown by the last three messages then you can turn part of the pre-processing off as described in section Iteration 1: Preprocessing . in Appendix A. However, this may make the model harder to solve, so it is usually better to add bounds and/or initial values.

5.7.5 The CONOPT Options File

CONOPT has been designed to be self-tuning. Most tolerances are dynamic. As an example: The feasibility of a constraint is always judged relative to the dual variable on the constraint and relative to the expected change in objective in the coming iteration. If the dual variable is large then the constraint must be satisfied with a small tolerance, and if the dual variable is small then the tolerance is larger. When the expected change in objective in the first iterations is large then the feasibility tolerances are also large. And when we approach the optimum and the expected change in objective becomes smaller then the feasibility tolerances become smaller.

Because of the self-tuning nature of CONOPT you should in most cases be well off with default tolerances. If you do need to change some tolerances, possibly following the advice in Appendix A, it can be done in the CONOPT Options file. The name of the CONOPT Options file is on most systems "conopt.opt". You must tell the solver that you want to use an options file with the statement `<model>.OptFile = 1` in your GAMS source file before the Solve statement or with `OptFile = 1` on the command line.

The format of the CONOPT Options file consists in its simplest form of a number of lines like these:

```
rtmaxv = 1.e12
lfnsup = 500
```

The value must be written using legal GAMS format, i.e. a real number may contain an optional E exponent, but a number may not contain blanks. The value must have the same type as the option, i.e. real options must be assigned real values, integer options must be assigned integer values, and logical options must be assigned logical values. The logical value representing true are `true, t, yes`, or `1`, and the logical values representing false are `false, f, no`, or `0`.

In previous versions of CONOPT you could add "SET" in front of the option assignment. This is no longer supported. You can still replace the equal sign with `:=` and you can add end of line comments after a `#` or `!` character. Lines starting with $*$ in column 1 are treated as comment lines.

5.7.6 Hints on Good Model Formulation

This section will contain some comments on how to formulate a nonlinear model so it becomes easier to solve with CONOPT. Most of the recommendations will be useful for any nonlinear solver, but not all. We will try to mention when a recommendation is CONOPT specific.

5.7.6.1 Initial Values

Good initial values are important for many reasons. Initial values that satisfy or closely satisfy many of the constraints reduces the work involved in finding a first feasible solution. Initial values that in addition are close to the optimal ones also reduce the distance to the final point and therefore indirectly the computational effort. The progress of the optimization algorithm is based on good directional information and therefore on good derivatives. The derivatives in a nonlinear model depend on the current point, and the initial point in which the initial derivatives are computed is therefore again important. Finally, non-convex models may have multiple solutions, but the modeler is looking for one in a particular part of the search space; an initial point in the right neighborhood is more likely to return the desired solution.

The initial values used by CONOPT are all coming from GAMS. The initial values used by GAMS are by default the value zero projected on the bounds. I.e. if a variable is free or has a lower bound of zero, then its default initial value is zero. Unfortunately, zero is in many cases a bad initial value for a nonlinear variable. An initial value of zero is especially bad if the variable appears in a product term since the initial derivative becomes zero, and it appears as if the function does not depend on the variable. CONOPT will

warn you and ask you to supply better initial values if the number of derivatives equal to zero is larger than 20 percent.

If a variable has a small positive lower bound, for example because it appears as an argument to the Log function or as a denominator, then the default initial value is this small lower bound and it is also bad since this point will have very large first and second derivatives.

You should therefore supply as many sensible initial values as possible by making assignment to the level value, var.L, in GAMS. An easy possibility is to initialize all variables to 1, or to the scale factor if you use GAMS' scaling option. A better possibility is to select reasonable values for some variables that from the context are known to be important, and then use some of the equations of the model to derive values for other variables. A model may contain the following equation:

```
pmDef(it) .. pm(it) =e= pwm(it)*er*(1 + tm(it)) ;
```

where pm, pwm, and er are variables and tm is a parameter. The following assignment statements use the equation to derive consistent initial values for PM from sensible initial values for pwm and er:

```
er.l = 1; pwm.l(it) = 1;
pm.l(it) = pwm.l(it)*er.l*(1 + tm(it)) ;
```

With these assignments equation pmDef will be feasible in the initial point, and since CONOPT uses a feasible path method it will remain feasible throughout the optimization (unless the pre-processor destroys it, see section Iteration 1: Preprocessing in Appendix A).

If CONOPT has difficulties finding a feasible solution for your model you should try to use this technique to create an initial point in which as many equations as possible are satisfied. You may also try the optional Crash procedure described in section Preprocessing: The Optional Crash Procedure in Appendix A by adding the line `lstcrs=t` to the CONOPT options file. The crash procedure tries to identify equations with a mixture of uninitialized variables and variables with initial values, and it solves the equations with respect to the uninitialized variables; the effect is similar to the manual procedure shown above.

5.7.6.2 Bounds

Bounds have two purposes in nonlinear models. Some bounds represent constraints on the reality that is being modeled, e.g. a variable must be positive. These bounds are called model bounds. Other bounds help the algorithm by preventing it from moving far away from any optimal solution and into regions with singularities in the nonlinear functions or unreasonably large function or derivative values. These bounds are called algorithmic bounds.

Model bounds have natural roots and do not cause any problems. Algorithmic bounds require a closer look at the functional form of the model. The content of a Log should be greater than say 1.e-3, the content of an Exp should be less than 5 to 8, and a denominator should be greater than say 1.e-2. These recommended lower bounds of 1.e-3 and 1.e-2 may appear to be unreasonably large. However, both Log(x) and 1/x are extremely nonlinear for small arguments. The first and second derivatives of Log(x) at x=1.e-3 are 1.e+3 and -1.e6, respectively, and the first and second derivatives of 1/x at x=1.e-2 are -1.e+4 and 2.e+6, respectively.

If the content of a Log or Exp function or a denominator is an expression then it may be advantageous to introduce a bounded intermediate variable as discussed in the next section.

Note that bounds in some cases can slow the solution process down. Too many bounds may for example introduce degeneracy. If you have constraints of the following type

```
vub(i) .. x(i) =l= y;
```

or

```
ysum   .. y =e= sum( i, x(i) );
```

and x is a `Positive Variable` then you should in general not declare y a `Positive Variable` or add a lower bound of zero on y. If y appears in a nonlinear function you may need a strictly positive bound. Otherwise, you should declare y a free variable; CONOPT will then make y basic in the initial point and y will remain basic throughout the optimization. New logic in CONOPT tries to remove this problem by detecting when a harmful bound is redundant so it can be removed, but it is not yet a fool proof procedure.

Section Iteration 1: Preprocessing in Appendix A gives another example of bounds that can be counter productive.

5.7.6.3 Simple Expressions

The following model component

```
Parameter mu(i);
Variable  x(i), s(i), obj;
Equation  objDef;
objDef .. obj =e= Exp( Sum( i, Sqr( x(i) - mu(i) ) / s(i) ) );
```

can be re-written in the slightly longer but simpler form

```
Parameter mu(i);
Pariable  x(i), s(i), obj, inTerm;
Equation  intDef, objDef;
intDef .. inTerm =e= Sum( i, Sqr( x(i) - mu(i) ) / s(i) );
objDef .. obj    =e= Exp( inTerm );
```

The first formulation has very complex derivatives because Exp is taken of a long expression. The second formulation has much simpler derivatives; Exp is taken of a single variable, and the variables in intDef appear in a sum of simple independent terms.

In general, try to avoid nonlinear functions of expressions, divisions by expressions, and products of expressions, especially if the expressions depend on many variables. Define intermediate variables that are equal to the expressions and apply the nonlinear function, division, or product to the intermediate variable. The model will become larger, but the increased size is taken care of by CONOPT's sparse matrix routines, and it is compensated by the reduced complexity. If the model is solved with CONOPT using explicit second derivatives then simple expressions will result in sparser second derivatives that are both faster to compute and to use.

The reduction in complexity can be significant if an intermediate expression is linear. The following model fragment:

```
Variable x(i), y;
Equation yDef;
yDef ..  y =e= 1 / Sum(i, x(i) );
```

should be written as

```
Variable x(i), xSum, y;
Equation xSumDef, yDef;
xSumDef .. xSum =e= Sum(i, x(i) );
yDef    .. y =e= 1 / xSum;
xSum.lo = 1.e-2;
```

for three reasons. First, because the number of nonlinear derivatives is reduced in number and complexity. Second, because the lower bound on the intermediate result will bound the search away from the singularity at xSum = 0. And third, because the matrix of second derivatives for the last model only depend on xSum while it depends on all x in the first model.

The last example shows an added potential saving by expanding functions of linear expressions. A constraint depends in a nonlinear fashion on the accumulated investments, inv, like

```
con(i) .. f( Sum( j$(ord(j) le ord(i)), inv(j) ) ) =l= b(i);
```

A new intermediate variable, cap(i), that is equal to the content of the Sum can be defined recursively with the constraints

```
cDef(i) .. cap(i) =e= inv(i) + cap(i-1);
```

and the original constraints become

```
con(i) .. f( cap(i) ) =l= b(i);
```

The reformulated model has n additional variables and n additional linear constraints. In return, the original n complex nonlinear constraints have been changed into n simpler nonlinear constraints. And the number of Jacobian elements, that has a direct influence on much of the computational work both in GAMS and in CONOPT, has been reduced from n∗(n+1)/2 nonlinear elements to 3∗n-1 linear elements and only n nonlinear element. If f is an invertible increasing function you may even rewrite the last constraint as a simple bound:

```
cap.lo(i) = finv(b(i));
```

Some NLP solvers encourage you to move as many nonlinearities as possible into the objective which may make the objective very complex. This is neither recommended nor necessary with CONOPT. A special pre-processing step (discussed in section Iteration 1: Preprocessing in Appendix A) will aggregate parts of the model if it is useful for CONOPT without increasing the complexity in GAMS.

5.7.6.4 Scaling

Nonlinear as well as Linear Programming Algorithms use the derivatives of the objective function and the constraints to determine good search directions, and they use function values to determine if constraints are satisfied or not. The scaling of the variables and constraints, i.e. the units of measurement used for the variables and constraints, determine the relative size of the derivatives and of the function values and thereby also the search path taken by the algorithm.

Assume for example that two goods of equal importance both cost $1 per kg. The first is measured in gram, the second in tons. The coefficients in the cost function will be $1000/g and $0.001/ton, respectively. If cost is measured in $1000 units then the coefficients will be 1 and 1.e-6, and the smaller may be ignored by the algorithm since it is comparable to some of the zero tolerances.

CONOPT assumes implicitly that the model to be solved is well scaled. In this context well scaled means:

- Basic and superbasic solution values are expected to be around 1, e.g. from 0.01 to 100. Nonbasic variables will be at a bound, and the bound values should not be larger than say 100.
- Dual variables (or marginals) on active constraints are expected to be around 1, e.g. from 0.01 to 100. Dual variables on non-binding constraints will of course be zero.
- Derivatives (or Jacobian elements) are expected to be around 1, e.g. from 0.01 to 100.

Variables become well scaled if they are measured in appropriate units. In most cases you should select the unit of measurement for the variables so their expected value is around unity. Of course there will always be some variation. Assume x(i) is the production at location i. In most cases you should select the same unit of measurement for all components of x, for example a value around the average capacity.

Equations become well scaled if the individual terms are measured in appropriate units. After you have selected units for the variables you should select the unit of measurement for the equations so the expected values of the individual terms are around one. If you follow these rules, material balance equations will usually have coefficients of plus and minus one.

Derivatives will usually be well scaled whenever the variables and equations are well scaled. To see if the derivatives are well scaled, run your model with a positive Option LimRow and look for very large or very small coefficients in the equation listing in the GAMS output file.

CONOPT computes a measure of the scaling of the Jacobian, both in the initial and in the final point, and if it seems large it will be printed. The message looks like:

```
** WARNING **  The variance of the derivatives in the initial
               point is large (= 4.1 ). A better initial
               point, a better scaling, or better bounds on the
               variables will probably help the optimization.
```

The variance is computed as Sqrt(Sum(Log(Abs(jac(i)))**2)/nz) where jac(i) represents the nz nonzero derivatives (Jacobian elements) in the model. A variance of 4.1 corresponds to an average value of Log(jac)**2 of 4.1**2, which means that Jacobian values outside the range Exp(-4.1)=0.017 to Exp(+4.1)=60.4 are about as common at values inside. This range is for most models acceptable, while a variance of 5, corresponding to about half the derivatives outside the range Exp(-5)=0.0067 to Exp(+5)=148, can be dangerous.

Scaling of Intermediate Variables

Many models have a set of variables with a real economic or physical interpretation plus a set of intermediate or helping variables that are used to simplify the model. We have seen some of these in section Simple Expressions on Simple Expressions. It is usually rather easy to select good scaling units for the real variables since we know their order of magnitude from economic or physical considerations. However, the intermediate variables and their defining equations should preferably also be well scaled, even if they do not have an immediate interpretation. Consider the following model fragment where x, y, and z are variables and y is the intermediate variable:

```
Set p / p0*p4 /
Parameter a(p) / p0 211, p1 103, p2 42, p3 31, p4 6 /
yDef .. y =e= Sum(p, a(p)*Power(x,Ord(p)-1));
zDef .. z =e= Log(y);
```

x lies in the interval 1 to 10 which means that y will be between 211 and 96441 and Z will be between 5.35 and 11.47. Both x and z are reasonably scaled while y and the terms and derivatives in yDef are about a factor 1.e4 too large. Scaling y by 1.e4 and renaming it ys gives the following scaled version of the model fragment:

```
yDefs1 .. ys =e= Sum(p, a(p)*Power(x,Ord(p)-1))*1.e-4;
zDefs1 .. z  =e= Log(ys*1.e4);
```

The z equation can also be written as

```
zDefs2 .. z  =e= Log(ys) + Log(1.e4);
```

Note that the scale factor 1.e-4 in the yDefs1 equation has been placed on the right hand side. The mathematically equivalent equation

```
yDefs2 .. ys*1.e4 =e= Sum(p, a(p)*Power(x,Ord(p)-1));
```

will give a well scaled YS, but the right hand side terms of the equation and their derivatives have not changed from the original equation yDef and they are still far too large.

Using the Scale Option in GAMS

The rules for good scaling mentioned above are exclusively based on algorithmic needs. GAMS has been developed to improve the effectiveness of modelers, and one of the best ways seems to be to encourage modelers to write their models using a notation that is as "natural" as possible. The units of measurement is one part of this natural notation, and there is unfortunately often a conflict between what the modeler thinks is a good unit and what constitutes a well scaled model.

To facilitate the translation between a natural model and a well scaled model GAMS has introduced the concept of a scale factor, both for variables and equations. The notation and the definitions are quite simple. First of all, scaling is by default turned off. To turn it on, enter the statement <model>`.ScaleOpt = 1;` in your GAMS program somewhere after the `Model` statement and before the `Solve` statement. <model> is the name of the model to be solved. If you want to turn scaling off again, enter the statement <model>`.ScaleOpt = 0;` somewhere before the next `Solve`.

The scale factor of a variable or an equation is referenced with the suffix ".Scale", i.e. the scale factor of variable x(i) is referenced as x.Scale(i). Note that there is one scale value for each individual component of a multidimensional variable or equation. Scale factors can be defined in assignment statements with x.Scale(i) on the left hand side, and scale factors, both from variables and equations, can be used on the right hand side, for example to define other scale factors. The default scale factor is always 1, and a scale factor must be positive; GAMS will generate an execution time error if the scale factor is less than 1.e-20.

The mathematical definition of scale factors is a follows: The scale factor on a variable, V^s is used to related the variable as seen by the modeler, V^m, to the variable as seen by the algorithm, V^a, as follows:

$$V^m = V^a * V^s$$

This means, that if the variable scale, V^s, is chosen to represent the order of magnitude of the modeler's variable, V^m, then the variable seen by the algorithm, V^a, will be around 1. The scale factor on an equation, G^s, is used to related the equation as seen by the modeler, G^m, to the equation as seen by the algorithm, G^a, as follows:

$$G^m = G^a * G^s$$

This means, that if the equation scale, G^s, is chosen to represent the order of magnitude of the individual terms in the modelers version of the equation, G^m, then the terms seen by the algorithm, G^a, will be around 1.

The derivatives in the scaled model seen by the algorithm, i.e. dG^a/dV^a, are related to the derivatives in the modelers model, dG^m/dV^m, through the formula:

$$dG^a/dV^a = dG^m/dV^m * V^s/G^s$$

i.e. the modelers derivative is multiplied by the scale factor of the variable and divided by the scale factor of the equation. Note, that the derivative is unchanged if $V^s = G^s$. Therefore, if you have a GAMS equation like

```
G .. V =e= expression;
```

and you select $G^s = V^s$ then the derivative of V will remain 1. If we apply these rules to the example above with an intermediate variable we can get the following automatic scale calculation, based on an "average" reference value for x:

```
Scalar xRef; xRef = 6;
y.Scale = Sum(p, a(p)*Power(xRef,Ord(p)-1));
yDef.Scale = y.Scale;
```

or we could scale y using values at the end of the x interval and add safeguards as follows:

```
y.Scale = Max( Abs(Sum(p, a(p)*Power(x.Lo,Ord(p)-1))),
               Abs(Sum(p, a(p)*Power(x.Up,Ord(p)-1))),
               0.01 );
```

Lower and upper bounds on variables are automatically scaled in the same way as the variable itself. Integer and binary variables cannot be scaled.

GAMS' scaling is in most respects hidden for the modeler. The solution values reported back from a solution algorithm, both primal and dual, are always reported in the user's notation. The algorithm's versions of the equations and variables are only reflected in the derivatives in the equation and column listings in the GAMS output if `Option LimRow` and/or `LimCol` are positive, and in debugging output from the solution algorithm, generated with `Option SysOut = On`. In addition, the numbers in the algorithms iteration log will represent the scaled model: the infeasibilities and reduced gradients will correspond to the scaled model, and if the objective variable is scaled, the value of the objective function will be the scaled value.

A final warning about scaling of multidimensional variables is appropriate. Assume variable x(i,j,k) only appears in the model when the parameter ijk(i,j,k) is nonzero, and assume that Card(i) = Card(j) = Card(k) = 100 while Card(ijk) is much smaller than 100**2 = 1.e6. Then you should only scale the variables that appear in the model, i.e.

```
x.Scale(i,j,k)$ijk(i,j,k) = expression;
```

The statement

```
x.Scale(i,j,k) = expression;
```

will generate records for x in the GAMS database for all combinations of i, j, and k for which the expression is different from 1, i.e. up to 1.e6 records, and apart from spending a lot of time you will very likely run out of memory. Note that this warning also applies to non-default lower and upper bounds.

5.7.7 NLP and DNLP Models

GAMS has two classes of nonlinear model, NLP and DNLP. NLP models are defined as models in which all functions that appear with endogenous arguments, i.e. arguments that depend on model variables, are smooth with smooth derivatives. DNLP models can in addition use functions that are smooth but have discontinuous derivatives. The usual arithmetic operators (+, -, ∗, /, and ∗∗) can appear on both model classes.

The functions that can be used with endogenous arguments in a DNLP model and not in an NLP model are Abs, Min, and Max and as a consequence the indexed operators SMin and SMax.

Note that the offending functions can be applied to expressions that only involve constants such as parameters, var.l, and equ.m. Fixed variables are in principle constants, but GAMS makes its tests based on the functional form of a model, ignoring numerical parameter values and numerical bound values, and terms involving fixed variables can therefore not be used with Abs, Min, or Max in an NLP model.

The NLP solvers used by GAMS can also be applied to DNLP models. However, it is important to know that the NLP solvers attempt to solve the DNLP model as if it was an NLP model. The solver uses the derivatives of the constraints with respect to the variables to guide the search, and it ignores the fact that some of the derivatives may change discontinuously. There are at the moment no GAMS solvers designed specifically for DNLP models and no solvers that take into account the discontinuous nature of the derivatives in a DNLP model.

5.7.7.1 DNLP Models: What Can Go Wrong?

Solvers for NLP Models are all based on making marginal improvements to some initial solution until some optimality conditions ensure no direction with marginal improvements exist. A point with no marginally improving direction is called a Local Optimum.

The theory about marginal improvements is based on the assumption that the derivatives of the constraints with respect to the variables are a good approximations to the marginal changes in some neighborhood around the current point.

Consider the simple NLP model, Min Sqr(x), where x is a free variable. The marginal change in the objective is the derivative of Sqr(x) with respect to x, which is 2∗x. At x = 0, the marginal change in all directions is zero and x = 0 is therefore a Local Optimum.

Next consider the simple DNLP model, Min Abs(x), where x again is a free variable. The marginal change in the objective is still the derivative, which is +1 if x > 0 and -1 if x < 0. When x = 0, the derivative depends on whether we are going to increase or decrease x. Internally in the DNLP solver, we cannot be sure whether the derivative at 0 will be -1 or +1; it can depend on rounding tolerances. An NLP solver will start in some initial point, say x = 1, and look at the derivative, here +1. Since the derivative is positive, x is reduced to reduce the objective. After some iterations, x will be zero or very close to zero. The derivative will be +1 or -1, so the solver will try to change x. however, even small changes will not lead to a better objective function. The point x = 0 does not look like a Local Optimum, even though it is a Local Optimum. The result is that the NLP solver will muddle around for some time and then stop with a message saying something like: "The solution cannot be improved, but it does not appear to be optimal."

In this first case we got the optimal solution so we can just ignore the message. However, consider the following simple two-dimensional DNLP model: Min Abs(x1+x2) + 5∗Abs(x1-x2) with x1 and x2 free variables. Start the optimization from x1 = x2 = 1. Small increases in x1 will increase both terms and small decreases in x1 (by dx) will decrease the first term by dx but it will increase the second term by 5∗dx. Any change in x1 only is therefore bad, and it is easy to see that any change in x2 only also is bad. An NLP solver may therefore be stuck in the point x1 = x2 = 1, even though it is not a local solution: the direction (dx1,dx2) = (-1,-1) will lead to the optimum in x1 = x2 = 0. However, the NLP solver cannot distinguish what happens with this model from what happened in the previous model; the message will be of the same type: "The solution cannot be improved, but it does not appear to be optimal."

5.7.7.2 Reformulation from DNLP to NLP

The only reliable way to solve a DNLP model is to reformulate it as an equivalent smooth NLP model. Unfortunately, it may not always be possible. In this section we will give some examples of reformulations.

The standard reformulation approach for the ABS function is to introduce positive and negative deviations as extra variables: The term z = Abs(f(x)) is replaced by z = fPlus + fMinus, fPlus and fMinus are declared as positive variables and they are defined with the identity: f(x) =e= fPlus - fMinus. The discontinuous derivative from the Abs function has disappeared and the part of the model shown here is smooth. The discontinuity has been converted into lower bounds on the new variables, but bounds are handled routinely by any NLP solver. The feasible space is larger than before; f(x) = 5 can be obtained both with fPlus = 5, fMinus = 0, and z = 5, and with fPlus = 1000, fMinus = 995, and z = 1995. Provided the objective function has some term that tries to minimize z, either fPlus or fMinus will become zero and z will end with its proper value.

You may think that adding the smooth constraint fPlus $*$ fMinus =e= 0 would ensure that either fPlus or fMinus is zero. However, this type of so-called complementarity constraint is "bad" in any NLP model. The feasible space consists of the two half lines: (fPlus = 0 and fMinus $\geq$ 0) and (fPlus $\geq$ 0 and fMinus = 0). Unfortunately, the marginal change methods used by most NLP solvers cannot move from one half line to the other, and the solution is stuck at the half line it happens to reach first.

There is also a standard reformulation approach for the Max function. The equation z =e= Max(f(x),g(y)) is replace by the two inequalities, z =g= f(x) and z =g= g(y). Provided the objective function has some term that tries to minimize z, one of the constraints will become binding as equality and z will indeed be the maximum of the two terms.

The reformulation for the Min function is similar. The equation z =E= Min(f(x),g(y)) is replaced by the two inequalities, z =l= f(x) and z =l= g(y). Provided the objective function has some term that tries to maximize z, one of the constraints will become binding as equality and z is indeed the minimum of the two terms.

Max and Min can have more than two arguments and the extension should be obvious.

The non-smooth indexed operators, SMax and SMin can be handled using a similar technique: for example, z =E= SMax(i, f(x,i)) is replaced by the indexed inequality: inEq(i) .. z =l= f(x,i);

The reformulations that are suggested here all enlarge the feasible space. They require the objective function to move the final solution to the intersection of this larger space with the original feasible space. Unfortunately, the objective function is not always so helpful. If it is not, you may try using one of the smooth approximations described next. However, you should realize, that if the objective function cannot help the "good" approximations described here, then your overall model is definitely non-convex and it is likely to have multiple local optima.

5.7.7.3 Smooth Approximations

Smooth approximations to the non-smooth functions ABS, MAX, and MIN are approximations that have function values close to the original functions, but have smooth derivatives.

A smooth GAMS approximation for Abs(f(x)) is

```
Sqrt( Sqr(f(x)) + Sqr(delta) )
```

where delta is a small scalar. The value of delta can be used to control the accuracy of the approximation and the curvature around f(x) = 0. The approximation error is largest when f(x) is zero, in which case the error is delta. The error is reduced to approximately Sqr(delta)/2 for f(x) = 1. The second derivative is 1/delta at f(x) = 0 (excluding terms related to the second derivative of f(x)). A delta value between 1.e-3 and 1.e-4 should in most cases be appropriate. It is possible to use a larger value in an initial optimization, reduce it and solve the model again. You should note, that if you reduce delta below 1.e-4 then large second order terms might lead to slow convergence or even prevent convergence.

The approximation shown above has its largest error when f(x) = 0 and smaller errors when f(x) is far from zero. If it is important to get accurate values of Abs exactly when f(x) = 0, then you may use the alternative approximation

```
Sqrt( Sqr(f(x)) + Sqr(delta) ) - delta
```

instead. The only difference is the constant term. The error is zero when f(x) is zero and the error grows to -delta when f(x) is far from zero.

Some theoretical work uses the Huber, H(∗), function as an approximation for Abs. The Huber function is defined as

```
H(x) =  x for x  >   delta,
H(x) = -x for x  <  -delta and
H(x) = Sqr(x)/2/delta + delta/2 for -delta  <  x  <  delta.
```

Although the Huber function has some nice properties, it is for example accurate when Abs(x) > delta, it is not so useful for GAMS work because it is defined with different formula for the three pieces.

A smooth GAMS approximation for Max(f(x),g(y)) is

```
( f(x) + g(y) + Sqrt( Sqr(f(x)-g(y)) + Sqr(delta) ) )/2
```

where delta again is a small scalar. The approximation error is delta/2 when f(x) = g(y) and decreases with the difference between the two terms. As before, you may subtract a constant term to shift the approximation error from the area f(x) = g(y) to areas where the difference is large. The resulting approximation becomes

```
( f(x) + g(y) + Sqrt( Sqr(f(x)-g(y)) + Sqr(delta) ) - delta )/2
```

Similar smooth GAMS approximations for Min(f(x),g(y)) are

```
( f(x) + g(y) - Sqrt( Sqr(f(x)-g(y)) + Sqr(delta) ) )/2
```

and

```
( f(x) + g(y) - Sqrt( Sqr(f(x)-g(y)) + Sqr(delta) ) + delta )/2
```

Appropriate delta values are the same as for the Abs approximation: in the range from 1.e-2 to 1.e-4.

It appears that there are no simple symmetric extensions for Max and Min of three or more arguments or for indexed SMax and SMin.

5.7.7.4 Are DNLP Models Always Non-smooth?

A DNLP model is defined as a model that has an equation with an Abs, Max, or Min function with endogenous arguments. The non-smooth properties of DNLP models are derived from the non-smooth properties of these functions through the use of the chain rule. However, composite expressions involving Abs, Max, or Min can in some cases have smooth derivatives and the model can therefore in some cases be smooth.

One example of a smooth expression involving an Abs function is common in water systems modeling. The pressure loss over a pipe, dH, is proportional to the flow, Q, to some power, P. P is usually around +2. The sign of the loss depend on the direction of the flow so dH is positive if Q is positive and negative if Q is negative. Although GAMS has a Sign function, it cannot be used in a model because of its discontinuous nature. Instead, the pressure loss can be modeled with the equation dH =E= const * Q * Abs(Q)**(P-1), where the sign of the Q-term takes care of the sign of dH, and the Abs function guaranties that the real power ** is applied to a non-negative number. Although the expression involves the Abs function, the derivatives are smooth as long as P is greater than 1. The derivative with respect to Q is const * (P-1) * Abs(Q)**(P-1) for $Q > 0$ and -const * (P-1) * Abs(Q)**(P-1) for $Q < 0$. The limit for Q going to zero from both right and left is 0, so the derivative is smooth in the critical point $Q = 0$ and the overall model is therefore smooth.

Another example of a smooth expression is the following terribly looking Sigmoid expression:

```
Sigmoid(x) = Exp( Min(x,0) ) / (1+Exp(-Abs(x)))
```

The standard definition of the sigmoid function is

```
Sigmoid(x) = Exp(x) / ( 1+Exp(x) )
```

This definition is well behaved for negative and small positive x, but it not well behaved for large positive x since Exp overflows. The alternative definition:

```
Sigmoid(x) = 1 / ( 1+Exp(-x) )
```

is well behaved for positive and slightly negative x, but it overflows for very negative x. Ideally, we would like to select the first expression when x is negative and the second when x is positive, i.e.

```
Sigmoid(x) = (Exp(x)/(1+Exp(x)))$(x < 0) + (1/(1+Exp(-x)))$(x > 0)
```

but a $ -control that depends on an endogenous variable is illegal. The first expression above solves this problem. When x is negative, the nominator becomes Exp(x) and the denominator becomes 1+Exp(x). And when x is positive, the nominator becomes Exp(0) = 1 and the denominator becomes 1+Exp(-x). Since the two expressions are mathematically identical, the combined expression is of course smooth, and the Exp function is never evaluated for a positive argument.

Unfortunately, GAMS cannot recognize this and similar special cases so you must always solve models with endogenous Abs, Max, or Min as DNLP models, even in the cases where the model is smooth.

5.7.7.5 Are NLP Models Always Smooth?

NLP models are defined as models in which all operators and functions are smooth. The derivatives of composite functions, that can be derived using the chain rule, will therefore in general be smooth. However, it is not always the case. The following simple composite function is not smooth: y = Sqrt(Sqr(x)). The composite function is equivalent to y = Abs(x), one of the non-smooth DNLP functions.

What went wrong? The chain rule for computing derivatives of a composite function assumes that all intermediate expressions are well defined. However, the derivative of Sqrt grows without bound when the argument approaches zero, violating the assumption.

There are not many cases that can lead to non-smooth composite functions, and they are all related to the case above: The real power, x**y, for $0 < y < 1$ and x approaching zero. The Sqrt function is a special case since it is equivalent to x**y for y = 0.5.

If you have expressions involving a real power with an exponent between 0 and 1 or a Sqrt, you should in most cases add bounds to your variables to ensure that the derivative or any intermediate terms used in their calculation become undefined. In the example above, Sqrt(Sqr(x)), a bound on x is not possible since x should be allowed to be both positive and negative. Instead, changing the expression to Sqrt(Sqr(x) + Sqr(delta)) may lead to an appropriate smooth formulation.

Again, GAMS cannot recognize the potential danger in an expression involving a real power, and the presence of a real power operator is not considered enough to flag a model as a DNLP model. During the solution process, the NLP solver will compute constraint values and derivatives in various points within the bounds defined by the modeler. If these calculations result in undefined intermediate or final values, a function evaluation error is reported, an error counter is incremented, and the point is flagged as a bad point. The following action will then depend on the solver. The solver may try to continue, but only if the modeler has allowed it with an `Option DomLim = xxx;`. The problem of detecting discontinuities is changed from a structural test at the GAMS model generation stage to a dynamic test during the solution process.

You may have a perfectly nice model in which intermediate terms become undefined. The composite function Sqrt(Power(x,3)) is mathematically well defined around x = 0, but the computation will involve the derivative of Sqrt at zero, that is undefined. It is the modeler's responsibility to write expressions in a way that avoids undefined intermediate terms in the function and derivatives computations. In this case, you may either add a small strictly positive lower bound on x or rewrite the function as x**1.5

5.7.8 APPENDIX A: Algorithmic Information

The objective of this Appendix is to give technically oriented users some understanding of what CONOPT is doing so they can get more information out of the iteration log. This information can be used to prevent or circumvent algorithmic difficulties or to make informed guesses about which options to experiment with to improve CONOPT's performance on particular model classes.

5.7.8.1 Overview of CONOPT

CONOPT is a GRG-based algorithm specifically designed for large nonlinear programming problems expressed in the following form

```
min or max      f(x)            (1)
subject to      g(x) = b        (2)
lo < x < up                     (3)
```

where x is the vector of optimization variables, lo and up are vectors of lower and upper bounds, some of which may be minus or plus infinity, b is a vector of right hand sides, and f and g are differentiable nonlinear functions that define the model. n will in the following denote the number of variables and m the number of equations. (2) will be referred to as the (general) constraints and (3) as the bounds.

The relationship between the mathematical model in (1)-(3) above and the GAMS model is simple: The inequalities defined in GAMS with `=l=` or `=g=` are converted into equalities by addition of properly bounded slacks. Slacks with lower and upper bound of zero are added to all GAMS equalities to ensure that the Jacobian matrix, i.e. the matrix of derivatives of the functions g with respect to the variables x, has full row rank. All these slacks are together with the normal GAMS variables included in x. lo represent the lower bounds defined in GAMS, either implicitly with the Positive Variable declaration, or explicitly with the Var.Lo notation, as well as any bounds on the slacks. Similarly, up represent upper bounds defined in GAMS, e.g. with the Var.Up notation, as well as any bounds on the slacks. g represent the non-constant terms of the GAMS equations themselves; non-constant terms appearing on the right hand side are by GAMS moved to the left hand side and constant terms on the left hand side are moved to the right. The objective function f is simply the GAMS variable to be minimized or maximized.

Additional comments on assumptions and design criteria can be found in the Introduction to the main text.

5.7.8.2 The CONOPT Algorithm

The algorithm used in CONOPT is based on the GRG algorithm first suggested by Abadie and Carpentier (1969). The actual implementation has many modifications to make it efficient for large models and for models written in the GAMS language. Details on the algorithm can be found in Drud (1985 and 1992). Here we will just give a short verbal description of the major steps in a generic GRG algorithm. The later sections in this Appendix will discuss some of the enhancements in CONOPT that make it possible to solve large models.

The key steps in any GRG algorithm are:

1. Initialize and Find a feasible solution.
2. Compute the Jacobian of the constraints, J.
3. Select a set of n basic variables, x_b, such that B, the sub- matrix of basic column from J, is nonsingular. Factorize B. The remaining variables, x_n, are called nonbasic.
4. Solve $B^T u = df/dx_b$ for the multipliers u.
5. Compute the reduced gradient, $r = df/dx - J^T u$. r will by definition be zero for the basic variables.
6. If r projected on the bounds is small, then stop. The current point is close to optimal.
7. Select the set of superbasic variables, x_s, as a subset of the nonbasic variables that profitably can be changed, and find a search direction, d_s, for the superbasic variables based on r_s and possibly on some second order information.
8. Perform a line search along the direction d. For each step, x_s is changed in the direction d_s and x_b is subsequently adjusted to satisfy $g(x_b, x_s) = b$ in a pseudo-Newton process using the factored B from step 3.
9. Go to 2.

The individual steps are of course much more detailed in a practical implementation like CONOPT. Step 1 consists of several pre-processing steps as well as a special Phase 0 and a scaling procedure as described in the following sections Iteration 0: The Initial Point to Finding a Feasible Solution: Phase 0 . The optimizing steps are specialized in several versions according to the whether the model appears to be almost linear or not. For "almost" linear models some of the linear algebra work involving the matrices J and B can be avoided or done using cheap LP-type updating techniques, second order information is not relevant in step 7, and the line search in step 8 can be improved by observing that the optimal step as in LP almost always will be determined by the first variable that reaches a bound. Similarly, when the model appears to be fairly nonlinear other aspects can be optimized: the set of basic variables will often remain constant over several iterations, and other parts of the sparse matrix algebra will take advantage of this (section Finding a Feasible Solution: Phase 1 and 2 and Linear and Nonlinear Mode: Phase 1 to 4). If the model is "very" linear an improved search direction (step 7) can be computed using specialized inner LP-like iterations (section Linear Mode: The SLP Procedure) and a steepest edge procedure can be useful for certain models that needs very many iterations (section Linear Mode: The Steepest Edge Procedure). If the model is "very" nonlinear and has many degrees of freedom an improved search direction (step 7) can be computed using specialized inner SQP-like iterations based on exact second derivatives for the model (section Nonlinear Mode: The SQP Procedure).

The remaining two sections give some short guidelines for selecting non-default options (section How to Select Non-default Options), and discuss miscellaneous topics (section Miscellaneous Topics) such as CONOPT's facilities for strictly triangular models (section Triangular Models) and for square systems of equations, in GAMS represented by the model class called CNS or Constrained Nonlinear System (section Constrained Nonlinear System or Square Systems of Equations), as well as numerical difficulties due to loss of feasibility (section Loss of Feasibility) and slow or no progress due to stalling (section Stalling).

5.7.8.3 Iteration 0: The Initial Point

The first few "iterations" in the iteration log (See section Iteration Output), in the main text for an example) are special initialization iterations, but they have been counted as real iterations to allow the user to interrupt at various stages during initialization. Iteration 0 corresponds to the input point exactly as it was received from GAMS. The sum of infeasibilities in the column labeled "Infeasibility" includes all residuals, also from the objective constraint where "z =e= expression" will give rise to the term Abs(z - expression) that may be nonzero if z has not been initialized. You may stop CONOPT after iteration 0 with `Option IterLim = 0;` in GAMS. The solution returned to GAMS will contain the input point and the values of the constraints in this point. The marginals of both variables and equations have not yet been computed and they will be returned as EPS.

This possibility can be used for debugging when you have a reference point that should be feasible, but is infeasible for unknown reasons. Initialize all variables to their reference values, also all intermediate variables, and call CONOPT with IterLim = 0. Then compute and display the following measures of infeasibility for each block of constraints, represented by the generic name eq:

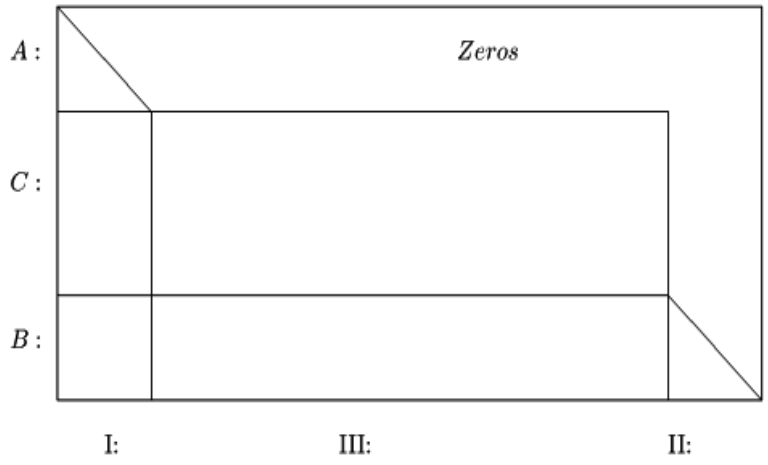
```
=e= constraints: Round(Abs(eq.L - eq.Lo),3)
=l= constraints: Round(Min(0,eq.L - eq.Up),3)
=g= constraints: Round(Min(0,eq.Lo - eq.L),3)
```

The Round function rounds to 3 decimal places so GAMS will only display the infeasibilities that are larger than 5.e-4.

Similar information can be derived from inspection of the equation listing generated by GAMS with `Option LimRow = nn;`, but although the method of going via CONOPT requires a little more work during implementation it can be convenient in many cases, for example for large models and for automated model checking.

5.7.8.4 Iteration 1: Preprocessing

Iteration 1 corresponds to a pre-processing step. Constraint-variable pairs that can be solved a priori (so-called pre-triangular equations and variables) are solved and the corresponding variables are assigned their final values. Constraints that always can be made feasible because they contain a free variable with a constant coefficient (so-called post-triangular equation-variable pairs) are excluded from the search for a feasible solution and from the Infeasibility measure in the iteration log. Implicitly, equations and variables are ordered as shown in Figure 1.



Preprocessing: Pre-triangular Variables and Constraints

The pre-triangular equations are those labeled A in Figure 1. They are solved one by one along the "diagonal" with respect to the pre-triangular variables labeled I. In practice, CONOPT looks for equations with only one non-fixed variable. If such an equation exists, CONOPT tries to solve it with respect to this non-fixed variable. If this is not possible the overall model is infeasible, and the exact reason for the infeasibility is easy to identify as shown in the examples below. Otherwise, the final value of the variable has been determined, the variable can for the rest of the optimization be considered fixed, and the equation can be removed from further consideration. The result is that the model has one equation and one non-fixed variable less. As variables are fixed new equations with only one non-fixed variable may emerge, and CONOPT repeats the process until no more equations with one non-fixed variable can be found.

This pre-processing step will often reduce the effective size of the model to be solved. Although the pre-triangular variables and equations are removed from the model during the optimization, CONOPT keeps them around until the final solution is found. The dual variables for the pre-triangular equations are then computed so they become available in GAMS.

CONOPT has a special option for analyzing and solving completely triangular models. This option is described in section Triangular Models .

The following small GAMS model shows an example of a model with pre-triangular variables and equations:

```
Variable x1, x2, x3, obj;
Equation e1, e2, e3;
e1 .. Log(x1) + x2 =e= 1.6;
e2 .. 5 * x2 =e= 3;
e3 .. obj =e= Sqr(x1) + 2 * Sqr(x2) + 3 * Sqr(x3);
x1.Lo = 0.1;
Model demo / All /; Solve demo using NLP Minimizing obj;
```

Equation e2 is first solved with respect to x2 (result 3/5 = 0.6). It is easy to solve the equation since x2 appears linearly, and the result will be unique. x2 is then fixed and the equation is removed. Equation e1 is now a candidate since x1 is the only remaining non- fixed variable in the equation. Here x1 appears nonlinear and the value of x1 is found using an iterative scheme based on Newton's method. The iterations are started from the value provided by the modeler or from the default initial value. In this case x1 is started from the default initial value, i.e. the lower bound of 0.1, and the result after some iterations is x1 = 2.718 = Exp(1).

During the recursive solution process it may not be possible to solve one of the equations. If the lower bound on x1 in the model above is changed to 3.0 you will get the following output:

```
** An equation in the pre-triangular part of the model cannot
   be solved because the critical variable is at a bound.

   Residual=           9.86122887E-02
   Tolerance (RTNWTR)= 6.34931126E-07

   e1: Infeasibility in pre-triangular part of model.
   x1: Infeasibility in pre-triangular part of model.

   The solution order of the critical equations and
   variables is:

   e2 is solved with respect to
   x2. Solution value =  6.0000000000E-01

   e1 could not be solved with respect to
   x1. Final solution value =  3.0000000000E+00
   e1 remains infeasible with residual = 9.8612288668E-02
```

The problem is as indicated that the variable to be solved for is at a bound, and the value suggested by Newton's method is on the infeasible side of the bound. The critical variable is x1 and the critical equation is e1, i.e. x1 tries to exceed its bound when CONOPT solves equation e1 with respect to x1. To help you analyze the problem, especially for larger models, CONOPT reports the solution sequence that led to the infeasibility: In this case equation e2 was first solved with respect to variable x2, then equation e1 was attempted to be solved with respect to x1 at which stage the problem appeared. To make the analysis easier CONOPT will always report the minimal set of equations and variables that caused the infeasibility.

Another type of infeasibility is shown by the following model:

```
Variable x1, x2, x3, obj;
Equation e1, e2, e3;
e1 .. Sqr(x1) + x2 =e= 1.6;
e2 .. 5 * x2 =e= 3;
e3 .. obj =e= Sqr(x1) + 2 * Sqr(x2) + 3 * Sqr(x3);
Model demo / All /; Solve demo using NLP Minimizing obj;
```

where Log(x1) has been replaced by Sqr(x1) and the lower bound on x1 has been removed. This model gives the message:

```
** An equation in the pre-triangular part of the model cannot
   be solved because of too small pivot.
   Adding a bound or initial value may help.

   Residual=           4.0000000
```

```
Tolerance (RTNWTR)= 6.34931126E-07

e1: Infeasibility in pre-triangular part of model.
x1: Infeasibility in pre-triangular part of model.

The solution order of the critical equations and
variables is:

e2 is solved with respect to
x2. Solution value =  6.0000000000E-01

e1 could not be solved with respect to
x1. Final solution value =  0.0000000000E+00
e1 remains infeasible with residual =-4.0000000000E+00
```

After equation e2 has been solved with respect to x2, equation e1 that contains the term Sqr(x) should be solved with respect to x1. The initial value of x1 is the default value zero. The derivative of e1 with respect to x1 is therefore zero, and it is not possible for CONOPT to determine whether to increase or decrease x1. If x1 is given a nonzero initial value the model will solve. If x1 is given a positive initial value the equation will give x1 = 1, and if x1 is given a negative initial value the equation will give x1 = -1. The last type of infeasibility that can be detected during the solution of the pre-triangular or recursive equations is shown by the following example

```
Variable x1, x2, x3, obj;
Equation e1, e2, e3;
e1 .. Log(x1) + x2 =e= 1.6;
e2 .. 5 * x2 =e= 3;
e3 .. obj =e= Sqr(x1) + 2 * Sqr(x2) + 3 * Sqr(x3);
e4 .. x1 + x2 =e= 3.318;
x1.Lo = 0.1;
Model demo / All /; Solve demo using NLP Minimizing obj;
```

that is derived from the first model by the addition of equation e4. This model produces the following output

```
** An equation is inconsistent with other equations in the
   pre-triangular part of the model.

   Residual=           2.81828458E-04
   Tolerance (RTNWTR)= 6.34931126E-07


   The pre-triangular feasibility tolerance may be relaxed with
   a line:

               SET       RTNWTR    X.XX

   in the CONOPT control program.

   e4: Inconsistency in pre-triangular part of model.

   The solution order of the critical equations and
   variables is:


   e2 is solved with respect to
```

```
x2. Solution value =  6.0000000000E-01

e1 is solved with respect to
x1. Solution value =  2.7182818285E+00

All variables in equation e4 are now fixed
and the equation is infeasible. Residual = 2.8182845830E-04
```

First e2 is solved with respect to x2, then e1 is solved with respect to x1 as indicated by the last part of the output. At this point all variables that appear in equation e4, namely x1 and x2, are fixed, but the equation is not feasible. e4 is therefore inconsistent with e1 and e2 as indicated by the first part of the output. In this case the inconsistency is fairly small, 2.8E-04, so it could be a tolerance problem. CONOPT will always report the tolerance that was used, `rtnwtr` - the triangular Newton tolerance, and if the infeasibility is small it will also tell how the tolerance can be relaxed. Section The CONOPT Options File gives further details on how to change tolerances, and a complete list of options is given in Appendix B.

You can turn the identification and solution of pre-triangular variables and equations off by adding the line `lspret = f` in the CONOPT control program. This can be useful in some special cases where the point defined by the pre-triangular equations gives a function evaluation error in the remaining equations. The following example shows this:

```
Variable x1, x2, x3, x4, obj;
Equation e1, e2, e3, e4;
e1 .. Log(1+x1) + x2 =e= 0;
e2 .. 5 * x2 =e= -3;
e3 .. obj =e= 1*Sqr(x1) + 2*Sqrt(0.01 + x2 - x4) + 3*Sqr(x3);
e4 .. x4 =l= x2;
MOdel fer / All /; Solve fer Minimizing obj using NLP;
```

All the nonlinear functions are defined in the initial point in which all variables have their default value of zero. The pre-processor will compute x2 = -0.6 from e2 and x1 = 0.822 from e1. When CONOPT continues and attempts to evaluate e3, the argument to the Sqrt function is negative when these new triangular values are used together with the initial x4 = 0, and CONOPT cannot backtrack to some safe point since the function evaluation error appears the first time e3 is evaluated. When the pre-triangular preprocessor is turned off, x2 and x4 are changed at the same time and the argument to the Sqrt function remains positive throughout the computations. Note, that although the purpose of the e4 inequality is to guarantee that the argument of the Sqrt function is positive in all points, and although e4 is satisfied in the initial point, it is not satisfied after the pre-triangular constraints have been solved. Only simple bounds are strictly enforced at all times. Also note that if the option `lspret = f` is used then feasible linear constraints will in fact remain feasible.

An alternative (and preferable) way of avoiding the function evaluation error is to define an intermediate variable equal to 0.01+x2-x4 and add a lower bound of 0.01 on this variable. The inequality e4 could then be removed and the overall model would have the same number of constraints.

Preprocessing: Post-triangular Variables and Constraints

Consider the following fragment of a larger GAMS model:

```
Variable util(t)  Utility in period t
         totutil  Total Utility;
Equation utilDef(T) Definition of Utility
         tutilDef   Definition of Total Utility;
utilDef(T).. util(t) =e= nonlinear function of other variables;
tutilDef  .. totutil =e= Sum( t , util(t) / (1+r)**Ord(t) );
Model demo / All /; Solve demo Maximizing totutil using NLP;
```

The part of the model shown here is easy to read and from a modeling point of view it should be considered well written. However, it could be more difficult to solve than a model in which variable util(t) was substituted out because all the utilDef equations are nonlinear constraints that the algorithms must ensure are satisfied.

To make well written models like this easy to solve CONOPT will move as many nonlinearities as possible from the constraints to the objective function. This automatically changes the model from the form that is preferable for the modeler to the form that is preferable for the algorithm. In this process CONOPT looks for free variables that only appear in one equation outside the objective function. If such a variable exists and it appears linearly in the equation, like util(t) appears with coefficient 1 in equation utilDef(t), then the equation can always be solved with respect to the variable. This means that the variable logically can be substituted out of the model and the equation can be removed. The result is a model that has one variable and one equation less, and a more complex objective function. As variables and equations are substituted out, new candidates for elimination may emerge, so CONOPT repeats the process until no more candidates exist.

This so-called post-triangular preprocessing step will often move several nonlinear constraints into the objective function where they are much easier to handle, and the effective size of the model will decrease. In some cases the result can even be a model without any general constraints. The name post-triangular is derived from the way the equations and variables appear in the permuted Jacobian in Figure 1. The post-triangular equations and variables are the ones on the lower right hand corner labeled B and II, respectively.

In the example above, the util variables will be substituted out of the model together with the nonlinear utilDef equations provided the util variables are free and do not appear elsewhere in the model. The resulting model will have fewer nonlinear constraints, but more nonlinear terms in the objective function.

Although you may know that the nonlinear functions on the right hand side of utilDef always will produce positive util values, you should in general not declare util to be a Positive Variable. If you do, CONOPT may not be able to eliminate util(t), and the model will be harder to solve. It is of course unfortunate that a redundant bound changes the solution behavior, and to reduce this problem CONOPT will try to estimate the range of nonlinear expressions using interval arithmetic. If the computed range of the right hand side of the utilDef constraint is within the bounds of util, then these bounds cannot be binding and util is a so-called implied free variable that can be eliminated.

The following model fragment from a least squares model shows another case where the preprocessing step in CONOPT is useful:

```
Variable residual(case)  Residuals
         ssq             Sum of Squared Residuals;
Equation eqEst(case)     Equation to be estimated
         ssqDef          Definition of objective;
eqEst(case).. residual(case) =e= expression in other variables;
SSQDEF     .. ssq =e= Sum( case, Sqr( residual(case) ) );
Model lsqLarge / All /; Solve lsqLarge using NLP Minimizing ssq;
```

CONOPT will substitute the residual variables out of the model using the eqEst equations. The model solved by CONOPT is therefore mathematically equivalent to the following GAMS model

```
Variable ssq     Sum of Squared Residuals;
Equation ssqd    Definition of objective;
Ssqd .. ssq =e= Sum( case, Sqr(expression in other variables));
Model lsqSmall / All /;
Solve lsqSmall using NLP Minimizing ssq;
```

However, if the "expression in other variables" is a little complicated, e.g. if it depends on several variables, then the first model, lsqLarge, will be much faster to generate with GAMS because its derivatives in equation qEst and ssqDef are much simpler than the derivatives in the combined ssqd equation in the second model, lsqSmall. The larger model will therefore be faster to generate, and it will also be faster to solve because the computation of both first and second derivatives will be faster.

Note that the comments about what are good model formulations are dependent on the preprocessing capabilities in CONOPT. Other algorithms may prefer models like lsqSmalL over lsqLarge. Also note that the variables and equations that are substituted out are still indirectly part of the model. CONOPT evaluates the equations and computes values for the variables each time the value of the objective function is needed, and their values are available in the GAMS solution.

It is not necessary to have a coefficient of 1 for the variable to be substituted out in the post-triangular phase. However, a non-zero coefficient cannot be smaller than the absolute pivot tolerance used by CONOPT, `rtpiva`.

The number of pre- and post-triangular equations and variables is printed in the log file between iteration 0 and 1 as shown in the iteration log in section Iteration Output of the main text. The sum of infeasibilities will usually decrease from iteration 0 to 1 because fewer constraints usually will be infeasible. However, it may increase as shown by the following example:

```
Positive Variable x, y, z;
Equation e1, e2;
e1.. x =e= 1;
e2.. 10*x - y + z =e= 0;
```

started from the default values x.L = 0, y.L = 0, and z.L = 0. The initial sum of infeasibilities is 1 (from e1 only). During pre-processing x is selected as a pre-triangular variable in equation e1 and it is assigned its final value 1 so e1 becomes feasible. After this change the sum of infeasibilities increases to 10 (from e2 only).

You may stop CONOPT after iteration 1 with `Option IterLim = 1;` in GAMS. The solution returned to GAMS will contain the pre-processed values for the variables that can be assigned values from the pre-triangular equations, the computed values for the variables used to solve the post-triangular equations, and the input values for all other variables. The pre- and post-triangular constraints will be feasible, and the remaining constraints will have values that correspond to this point. The marginals of both variables and equations have not been computed yet and will be returned as EPS.

The crash procedure described in the following sub-section is an optional part of iteration 1.

Preprocessing: The Optional Crash Procedure

In the initial point given to CONOPT the variables are usually split into a group with initial value provided by the modeler (in the following called the assigned variables) and a group of variables for which no initial value has been provided (in the following called the default variables). The objective of the optional crash procedure is to find a point in which as many of the constraints as possible are feasible, primarily by assigning values to the default variables and by keeping the assigned variables at their initial values. The implicit assumption in this procedure is that if the modeler has assigned an initial value to a variable then this value is "better" then a default initial value.

The crash procedure is an extension of the triangular pre-processing procedure described above and is based on a simple heuristic: As long as there is an equation with only one non-fixed variable (a singleton row) then we should assign a value to the variable so the equation is satisfied or satisfied as closely as possible, and we should then temporarily fix the variable. When variables are fixed additional singleton rows may emerge and we repeat the process. When there are no singleton rows we fix one or more variables at their initial value until a singleton row appears, or until all variables have been fixed. The variables to

be fixed at their initial value are selected using a heuristic that both tries to create many row singletons and tries to select variables with "good values". Since the values of many variables will come to depend in the fixed variables, the procedure favors assigned variables and among these it favors variables that appear in many feasible constraints.

Figure 2 shows a reordered version of Figure 1. The variables labeled IV are the variables that are kept at their initial values, primarily selected from the assigned variables. The equations labeled C are then solved with respect to the variables labeled III, called the crash-triangular variables. The crash-triangular variables will often be variables without initial values, e.g. intermediate variables. The number of crash-triangular variables is shown on the iteration output between iteration 0 and 1, but only if the crash procedure is turned on.

The result of the crash procedure is an updated initial point in which usually a large number of equations will be feasible, namely all equations labeled A, B, and C in Figure 2. There is, as already shown with the small example in Section Preprocessing: Post-triangular Variables and Constraints above, no guarantee that the sum of infeasibilities will be reduced, but it is often the case, and the point will often provide a good starting point for the following procedures that finds an initial feasible solution.

The crash procedure is activated by adding the line `lstcrs=t` in the options file. The default value of `lstcrs` (`lstcrs` = Logical Switch for Triangular Crash) is `f` or `false`, i.e. the crash procedure is not normally used. Before turning the crash procedure on you must turn the definitional equations (see next sub-section) off.

Preprocessing: Definitional Equations

From version 3.16 CONOPT has introduced the concept of Definitional Equations. In section Simple Expressions it was recommended to introduce intermediate variables to simplify complex expressions. If the intermediate variables are free or if the bounds defined by the modeler cannot be binding then we call the constraints that define the intermediate variables "Definitional Equations" and the intermediate variables "Defined Variables". Some models have a large number of definitional equations, and CONOPT tries to recognize them and take advantage of the special structure. Defined variables that only appear in or feed into the objective are recognized as post-triangular variables as discussed in section Preprocessing: Post-triangular Variables and Constraints but defined variables can also be used in the simultaneous constraints. The picture is similar to fig 2. with the C-rows representing the definitional constraints and the III-variables the defined variables. The main differences between crash- triangular variables and defined variable are that (1) defined variables are free or have non-binding bounds and the definitional equations can therefore always be made feasible, (2) defined variables are cheaper to recognize, (3) since they have a natural interpretation, using them is probably more numerical stable, and (4) there are most likely fewer defined variables than crash-triangular variables.

The number of definitional equations is printed in the log file between iteration 0 and 1 if CONOPT finds any. The definitional equations are used to give the defined variables new values, so it is no longer so important the give intermediate variables initial values. In the process the sum of infeasibilities may grow but CONOPT consider it more important to keep these constraints feasible. The defined variables are also made basic and they will most likely stay basic throughout the solution process.

There are three new options introduced to controls the detection of definitional equations: `lsusdf`, `lsuqdf`, and `lfusdf`. They are described in Appendix B. By default CONOPT will only look for unique definitional constraints, but the options allow the user to experiment with a more aggressive strategy.

From version 3.16F there is an additional option, `lmusdf`, used to control how the definitional equations are used during the optimization. The option is summarized in Appendix B. By default, definitional equations are only used to define initial values and an initial basis and their special properties are ignored during the optimization. If there are many definitional equations and they are fairly nonlinear then it can sometimes be beneficial to force the definitional variables to remain basic and to use the definitional equations to compute values for the defined variables in all intermediate trial points. This behavior is turned on with `lmusdf = 1`. The point where `lmusdf = 1` start to pay off depends on the model and the degree of nonlinearity, but a guess is that the number of definitional equations should exceed half the number of equations remaining after the pre- and post-triangular equations have been removed before it is worth while.

Preprocessing and Function evaluation errors

Function evaluation errors may occure during pre-processing is an equation that is a candidate for becoming pre-triangular. Since GAMS requires all functions to be defined in the starting point this can only happen if a previously solved pre-triangular variable has recived new a value that makes the function undefined. An example is:

```
positive variable x1, x2
equation e1, e2;
e1 .. x1 =E= 10;
e2 .. x2 =E= log( 5-x1 );
```

The pre-triangular variable x1 is changed to 10, a value that is inconsistent with equation e2. You can see the changes that CONOPT has made during pre-processing with the option `prprec = true` and this should help identify why the model does not solve.

After identifying and solving the pre-triangular part of the model CONOPT looks for implied bounds, and if an initial variable violates such an implied bound, CONOPT will move it inside the bound. This change can give rise to a function evaluation error in one of the remaining constraints. An example is:

```
positive variable x1, x2, x3, x4;
equation e1, e2;
e1.. x1 + x2 =L= 10;
e2.. x4 =E= log(x1+x3-20);
x1.l = 100; x3.up = 100;
```

From constraint e1 we can derive that both x1 and x2 must have (implied) upper bounds of 10. Since the initial value of x1 is above this implied bound it is changed by the pre-processor. After the pre-processor finishes, e2 will give a function evaluation error. Again, option `prprec = true` will give output that describes the changes in the variables which should help identify the problem.

5.7.8.5 Iteration 2: Scaling

Iteration 2 is the last dummy iteration during which the model is scaled, if scaling is turned on. The default is to turn scaling on. The Infeasibility column shows the scaled sum of infeasibilities. You may again stop CONOPT after iteration 2 with `Option IterLim = 2;` in GAMS, but the solution that is reported in GAMS will have been scaled back again so there will be no change from iteration 1 to iteration 2.

The following description of the automatic scaling procedure is included for completeness. Experiments have so far given mixed results with some advantage for scaling, and scaling is therefore by default turned on, corresponding to the CONOPT option `lsscal = t`. Users are recommended to be cautious with the automatic scaling procedure. If scaling is a problem, try to use manual scaling or scaling in GAMS (see section Scaling in the main text) based on an understanding of the model.

The scaling procedure multiplies all variables in group III and all constraints in group C (see Figure 1) by scale factors computed as follows:

1. CONOPT computes the largest term for each constraint, i. This is defined as the maximum of the constant right hand side, the slack (if any), and Abs(jac(i,j)∗x(j)) where jac(i,j) is the derivative and x(j) is the variable.

2. The constraint scale factor is defined as the largest term in the constraint, projected on the interval `[rtmins, rtmaxs]`. The constraint is then divided by the constraint scale factor. Ignoring the projection, the result is a model in which the largest term in each constraint is exactly 1. The purpose of the projection is to prevent extreme scaling. The default value of `rtmins` is 1 which implies that we do not scale the constraints up. Constraints with only small terms remain unchanged. The default value of `rtmaxs` is 2**30 or around 1.07e9 so terms much larger than 1.e9 are only partially scaled down and will still remain large.

3. The terms Abs(jac(i,j)*x(j)) after constraint scaling measure the importance of each variable in the particular constraint. The variable scale is selected so the largest importance of the variable over all constraints is 1. This gives a very simple variable scale factor, namely the absolute value of the variable. The variable is then divided by the variable scale factor. To avoid extreme scaling we again project on the interval `[rtmins, rtmaxs]`. Variables less than `rtmins` (default 1) are therefore not scaled up and variables over `rtmaxs` (default 2**30 = 1.07e9) are only partially scaled down.

You should note that CONOPT by default scales large numbers down, but it does not scale small numbers up. You should therefore try to avoid having variables or terms in expressions that are small but significant. If this is not possible, allow CONOPT to scale up by giving the option `rtmins` a value less than 1.

All scale factors are rounded down to a power of 2 to preserve precision in the internal computations. To avoid difficulties with rapidly varying variables and derivatives CONOPT recomputes the scale factors at regular intervals (see `lfscal`).

The options that control scaling, `lsscal`, `lfscal`, `rtmins`, and `rtmaxs`, are all described in Appendix B.

5.7.8.6 Finding a Feasible Solution: Phase 0

The GRG algorithm used by CONOPT is a feasible path algorithm. This means that once it has found a feasible point it tries to remain feasible and follow a path of improving feasible points until it reaches a local optimum. CONOPT starts with the point provided by GAMS. This point will always satisfy the bounds (3): GAMS will simply move a variable that is outside its bounds to the nearer bound before it is presented to the solver. If the general constraints (2) also are feasible then CONOPT will work with feasible solutions throughout the optimization. However, the initial point may not satisfy the general constraints (2). If this is not the case, CONOPT must first find an initial feasible point. This first step can be just as hard as finding an optimum for some models. For some models feasibility is the only problem.

CONOPT has two methods for finding an initial feasible point. The first method is not very reliable but it is fast when it works; the second method is reliable but slower. The fast method is called Phase 0 and it is described in this section. It is used first. The reliable method, called Phase 1 and 2, will be used if Phase 0 terminates without a feasible solution.

Phase 0 is based on the observation that Newton's method for solving a set of equations usually is very fast, but it may not always converge. Newton's method in its pure form is defined for a model with the same number of variables as equations, and no bounds on the variables. With our type of model there are usually too many variables, i.e. too many degrees of freedom, and there are bounds. To get around the problem of too many variables, CONOPT selects a subset with exactly m "basic" variables to be changed. The rest of the variables will remain fixed at their current values, that are not necessarily at bounds. To accommodate the bounds, CONOPT will try to select variables that are away from their bounds as basic, subject to the requirement that the Basis matrix, consisting of the corresponding columns in the Jacobian, must have full rank and be well conditioned.

The Newton equations are solved to yield a vector of proposed changes for the basic variables. If the full proposed step can be applied we can hope for the fast convergence of Newton's method. However, several things may go wrong:

1. The infeasibilities, measured by the 1-norm of g (i.e. the sum of the absolute infeasibilities, excluding the pre- and post-triangular equations), may not decrease as expected due to nonlinearities.

2. The maximum step length may have to be reduced if a basic variable otherwise would exceed one of its bounds.

In case 1. CONOPT tries various heuristics to find a more appropriate set of basic variables. If this does not work, some "difficult" equations, i.e. equations with large infeasibilities and significant nonlinearities, are temporarily removed from the model, and Newton's method is applied to the remaining set of "easy" equations.

In case 2. CONOPT will remove the basic variable that first reaches one of its bounds from the basis and replace it by one of the nonbasic variables. Newton's method is then applied to the new set of basic variables. The logic is very close to that of the dual simplex method. In cases where some of the basic variables are exactly at a bound CONOPT uses an anti degeneracy procedure based on Ryan and Osborne (1988) to prevent cycling.

Phase 0 will end when all equations except possibly some "difficult" equations are feasible within some small tolerance. If there are no difficult equations, CONOPT has found a feasible solution and it will proceed with Phase 3 and 4. Otherwise, Phase 1 and 2 is used to make the difficult equations feasible.

The iteration output will during Phase 0 have the following columns in the iteration log: Iter, Phase, Ninf, Infeasibility, Step, MX, and OK. The number in the Ninf column counts the number of "difficult" infeasible equations, and the number in the Infeasibility column shows the sum of the absolute infeasibilities in all the general constraints, both in the easy and in the difficult ones. There are three possible combinations of values in the MX and OK columns: combination (1) has F in the MX column and T in the OK column and it will always be combined with 1.0 in the Step column: this is an ideal Newton step. The infeasibilities in the easy equations should be reduced quickly, but the difficult equations may dominate the number in the Infeasibility column so you may not observe it. However, a few of these iterations is usually enough to terminate Phase 0. Combination (2) has T in the MX column indicating that a basic variable has reached its bound and is removed from the basis as in case 2. above. This will always be combined with T in the OK column. The Step column will show a step length less than the ideal Newton step of 1.0. Combination (3) has F in both the MX and OK column. It is the bad case and will always be combined with a step of 0.0: this is an iteration where nonlinearities are dominating and one of the heuristics from case 1. must be used.

The success of the Phase 0 procedure is based on being able to choose a good basis that will allow a full Newton step. It is therefore important that as many variables as possible have been assigned reasonable initial values so CONOPT has some variables away from their bounds to select from. This topic was discussed in more detail in section Initial Values .

The start and the iterations of Phase 0 can, in addition to the crash option described in section Finding a Feasible Solution: Phase 0 be controlled with the three options `lslack`, `lsmxbs`, and `lmmxsf` described in Appendix B.

5.7.8.7 Finding a Feasible Solution: Phase 1 and 2

Most of the equations will be feasible when phase 0 stops. To remove the remaining infeasibilities CONOPT uses a procedure similar to the phase 1 procedure used in Linear Programming: artificial variables are added to the infeasible equations (the equations with Large Residuals), and the sum of these artificial variables is minimized subject to the feasible constraints remaining feasible. The artificial variable are already part of the model as slack variables; their bounds are simply relaxed temporarily.

This infeasibility minimization problem is similar to the overall optimization problem: minimize an objective function subject to equality constraints and bounds on the variables. The feasibility problem is therefore solved with the ordinary GRG optimization procedure. As the artificial variables gradually become zero, i.e. as the infeasible equations become feasible, they are taken out of the auxiliary objective function. The number of infeasibilities (shown in the Ninf column of the log file) and the sum of infeasibilities (in the Infeasibility column) will therefore both decrease monotonically.

The iteration output will label these iterations as phase 1 and/or phase 2. The distinction between phases 1 (linear mode) and 2 (nonlinear mode) is similar to the distinction between phases 3 and 4, which are described in the next sections.

5.7.8.8 Linear and Nonlinear Mode: Phase 1 to 4

The optimization itself follows step 2 to 9 of the GRG algorithm shown in The CONOPT Algorithm above. The factorization in step 3 is performed using an efficient sparse LU factorization similar to the one described by Suhl and Suhl (1990). The matrix operations in step 4 and 5 are also performed sparse.

Step 7, selection of the search direction, has several variants, depending on how nonlinear the model is locally. When the model appears to be fairly linear in the area in which the optimization is performed, i.e. when the function and constraint values are close to their linear approximation for the steps that are taken, then CONOPT takes advantages of the linearity: The derivatives (the Jacobian) are not computed in every iteration, the basis factorization is updated using cheap LP techniques as described by Reid (1982), the search direction is determined without use of second order information, i.e. similar to a steepest descend algorithm, and the initial steplength is estimated as the step length where the first variable reaches a bound; very often, this is the only step length that has to be evaluated. These cheap almost linear iterations are referred to a Linear Mode and they are labeled Phase 1 when the model is infeasible and objective is the sum of infeasibilities and Phase 3 when the model is feasible and the real objective function is optimized.

When the constraints and/or the objective appear to be more nonlinear CONOPT will still follow step 2 to 9 of the GRG algorithm. However, the detailed content of each step is different. In step 2, the Jacobian must be recomputed in each iteration since the nonlinearities imply that the derivatives change. On the other hand, the set of basic variables will often be the same and CONOPT will take advantage of this during the factorization of the basis. In step 7 CONOPT uses the BFGS algorithm to estimate second order information and determine search directions. And in step 8 it will often be necessary to perform more than one step in the line search. These nonlinear iterations are labeled Phase 2 in the output if the solution is still infeasible, and Phase 4 if it is feasible. The iterations in phase 2 and 4 are in general more expensive than the iteration in phase 1 and 3.

Some models will remain in phase 1 (linear mode) until a feasible solution is found and then continue in phase 3 until the optimum is found, even if the model is truly nonlinear. However, most nonlinear models will have some iterations in phase 2 and/or 4 (nonlinear mode). Phase 2 and 4 indicates that the model has significant nonlinear terms around the current point: the objective or the constraints deviate significantly from a linear model for the steps that are taken. To improve the rate of convergence CONOPT tries to estimate second order information in the form of an estimated reduced Hessian using the BFGS formula.

Each iteration is, in addition to the step length shown in column "Step", characterized by two logicals: MX and OK. MX = T means that the step was maximal, i.e. it was determined by a variable reaching a bound. This is the expected value in Phase 1 and 3. MX = F means that no variable reached a bound and the optimal step length will in general be determined by nonlinearities. OK = T means that the line search was well-behaved and an optimal step length was found; OK = F means that the line search was ill-behaved, which means that CONOPT would like to take a larger step, but the feasibility restoring Newton process used during the line search did not converge for large step lengths. Iterations marked with OK = F (and therefore also with MX = F) will usually be expensive, while iterations marked with MX = T and OK = T will be cheap.

5.7.8.9 Linear Mode: The SLP Procedure

When the model continues to appear linear CONOPT will often take many small steps, each determined by a new variable reaching a bound. Although the line searches are fast in linear mode, each require one or more evaluations of the nonlinear constraints, and the overall cost may become high relative to the progress. In order to avoid the many nonlinear constraint evaluations CONOPT may replace the steepest descend direction in step 7 of the GRG algorithm with a sequential linear programming (SLP) technique to find a search direction that anticipates the bounds on all variables and therefore gives a larger expected change in objective in each line search. The search direction and the last basis from the SLP procedure are used in an ordinary GRG-type line search in which the solution is made feasible at each step. The

SLP procedure is only used to generate good directions; the usual feasibility preserving steps in CONOPT are maintained, so CONOPT is still a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SLP-mode are identified by numbers in the column labeled `InItr` in the iteration log. The number in the InItr column is the number of non- degenerate SLP iterations. This number is adjusted dynamically according to the success of the previous iterations and the perceived linearity of the model.

The SLP procedure generates a scaled search direction and the expected step length in the following line search is therefore 1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with OK = F and MX = F.
- A basic variable reaches a bound before predicted by the linear model. This is indicated with MX = T and OK = T.
- The objective is nonlinear along the search direction and the optimal step is less than one. This is indicated with OK = T and MX = F.

CONOPT will by default determine if it should use the SLP procedure or not, based on progress information. You may turn it off completely with the line `lseslp = f` in the CONOPT options file (usually *conopt.opt*). The default value of `lseslp` (`lseslp` = Logical Switch Enabling SLP mode) is `t` or `true`, i.e. the SLP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define `lseslp`, but it can be useful if CONOPT repeatedly turns SLP on and off, i.e. if you see a mixture of lines in the iteration log with and without numbers in the `InItr` column.

5.7.8.10 Linear Mode: The Steepest Edge Procedure

When optimizing in linear mode (Phase 1 or 3) CONOPT will by default use a steepest descend algorithm to determine the search direction. CONOPT allows you to use a Steepest Edge Algorithm as an alternative. The idea, borrowed from Linear Programming, is to scale the nonbasic variables according to the Euclidean norm of the "updated column" in a standard LP tableau, the so-called edge length. A unit step for a nonbasic variable will give rise to changes in the basic variables proportional to the edge length. A unit step for a nonbasic variable with a large edge length will therefore give large changes in the basic variables which has two adverse effects relative to a unit step for a nonbasic variable with a small edge length: a basic variable is more likely to reach a bound after a very short step length, and the large change in basic variables is more likely to give rise to larger nonlinear terms.

The steepest edge algorithm has been very successful for linear programs, and our initial experience has also shown that it will give fewer iterations for most nonlinear models. However, the cost of maintaining the edge lengths can be more expensive in the nonlinear case and it depends on the model whether steepest edge results in faster overall solution times or not. CONOPT uses the updating methods for the edge lengths from LP, but it must re-initialize the edge lengths more frequently, e.g. when an inversion fails, which happens more frequently in nonlinear models than in linear models, especially in models with many product terms, e.g. blending models, where the rank of the Jacobian can change from point to point.

Steepest edge is turned on with the line, `lsanrm = t`, in the CONOPT options file (usually *conopt.opt*). The default value of `lsanrm` (`lsanrm` = Logical Switch for A- Norm) is `f` or `false`, i.e. the steepest edge procedure is turned off.

The steepest edge procedure is mainly useful during linear mode iterations. However, it has some influence in phase 2 and 4 also: The estimated reduced Hessian in the BFGS method is initialized to a diagonal matrix with elements on the diagonal computed from the edge lengths, instead of the usual scaled unit matrix.

5.7.8.11 Nonlinear Mode: The SQP Procedure

When progress is determined by nonlinearities CONOPT needs second order information. Some second order information can be derived from the line search and is used in the first iterations in Phase 2 or 4. Depending on progress, CONOPT may switch to a Sequential Quadratic Programming (SQP) procedure that works on a sub-model with linear constraints and a quadratic objective function. The constraints are a linearization of the nonlinear constraints, and the objective function is derived from the Hessian of the Lagrangian function. CONOPT will inside the SQP procedure use exact second order information computed by GAMS. The result of the SQP procedure is a search direction and a basis and CONOPT will afterwards use the same line search procedure and feasibility preserving steps as after the SLP procedure. CONOPT remains a feasible path method with all its advantages, especially related to reliability.

Iterations in this so-called SQP-mode are identified by numbers in the column labeled "InItr" in the iteration log. The number in the InItr column is the number of non-degenerate SQP iterations. The effort spend inside the SQP procedure is adjusted dynamically according to the success of the previous iterations and the reduction in reduced gradient in the quadratic model.

The SQP procedure generates a scaled search direction and the expected step length in the following line search is therefore 1.0. The step length may be less than 1.0 for several reasons:

- The line search is ill-behaved. This is indicated with OK = F and MX = F.
- A basic variable reaches a bound before predicted by the linear model of the constraints. This is indicated with MX = T and OK = T.
- The objective is much more nonlinear along the search direction than expected and the optimal step is not one. This is indicated with OK = T and MX = F.

CONOPT will by default determine if it should use the SQP procedure or not, based on progress information. You may turn it off completely with the line `lsesqp = f` in the CONOPT options file (usually `conopt.opt`). The default value of `lsesqp` (lsesqp = Logical Switch Enabling SQP mode) is `t` or `true`, i.e. the SQP procedure is enabled and CONOPT may use it when considered appropriate. It is seldom necessary to define `lsesqp`, but it can be used for experimentation.

In connection with 1st and 2nd derivatives the listing file (∗.lst) will have a few extra lines. The first looks as follows:

```
The model has 537 variables and 457 constraints
with 1597 Jacobian elements, 380 of which are nonlinear.
The Hessian of the Lagrangian has 152 elements on the diagonal,
228 elements below the diagonal, and 304 nonlinear variables.
```

The first two lines repeat information given in the GAMS model statistics and the last two lines describe second order information. CONOPT uses the matrix of second derivatives (the Hessian) of a linear combination of the objective and the constraints (the Lagrangian). The Hessian is symmetric and the statistics show that it has 152 elements on the diagonal and 228 below for a total of 380 elements in this case. This compares favorably to the number of elements in the matrix of first derivatives (the Jacobian).

For some models you may see the following message instead (before the usual CONOPT banner):

```
** Warning **  Memory Limit for Hessians exceeded.
You can use the Conopt option "rvhess"
```

The creation of the matrix of second derivatives has been interrupted because the matrix became too dense. A dense matrix of second derivatives will be slow to compute and it will need a lot of memory. In addition, it is likely that a dense Hessian will make some of the computations inside the SQP iterations so slow that the potential saving in number of iterations is used up computing and manipulating the Hessian.

CONOPT can use second derivatives even if the Hessian is not available. A special version of the function evaluation routine can compute the Hessian multiplied by a vector (the so-called directional second derivative) without computing the Hessian itself. This routine is used when the Hessian is not available. The directional second derivative approach will require one directional second derivative evaluation call per inner SQP iteration instead of one Hessian evaluation per SQP sub-model.

If you get the "Memory Limit for Hessians exceeded" message you may consider rewriting some equation. Look for nonlinear functions applied to long linear or separable expressions such as log(sum(i,x(i)); as discussed in Section Simple Expressions . An expression like this will create a dense Hessian with card(i) rows and columns. You should consider introducing an intermediate variable that is equal to the long linear or separable expression and then apply the nonlinear function to this single variable. You may also experiment with allocating more memory for the dense Hessian and use it despite the higher cost; it may reduce the number of iterations. This can be done by adding the option `rvhess = XX` to the CONOPT options file. rvhess is a memory factor with default value 10 so you need a larger value. The value 0.0 is special; it means do not impose a memory limit on the Hessian.

The time spend on the many types of function and derivative evaluations are reported in the listing file in a section like this:

```
CONOPT time Total                            0.734 seconds
  of which: Function evaluations             0.031 =  4.3%
            1st Derivative evaluations       0.020 =  2.7%
            2nd Derivative evaluations       0.113 = 15.4%
            Directional 2nd Derivative       0.016 =  2.1%
```

The function evaluations are computations of the nonlinear terms in the model, and 1st Derivatives evaluations are computations of the Jacobian of the model. 2nd Derivative evaluations are computations of the Hessian of the Lagrangian, and Directional 2nd derivative evaluations are computations of the Hessian multiplied by a vector, computed without computing the Hessian itself. The lines for 2nd derivatives will only be present if CONOPT has used this type of 2nd derivative.

If your model is not likely to benefit from 2nd derivative information or if you know you will run out of memory anyway you can save a small setup cost by telling CONOPT not to generate it using option `dohess = f`.

5.7.8.12 How to Select Non-default Options

The non-default options have an influence on different phases of the optimization and you must therefore first observe whether most of the time is spend in Phase 0, Phase 1 and 3, or in Phase 2 and 4.

Phase 0: The quality of Phase 0 depends on the number of iterations and on the number and sum of infeasibilities after Phase 0. The iterations in Phase 0 are much faster than the other iterations, but the overall time spend in Phase 0 may still be rather large. If this is the case, or if the infeasibilities after Phase 0 are large you may try to use the triangular crash options:

```
lstcrs = t
```

Observe if the initial sum of infeasibility after iteration 1 has been reduced, and if the number of phase 0 iterations and the number of infeasibilities at the start of phase 1 have been reduced. If lstcrs reduces the initial sum of infeasibilities but the number of iterations still is large you may try:

```
lslack = t
```

CONOPT will after the preprocessor immediately add artificial variables to all infeasible constraints so Phase 0 will be eliminated, but the sum and number of infeasibilities at the start of Phase 1 will be larger. You are in reality trading Phase 0 iterations for Phase 1 iterations.

You may also try the experimental bending line search with

```
lmmxsf = 1
```

The line search in Phase 0 will with this option be different and the infeasibilities may be reduced faster than with the default `lmmxsf = 0`. It is likely to be better if the number of iterations with both MX = F and OK∼=∼F is large. This option may be combined with `lstcrs = t`. Usually, linear constraints that are feasible will remain feasible. However, you should note that with the bending linesearch linear feasible constraints could become infeasible.

Phase 1 and 3: The number of iterations in Phase 1 and Phase 3 will probably be reduced if you use steepest edge, `lsanrm = t`, but the overall time may increase. Steepest edge seems to be best for models with less than 5000 constraints, but work in progress tries to push this limit upwards. Try it when the number of iterations is very large, or when many iterations are poorly behaved identified with OK = F in the iteration log. The default SLP mode is usually an advantage, but it is too expensive for a few models. If you observe frequent changes between SLP mode and non-SLP mode, or if many line searches in the SLP iterations are ill-behaved with OK = F, then it may be better to turn SLP off with `lseslp = f`.

Phase 2 and 4: There are currently not many options available if most of the time is spend in Phase 2 and Phase 4. If the change in objective during the last iterations is very small, you may reduce computer time in return for a slightly worse objective by reducing the optimality tolerance, `rtredg`.

5.7.8.13 Miscellaneous Topics

Triangular Models

A triangular model is one in which the non-fixed variables and the equations can be sorted such that the first equation only depends on the first variable, the second equation only depends on the first two variables, and the p-th equation only depends on the first p variables. Provided there are no difficulties with bounds or small pivots, triangular models can be solved one equation at a time using the method describe in section Preprocessing: Pre-triangular Variables and Constraints and the solution process will be very fast and reliable.

Triangular models can in many cases be useful for finding a good initial feasible solution: Fix a subset of the variables so the remaining model is known to be triangular and solve this triangular simulation model. Then reset the bounds on the fixed variables to their original values and solve the original model. The first solve will be very fast and if the fixed variables have been fixed at good values then the solution will also be good. The second solve will start from the good feasible solution generated by the first solve and it will usually optimize much more quickly than from a poor start.

The modeler can instruct CONOPT that a model is supposed to be triangular with the option `lstria = t`. CONOPT will then use a special version of the preprocessing routine (see section Preprocessing: Pre-triangular Variables and Constraints) that solves the model very efficiently. If the model is solved successfully then CONOPT terminates with the message:

```
** Feasible solution to a recursive model.
```

and the Model Status will be 2, Locally Optimal, or 1, Optimal, depending on whether there were any nonlinear pivots or not. All marginals on both variables and equations are returned as 0 (zero) or EPS.

Two SOLVEs with different option files can be arranged by writing the option files as they are needed from within the GAMS program with PUT statements followed by a PutClose. You can also have two different option files, e.g., *conopt.opt* and *conopt.op2*, and select the second with the GAMS statement `<model>.OptFile = 2;`.

The triangular facility handles a number of error situations:

1. Non-triangular models: CONOPT will ensure that the model is indeed triangular. If it is not, CONOPT will return model status 5, Locally Infeasible, plus some information that allows the modeler to identify the mistake. The necessary information is related to the order of the variables and equations and number of occurrences of variables and equations, and since GAMS does no have a natural place for this type of information CONOPT returns it in the marginals of the equations and variables. The solution order for the triangular equations and variables that have been solved successfully are defined with positive numbers in the marginals of the equations and variables. For the remaining non- triangular variables and equations CONOPT shows the number of places they appear as negative numbers, i.e. a negative marginal for an equation shows how many of the non-triangular variables that appear in this equation. You must fix one or more variables until at least one of the non-triangular equation only has one non-fixed variable left.

2. Infeasibilities due to bounds: If some of the triangular equations cannot be solved with respect to their variable because the variable will exceed the bounds, then CONOPT will flag the equation as infeasible, keep the variable at the bound, and continue the triangular solve. The solution to the triangular model will therefore satisfy all bounds and almost all equations. The termination message will be

   ```
   ** Infeasible solution. xx artificial(s) have been
      introduced into the recursive equations.
   ```

   and the model status will be 5, Locally Infeasible.
   The modeler may in this case add explicit artificial variables with high costs to the infeasible constraints and the resulting point will be an initial feasible point to the overall optimization model. You will often from the mathematics of the model know that only some of the constraints can be infeasible, so you will only need to check whether to add artificials in these equations. Assume that a block of equations matbal(m,t) could become infeasible. Then the artificials that may be needed in this equation can be modeled and identified automatically with the following GAMS constructs:

   ```
   Set aposart(m,t) Add a positive artificial in Matbal
       anegart(m,t) Add a negative artificial in Matbal;
   aposart(m,t) = No; anegart(m,t) = No;

   Positive Variable
       vposart(m,t) Positive artificial variable in Matbal
       vnegart(m,t) Negative artificial variable in Matbal;

   matbal(m,t).. Left hand side =e= right hand side
      + vposart(m,t)$aposart(m,t) - vnegart(m,t)$anegart(m,t);

   objDef.. obj =e= other_terms +
            weight * Sum((m,t), vposart(m,t)$aposart(m,t)
                               +vnegart(m,t)$anegart(m,t) );

   Solve triangular model ...

   aposart(m,t)$(matbal.l(m,t) > matbal.Up(m,t)) = Yes;
   anegart(m,t)$(matbal.l(m,t) < matbal.Lo(m,t)) = Yes;

   Solve final model ...
   ```

3. Small pivots: The triangular facility requires the solution of each equation to be locally unique which also means that the pivots used to solve each equation must be nonzero. The model segment

```
e1 .. x1 =e= 0;
e2 .. x1 * x2 =e= 0;
```

will give the message

```
   x2 appearing in
   e2: Pivot too small for triangular model. Value=0.000E+00

** Infeasible solution. The equations were assumed to be
   recursive but they are not. A pivot element is too small.
```

However, the uniqueness of x2 may not be relevant if the solution just is going to be used as an initial point for a second model. The option `lsismp = t` (for Logical Switch: Ignore Small Pivots) will allow zero pivots as long as the corresponding equation is feasible for the given initial values.

Constrained Nonlinear System or Square Systems of Equations

There is a special model class in GAMS called CNS - Constrained Nonlinear System. A constrained nonlinear system is a square system of equations, i.e. a model in which the number of non-fixed variables is equal to the number of constraints. Currently, CONOPT and PATH are the only solvers with special treatment for this model class. A CNS model can be solved with a solve statement like

```
Solve <model> using CNS;
```

without an objective term. In some cases it may be convenient to solve a CNS model with a standard solve statement combined with an options file that has the statement `lssqrs = t`. In the latter case, CONOPT will check that the number of non-fixed variables is equal to the number of constraints. In either case, CONOPT will attempt to solve the constraints with respect to the non-fixed variables using Newton's method. The solution process does not include a lot of the safeguards used for ordinary NLP models and when it work it is often very fast and it uses less memory than for the corresponding NLP model. The lack of safeguards means that the solution process just will stop with an error message in some difficult situations and return the current intermediate infeasible solution. Examples of difficulties are that the Jacobian to be inverted is singular, or if one of the non-fixed variables tries to move outside their bounds as described with examples below.

Slacks in inequalities are counted as non-fixed variables which effectively means that inequalities should not be binding. Bounds on the variables are allowed, especially to prevent function evaluation errors for functions that only are defined for some arguments, but the bounds should not be binding in the final solution.

The solution returned to GAMS will in all cases have marginal values equal to 0 or EPS, both for the variables and the constraints.

The termination messages for CNS models are different from the termination messages for optimization models. The message you hope for is

```
** Feasible solution to a square system.
```

that usually will be combined with model status 16-Solved. If CONOPT in special cases can guarantee that the solution is unique, for example if the model is linear, then the model status will be 15-Solved Unique.

There are two potential error termination messages related to CNS models. A model with the following two constraints

```
e1 ..   x1 +   x2 =e= 1;
e2 .. 2*x1 + 2*x2 =e= 2;
```

will result in the message

```
** Error in Square System: Pivot too small.
    e2: Pivot too small.
    x1: Pivot too small.
```

"Pivot too small" means that the set of constraints is linearly dependent in the current point and there is no unique search direction for Newtons method so CONOPT terminates. The message points to one variable and one constraint. However, this just indicates that the linearly dependent set of constraints and variables include the constraint and variable mentioned. The offending constraint and variable will also be labeled 'DEPND' for linearly dependent in the equation listing. The error will usually be combined with model status 5 - Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities the model status will be 4 - Infeasible. If the constraints are linearly dependent but the current point satisfy the constraints then the solution status will be 17 - Solved Singular, indicating that the point is feasible, but there is probably a whole ray of feasible solution through the current point.

It should be mentioned that the linear dependency and small pivot could be caused by the initial point and that the model could have a solution. An example is

```
e1.. x1*x2 =E= 1;
e2.. x1+x2 =E= 3;
x1.l = 1; x2.l = 1;
```

A model with these two constraints and the bound

```
e1 .. x1 + x2 =e= 2;
e2 .. x1 - x2 =e= 0;
x1.lo = 1.5;
```

will result in the message

```
** Error in Square System: A variable tries to exceed its bound.
    x1: The variable tries to exceed its bound.
```

because the solution, (x1,x2) = (1,1) violates the bound on x1. This error case will also be combined with model status 5-Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities the model status will be 4 - Infeasible. If you encounter problems with active bounds but you think it is caused by nonlinearities and that there is a solution, then you may try to use the bending linesearch with option `lmmxsf = t`.

The CNS facility can be used to generate an initial feasible solution in almost the same way as the triangular model facility: Fix a subset of the variables so the remaining model is uniquely solvable, solve this model with the CNS solver or with `lssqrs = t`, reset the bounds on the fixed variables, and solve the original model. The CNS facility can be used on a larger class of models that include simultaneous sets of equations. However, the square system must be non-singular and feasible; CONOPT cannot, like in the triangular case, add artificial variables to some of the constraints and solve the remaining system when a variable reaches one of its bounds.

Additional information on CNS can be found in the GAMS User's Guide.

Loss of Feasibility

During the optimization you may sometimes see a phase 0 iteration and in rare cases you will see the message "Loss of Feasibility - Return to Phase 0". The background for this is as follows:

To work efficiently, CONOPT uses dynamic tolerances for feasibility and during the initial part of the optimization where the objective changes rapidly fairly large infeasibilities may be acceptable. As the change in objective in each iteration becomes smaller it will be necessary to solve the constraints more accurately so the "noise" in objective value from the inaccurate constraints will remain smaller than the real change. The noise is measured as the scalar product of the constraint residuals with the constraint marginals.

Sometimes it is necessary to revise the accuracy of the solution, for example because the algorithmic progress has slowed down or because the marginal of an inaccurate constraint has grown significantly after a basis change, e.g. when an inequality becomes binding. In these cases CONOPT will tighten the feasibility tolerance and perform one or more Newton iterations on the basic variables. This will usually be very quick and it happens silently. However, Newton's method may fail, for example in cases where the model is degenerate and Newton tries to move a basic variable outside a bound. In this case CONOPT uses some special iteration similar to those discussed in section Finding a Feasible Solution: Phase 0 and they are labeled Phase 0.

These Phase 0 iterations may not converge, for example if the degeneracy is significant, if the model is very nonlinear locally, if the model has many product terms involving variables at zero, or if the model is poorly scaled and some constraints contain very large terms. If the iterations do not converge, CONOPT will issue the "Loss of feasibility ..." message, return to the real Phase 0 procedure, find a feasible solution with the smaller tolerance, and resume the optimization.

In rare cases you will see that CONOPT cannot find a feasible solution after the tolerances have been reduced, even though it has declared the model feasible at an earlier stage. We are working on reducing this problem. Until a final solution has been implemented you are encouraged to (1) consider if bounds on some degenerate variables can be removed, (2) look at scaling of constraints with large terms, and (3) experiment with the two feasibility tolerances, `rtnwma` and `rtnwmi` (see Appendix B), if this happens with your model.

Stalling

CONOPT will usually make steady progress towards the final solution. A degeneracy breaking strategy and the monotonicity of the objective function in other iterations should ensure that CONOPT cannot cycle. Unfortunately, there are a few places in the code where the objective function may move in the wrong direction and CONOPT may in fact cycle or move very slowly.

The objective value used to compare two points, in the following called the adjusted objective value, is computed as the true objective plus a noise adjustment term equal to the scalar product of the residuals with the marginals (see section Loss of Feasibility where this noise term also is used). The noise adjustment term is very useful in allowing CONOPT to work smoothly with fairly inaccurate intermediate solutions. However, there is a disadvantage: the noise adjustment term can change even though the point itself does not change, namely when the marginals change in connection with a basis change. The adjusted objective is therefore not always monotone. When CONOPT looses feasibility and returns to Phase 0 there is an even larger chance of non-monotone behavior.

To avoid infinite loops and to allow the modeler to stop in cases with very slow progress CONOPT has an anti-stalling option. An iteration is counted as a stalled iteration if it is not degenerate and (1) the adjusted objective is worse than the best adjusted objective seen so far, or (2) the step length was zero without being degenerate (see OK = F in section Linear and Nonlinear Mode: Phase 1 to 4). CONOPT will stop if the number of consecutive stalled iterations (again not counting degenerate iterations) exceeds `lfstal` and `lfstal` is positive. The default value of `lfstal` is 100. The message will be:

```
** Feasible solution. The tolerances are minimal and
   there is no change in objective although the reduced
   gradient is greater than the tolerance.
```

Large models with very flat optima can sometimes be stopped prematurely due to stalling. If it is important to find a local optimum fairly accurately then you may have to increase the value of `lfstal`.

Overflow and NaN (Not A Number)

Very large values or overflow can appear in a number of places. CONOPT cannot use these values in the optimization since the results would be unreliable or useless. CONOPT is therefore trying to detect these values and take appropriate action.

Even though most modern computers can work with numbers from 1.e-300 to 1.e+300 CONOPT considers all numbers that are larger than 1.e40 or NaN to be 'bad' and useles. NaN means Not A Number and includes Infinity and Real Overflow.

Bad numbers can have two sources. They can come from the modeler expressions or they can be generated internally in CONOPT. The nonlinear constraints or the derivatives of these constraints may return very large values. If a constraint returns a bad number you will see a message like

```
** A function value is very large or NaN (Not a Number).
   Add bound to ensure that the function values are defined and bounded.
```

and if a derivatives returns a bad number the message will be something like

```
** A derivative is very large or NaN (Not a Number).
   Add bound to ensure that the derivatives are defined and bounded.
```

The modeler must in both cases adjust the model to prevent that CONOPT receives these bad numbers so CONOPT stops immediately.

Bad numbers can also in rare cases appear as a result of computations inside CONOPT. Since all primal variables and all derivatives are bounded the bad numbers can only appear as a result of operations involving the factorization of the basis matrix, including the dual variables. You will in this case see a message like

```
** Overflow or Nan (Not A Number) has been encountered indicating
   numerical difficulties. Trying to repair by increasing the
   absolute and relative pivot tolerances (Rtpiva and Rvpivr)
   and allowing small values to be scaled up by decreasing
   the minimum scale factor, Rtmins.
```

Since the source of the problem is the factorization of the basis CONOPT will adjust the tolerances that are used while computing this factorization. Essentially, we must avoid very small pivots and the pivoting tolerances are therefore increased. The source of very small pivots can also be constraints were all terms are very small so CONOPT will also change the minimum scaling factor to try to avoid this source. CONOPT will continue the optimization after changing the tolerances and if the problem appears again if will change the tolerances even more, but since there is a limit to how much they can be adjusted CONOPT may have to give up and it will happen with this message

```
** Overflow or Nan (Not A Number) continues to be encountered
   after multiple attempt to repair by changed tolerances.
   CONOPT will give up.
```

The termination message will in this case be

```
** Feasible solution. The solution process has been terminated
   because intermediate results have become NaN (Not A Number).
```

or

```
** Feasible solution. The solution process has been terminated
   because intermediate results have become NaN (Not A Number).
```

If you encounter messages with NaN it is always a problem with scaling or structure. The most likely source is constraints with only small terms. Try to avoid them by scaling the variables and constraints appropriately, or try to use option `rtmins`. Or make CONOPT ignore these constraints by using a larger absolute pivot tolerance, `rtpiva`.

Another source are structured models with long chains of relationships. These models will usually have lags or leads and neighboring variables are related with a factor different from one, i.e.

```
dyn(i).. x(i) =e= x(i-1)*1.5 + u(i);
```

or

```
dif(i).. 2*x(i) =e= x(i-1) + x(i+1) + u(i);
```

CONOPT will try to solve the constraints in the proper order to avoid exploding variables, but a larger relative pivot tolerance, `rtpivr`, may also help. If you have a dynamic model and the variables are supposed to grow rapidly over time then you may consider having a scale factor that also grows with time.

External Equations and Extrinsic Functions

CONOPT can be used with external equations and extrinsic functions written in a programming language such as Fortran or C. Additional information is available in the GAMS User's Guide External Equations and Extrinsic Functions.

Extrinsic functions can be debugged using the function suffixes `gradn` and `hessn` which use finite differences to approximate the gradient and Hessian using multiple function calls. For details check model **derivtst** in the GAMS Model Library. If external equations are present CONOPT will automatically turn the Function and Derivative Debugger on in the initial point to discover potential errors in the gradient calculation inside the external library. The debugger will not only check the gradients of the external library but even check the gradients calculated by GAMS. In rare cases the debugger may report problems in the regular algebra part of the model for which GAMS has calculated the gradients. These violates should be small and can be ignored. After verifying that the external part of the model has been programmed correctly you may turn debugging off again by setting `Lkdebg` to 0 in an options file.

The debugger has two types of check. The first type ensures that the external equations do not depend on other variables than the ones you have specified in the GAMS representation. Structural errors found by these check are usually caused by programming mistakes and must be corrected. The second type of check verifies that the derivatives returned by the external equations and extrinsic functions are consistent with the rate of change in function values. A derivative is considered to be wrong if the value returned by the modeler deviates from the value computed using numerical differences by more than `rtmxj2` times the step used for the numerical difference (usually around 1.e-7). This check is correct if second derivatives are less than `rtmxj2`. `rtmxj2` has a default value of 1.e4. If your model has larger second derivatives you may increase it in order not to get wrong error messages.

The number of error messages from the Function and Derivative Debugger is limited by `lfderr` with a default value of 10.

See Debugging options for a list of options that control the debugger.

5.7.9 APPENDIX B - Options

The options that ordinary GAMS users can access are listed below. Options starting on R assume real values, options starting on LS assume logical values (TRUE, T, 1, or FALSE, F, or 0), and all other CR-Cells starting on L assume integer values. The logical option `dohess` is only used by the interface between GAMS and CONOPT.

5.7.9.1 Algorithmic options

Option	Description	Default
LF2DRV	Limit on errors in Directional Second Derivative evaluation.	10
LFDEGI	Limit on number of degenerate iterations before starting degeneracy breaking strategy.	10
LFEERR	Limit on number of function evaluation errors. Overwrites GAMS Domlim option	GAMS DomLim
LFHSOK	Limit on errors in Hessian evaluation.	10
LFITER	Maximum number of iterations. Overwrites GAMS Iterlim option.	GAMS IterLim
LFMXNS	Maximum number of new superbasic variables added in one iteration.	auto
LFNICR	Limit on number of iterations with slow progress (relative less than Rtobjl).	20
LFNSUP	Maximum number of superbasic variables in the approximation to the Reduced Hessian.	auto
LFSCAL	Rescaling frequency.	5
LFSTAL	Limit on the number of stalled iterations.	100
LFUSDF	Limit on the number of candidates for defined variable in one constraint	2
LMDEBG	Method used by the function and derivative debugger.	0
LMMXSF	Method used to determine the step in Phase 0.	0
LMMXST	Method used to determine the maximum step while tightening tolerances.	0
LMNDIA	Method for initializing the diagonal of the approximate Reduced Hessian	0
LMSCAL	Method used for scaling.	3
LMUSDF	Method used with defined variables	0
LS2NDI	Flag for approximating Hessian information for incoming superbasics.	0
LS2PTJ	Flag for use of perturbations to compute 2nd derivatives in SQP method.	1
LSANRM	Flag for turning Steepest Edge on.	0
LSCRSH	Flag for crashing an initial basis without fixed slacks	1
LSESLP	Flag for enabling SLP mode.	1
LSESQP	Flag for enabling of SQP mode.	1
LSISMP	Flag for Ignoring Small Pivots in Triangular models	0
LSLACK	Flag for selecting initial basis as Crash-triangular variables plus slacks.	0
LSPOST	Pre-processor flag for identifying and using post-triangular equations.	1
LSPRET	Pre-processor flag for identifying and using pre-triangular equations.	1

Option	Description	Default
LSSCAL	Flag for dynamic scaling.	1
LSSQRS	Flag for Square System. Alternative to defining modeltype=CNS in GAMS	0
LSTCRS	Flag for using the triangular crash method.	0
LSTRIA	Flag for triangular or recursive system of equations.	0
LSTRID	Flag for turning diagnostics on for the post-triangular equations.	0
LSUQDF	Flag for requiring defined variables to be unique	1
LSUSDF	Flag for forcing defined variables into the basis	1
PRDEF	Flag for printing the defined variables and their defining constraints.	0
PRPOST	Flag for printing the post-triangular part of the model	0
PRPREC	Flag for printing the variables changed by the pre-processor	0
PRPRET	Flag for printing the pre-triangular part of the model	0
RTBND1	Bound filter tolerance for solution values close to a bound.	1.e-7
RTBNDT	Bound tolerance for defining variables as fixed.	1.e-7
RTIPVA	Absolute Pivot Tolerance for building initial basis.	1.e-7
RTIPVR	Relative Pivot Tolerance for building initial basis	1.e-3
RTMAXJ	Upper bound on the value of a function value or Jacobian element.	1.e10
RTMAXS	Upper bound on scale factors.	1.e9
RTMAXV	Upper bound on solution values and equation activity levels	1.e10
RTMINA	Zero filter for Jacobian elements and inversion results.	1.e-20
RTMINJ	Filter for small Jacobian elements to be ignored during scaling.	1.e-5
RTMINS	Lower bound for scale factors computed from values and 1st derivatives.	1
RTMNS2	Lower bound for scale factors based on large 2nd derivatives.	1.e-6
RTNWMA	Maximum feasibility tolerance (after scaling).	1.e-7
RTNWMI	Minimum feasibility tolerance (after scaling).	4.e-10
RTNWTR	Feasibility tolerance for triangular equations.	2.0e-8
RTOBJL	Limit for relative change in objective for well-behaved iterations.	3.0e-12
RTOBJR	Relative accuracy of the objective function.	3.0e-13
RTONED	Accuracy of One-dimensional search.	0.2
RTPIVA	Absolute pivot tolerance.	1.e-10
RTPIVR	Relative pivot tolerance during basis factorizations.	0.05
RTPIVT	Absolute pivot tolerance for nonlinear elements in pre-triangular equations.	1.e-5
RTPIVU	Relative pivot tolerance during basis updates.	0.05
RTPREC	Tolerance for printing variables changed by the pre-processor	0.0
RTPREL	Tolerance for defining large changes in variables in pre-processor	0.01

Option	Description	Default
RTREDG	Optimality tolerance for reduced gradient.	1.e-7
RVFILL	Fill in factor for basis factorization.	5
RVTIME	Time Limit. Overwrites the GAMS Reslim option.	GAMS ResLim

5.7.9.2 Debugging options

Option	Description	Default
LFDERR	Limit on number of error messages from function and derivative debugger.	10
LKDEBG	Flag for debugging of first derivatives	0
RTMXJ2	Upper bound on second order terms.	1.e4
RTZERN	Zero-Noise in external equations	0.0

5.7.9.3 Output options

Option	Description	Default
LFEMSG	Limit on number of error messages related to large function value and Jacobian elements.	10
LFILOG	Frequency for log-lines for non-SLP/SQP iterations.	auto
LFILOS	Frequency for log-lines for SLP or SQP iterations.	auto
LFTMSG	Limit on number of error messages related to infeasible pre-triangle.	25

5.7.9.4 Interface options

Option	Description	Default
cooptfile		
DO2DIR	Flag for computing and using directional 2nd derivatives.	auto
DOHESS	Flag for computing and using 2nd derivatives as Hessian of Lagrangian.	auto
heaplimit	Maximum Heap size in MB allowed	Infinite
pretri2log	Send messages about the pre-triangular analyser to the log	0
RVHESS	Memory factor for Hessian generation: Skip if #Hessian elements > #Jacobian elements∗Rvhess, 0 means unlimited.	10

cooptfile *(string)*: ↩

DO2DIR *(boolean)*: Flag for computing and using directional 2nd derivatives. ↩

If turned on, make directional second derivatives (Hessian matrix times directional vector) available to CONOPT. The default is on, but it will be turned off of the model has external

equations (defined with =X=) and the user has not provided directional second derivatives. If both the Hessian of the Lagrangian (see DOHESS) and directional second derivatives are available then CONOPT will use both: directional second derivatives are used when the expected number of iterations in the SQP sub-solver is low and the Hessian is used when the expected number of iterations is large.

Default: `auto`

DOHESS *(boolean)*: Flag for computing and using 2nd derivatives as Hessian of Lagrangian. ↩

If turned on, compute the structure of the Hessian of the Lagrangian and make it available to CONOPT. The default is usually on, but it will be turned off if the model has external equations (defined with =X=) or if the Hessian becomes too dense. See also DO2DIR and RVHESS.

Default: `auto`

heaplimit *(real)*: Maximum Heap size in MB allowed ↩

Default: `Infinite`

LF2DRV *(integer)*: Limit on errors in Directional Second Derivative evaluation. ↩

If the evaluation of Directional Second Derivatives (Hessian information in a particular direction) has failed more than Lf2drv times CONOPT will not attempt to evaluate them any more and will switch to methods that do not use Directional Second Derivatives. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: `10`

LFDEGI *(integer)*: Limit on number of degenerate iterations before starting degeneracy breaking strategy. ↩

The default CONOPT pivoting strategy has focus on numerical stability, but it can potentially cycle. When the number of consecutive degenerate iterations exceeds LFDEGI CONOPT will switch to a pivoting strategy that is guaranteed to break degeneracy but with slightly weaker numerical properties.

Default: `10`

LFDERR *(integer)*: Limit on number of error messages from function and derivative debugger. ↩

The function and derivative debugger (see LKDEBG) may find a very large number of errors, all derived from the same source. To avoid very large amounts of output CONOPT will stop the debugger after LFDERR error have been found.

Range: {`1`, ..., ∞}

Default: `10`

LFEERR *(integer)*: Limit on number of function evaluation errors. Overwrites GAMS Domlim option ↩

Synonym: domlim

Function values and their derivatives are assumed to be defined in all points that satisfy the bounds of the model. If the function value or a derivative is not defined in a point CONOPT will try to recover by going back to a previous safe point (if one exists), but it will not do it more than at most Lfeerr times. If CONOPT is stopped by functions or derivatives not being defined it will return with a intermediate infeasible or intermediate non-optimal model status.

Default: `GAMS DomLim`

LFEMSG *(integer)*: Limit on number of error messages related to large function value and Jacobian elements. ↩

Very large function value or derivatives (Jacobian elements) in a model will lead to numerical difficulties and most likely to inaccurate primal and/or dual solutions. CONOPT is therefore imposing an upper bound on the value of all function value and derivatives. This bound is 1.e30 for scaled models and RTMAXJ for unscaled models. If the bound is violated CONOPT will return with an intermediate infeasible or intermediate non-optimal solution and it will issue error messages for all the violating function value and Jacobian elements, up to a limit of Lfemsg error messages.

Range: {1, ..., ∞}

Default: `10`

LFHSOK *(integer)*: Limit on errors in Hessian evaluation. ↩

If the evaluation of Hessian information has failed more than Lfhsok times CONOPT will not attempt to evaluate it any more and will switch to methods that do not use the Hessian. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: `10`

LFILOG *(integer)*: Frequency for log-lines for non-SLP/SQP iterations. ↩

Lfilog and Lfilos can be used to control the amount of iteration send to the log file. The non-SLP/SQP iterations, i.e. iterations in phase 0, 1, and 3, are usually fast and writing a log line for each iteration may be too much, especially for smaller models. The default value for the log frequency for these iterations is therefore set to 10 for small models, 5 for models with more than 500 constraints or 1000 variables and 1 for models with more than 2000 constraints or 3000 variables.

Range: {1, ..., ∞}

Default: `auto`

LFILOS *(integer)*: Frequency for log-lines for SLP or SQP iterations. ↩

Lfilog and Lfilos can be used to control the amount of iteration send to the log file. Iterations using the SLP and/or SQP sub-solver, i.e. iterations in phase 2 and 4, may involve several inner iterations and the work per iteration is therefore larger than for the non-SLP/SQP iterations and it may be relevant to write log lines more frequently. The default value for the log frequency is therefore 5 for small models and 1 for models with more than 500 constraints or 1000 variables.

Range: {1, ..., ∞}

Default: `auto`

LFITER *(integer)*: Maximum number of iterations. Overwrites GAMS Iterlim option. ↩

Synonym: iterlim

The iteration limit can be used to prevent models from spending too many resources. You should note that the cost of the different types of CONOPT iterations (phase 0 to 4) can be very different so the time limit (GAMS Reslim or option RVTIME) is often a better stopping criterion. However, the iteration limit is better for reproducing solution behavior across machines.

Default: `GAMS IterLim`

LFMXNS *(integer)*: Maximum number of new superbasic variables added in one iteration. ↩

When there has been a sufficient reduction in the reduced gradient in one subspace new non-basics can be selected to enter the superbasis. The ones with largest reduced gradient of proper sign are selected, up to a limit. If Lfmxns is positive then the limit is min(500,Lfmxns). If Lfmxns is zero (the default) then the limit is selected dynamically by CONOPT depending on model characteristics.

Default: `auto`

LFNICR *(integer)*: Limit on number of iterations with slow progress (relative less than Rtobjl). ↩

The optimization is stopped if the relative change in objective is less than RTOBJL for Lfnicr consecutive well-behaved iterations.

Range: {2, ..., ∞}

Default: `20`

LFNSUP *(integer)*: Maximum number of superbasic variables in the approximation to the Reduced Hessian. ↩

CONOPT uses and stores a dense lower-triangular matrix as an approximation to the Reduced Hessian. The rows and columns correspond to the superbasic variable. This matrix can use a large amount of memory and computations involving the matrix can be time consuming so CONOPT imposes a limit on on the size. The limit is LFNSUP if LFNSUP is defined by the modeler and otherwise a value determined from the overall size of the model. If the number of superbasics exceeds the limit, CONOPT will switch to a method based on a combination of SQP and Conjugate Gradient iterations assuming some kind of second order information is available. If no second order information is available CONOPT will use a quasi-Newton method on a subset of the superbasic variables and rotate the subset as the reduced gradient becomes small.

Range: {5, ..., ∞}

Default: `auto`

LFSCAL *(integer)*: Rescaling frequency. ↩

The row and column scales are recalculated at least every Lfscal new point (degenerate iterations do not count), or more frequently if conditions require it.

Range: {1, ..., ∞}

Default: `5`

LFSTAL *(integer)*: Limit on the number of stalled iterations. ↩

An iteration is considered a stalled iteration it there is no change in objective because the linesearch is limited by nonlinearities or numerical difficulties. Stalled iterations will have Step = 0 and F in the OK column of the log file. After a stalled iteration CONOPT will usually try various heuristics to get a better basis and a better search direction. However, the heuristics may not work as intended or they may even return to the original bad basis, so to prevent cycling CONOPT stops after Lfstal stalled iterations and returns an Intermediate Infeasible or Intermediate Nonoptimal solution.

Range: {2, ..., ∞}

Default: `100`

LFTMSG *(integer)*: Limit on number of error messages related to infeasible pre-triangle. ↩

If the pre-processor determines that the model is infeasible it tries to define a minimal set of variables and constraints that define the infeasibility. If this set is larger than LFTMSG elements the report is considered difficult to use and it is skipped.

Default: `25`

LFUSDF *(integer)*: Limit on the number of candidates for defined variable in one constraint ↩

When there are more than one candidate to be selected as a defined variable in a constraint CONOPT tries to select the most appropriate in order to select as many defined variables as possible. However, to avoid too much arbitrariness this is only attempted if there are not more than Lfusdf candidates.

Default: `2`

LKDEBG *(integer)*: Flag for debugging of first derivatives ↩

Lkdebg controls how often the derivatives are tested. Debugging of derivatives is only relevant for user-written functions in external equations defined with =X=. The amount of debugging is controlled by LMDEBG. See RTMXJ2 for a definition of when derivatives are considered wrong.

Default: `0`

value	meaning
`-1`	The derivatives are tested in the initial point only.
`0`	No debugging
`+n`	The derivatives are tested in all iterations that can be divided by Lkdebg, provided the derivatives are computed in this iteration. (During phase 0, 1, and 3 derivatives are only computed when it appears to be necessary.)

LMDEBG *(integer)*: Method used by the function and derivative debugger. ↩

The function and derivative debugger (turned on with LKDEBG) can perform a fairly cheap test or a more extensive test, controlled by LMDEBG. See RTMXJ2 for a definition of when derivatives are considered wrong. All tests are performed in the current point found by the optimization algorithm.

Default: `0`

value	meaning
`0`	Perform tests for sparsity pattern and tests that the numerical values of the derivatives appear to be correct. This is the default.
`1`	As 0 plus make extensive test to determine if the functions and their derivatives are continuous around the current point. These tests are much more expensive and should only be used of the cheap test does not find an error but one is expected to exist.

LMMXSF *(integer)*: Method used to determine the step in Phase 0. ↩

The steplength used by the Newton process in phase 0 is computed by one of two alternative methods controlled by LMMXSF:

Default: `0`

value	meaning
0	The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.
1	A method based on bending (projecting the target values of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum. Linear constraints that initially are feasible may become infeasible due to bending. The method does not use anti-degeneracy. This will to be taken care off by removing difficult constraints from the Newton Process at regular intervals. The bending method can sometimes be useful for CNS models that stop when a variable exceeds its bound in an intermediate point even though the final solution is known to be inside the bounds.

LMMXST *(integer)*: Method used to determine the maximum step while tightening tolerances. ↩

The steplength used by the Newton process when tightening tolerances is computed by one of two alternative methods controlled by LMMXST:

Default: 0

value	meaning
0	The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.
1	A method based on bending (projecting the target value of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum.

LMNDIA *(integer)*: Method for initializing the diagonal of the approximate Reduced Hessian ↩

Each time a nonbasic variable is made superbasic a new row and column is added to the approximate Reduced Hessian. The off-diagonal elements are set to zero and the diagonal to a value controlled by LMNDIA:

Default: 0

value	meaning
0	The new diagonal element is set to the geometric mean of the existing diagonal elements. This gives the new diagonal element an intermediate value and new superbasic variables are therefore not given any special treatment. The initial steps should be of good size, but build-up of second order information in the new sub-space may be slower. The larger diagonal element may also in bad cases cause premature convergence.
1	The new diagonal elements is set to the minimum of the existing diagonal elements. This makes the new diagonal element small and the importance of the new superbasic variable will therefore be high. The initial steps can be rather small, but better second order information in the new sub-space should be build up faster.

LMSCAL *(integer)*: Method used for scaling. ↩

CONOPT will by default use scaling of the equations and variables of the model to improve the numerical behavior of the solution algorithm and the accuracy of the final solution, see

LSSCAL and LMSCAL. The objective of the scaling process is to reduce the values of all large primal and dual variables as well as the values of all large first derivatives so they become closer to 1. Small values are usually not scaled up, see RTMAXS and RTMINS. Scaling method 3 is recommended. The others are only kept for backward compatibility.

Default: `3`

value	meaning
`0`	Scaling is based on repeatedly dividing the rows and columns by the geometric means of the largest and smallest elements in each row and column. Very small elements less than RTMINJ are considered equal to RTMINJ.
`1`	Similar to 3 below, but the projection on the interval [Rtmins,Rtmaxs] is applied at a different stage. With method 1, abs(X)∗abs(Jac) with small X and very large Jac is scaled very aggressively with a factor abs(Jac). With method 3, the scale factor is abs(X)∗abs(Jac). The difference is seen in models with terms like Sqrt(X) close to X = 0.
`2`	As 1 but the terms are computed based on a moving average of the squares X and Jac. The purpose of the moving average is to keep the scale factor more stable. This is often an advantage, but for models with very large terms (large variables and in particular large derivatives) in the initial point the averaging process may not have enough time to bring the scale factors into the right region.
`3`	Rows are first scaled by dividing by the largest term in the row, then columns are scaled by dividing by by the maximum of the largest term and the value of the variable. A term is here defined as abs(X)∗abs(Jac) where X is the value of the variable and Jac is the value of the derivative (Jacobian element). The scale factors are then projected on the interval between Rtmins and Rtmaxs.

LMUSDF *(integer)*: Method used with defined variables ↩

When defined variables are identified (see LSUSDF) they can be used in two ways, controlled by LMUSDF:

Default: `0`

value	meaning
`0`	Defined variables are only used in the initial point and for the initial basis (default).
`1`	Defined variables are kept basic and the defining constraints are used to recursively assign values to the defined variables in all trial points.

LS2NDI *(boolean)*: Flag for approximating Hessian information for incoming superbasics. ↩

If Ls2ndi is turned on (1) CONOPT will try to estimate second order Hessian information for incoming superbasic variables based on directional second derivatives. This is more costly than the standard method described under LMNDIA.

Default: `0`

LS2PTJ *(boolean)*: Flag for use of perturbations to compute 2nd derivatives in SQP method. ↩

If on (1) CONOPT may use perturbations of the Jacobian to compute directional 2nd derivatives if they are not provided with other cheaper and more accorate methods. With GAMS it is only relevant for models with functions defined in external function libraries or models with external equations defined with the =X= equation type.

Default: `1`

LSANRM *(boolean)*: Flag for turning Steepest Edge on. ↩

Flag used to turn steepest edge pricing on (1) or off (0). Steepest edge pricing makes the individual iterations more expensive but it may decrease their number. Only experimentation can show if it is worth while.

Default: `0`

LSCRSH *(boolean)*: Flag for crashing an initial basis without fixed slacks ↩

When turned on (1) CONOPT will try to crash a basis without fixed slacks in the basis. Fixed slacks are only included in a last round to fill linearly dependent rows. When turned off, large infeasible slacks will be included in the initial basis with preference for variables and slacks far from bound.

Default: `1`

LSESLP *(boolean)*: Flag for enabling SLP mode. ↩

If Lseslp is on (the default) then the SLP (sequential linear programming) sub-solver can be used, otherwise it is turned off.

Default: `1`

LSESQP *(boolean)*: Flag for enabling of SQP mode. ↩

If Lsesqp is on (the default) then the SQP (sequential quadratic programming) sub-solver can be used, otherwise it is turned off.

Default: `1`

LSISMP *(boolean)*: Flag for Ignoring Small Pivots in Triangular models ↩

Ignore SMall Pivots. When turned on CONOPT will ignore the non-uniqueness from small pivots during a triangular solve (see LSTRIA). Note, that the non-uniqueness may propagate to later equations, but we cannot check for it in nonlinear equations.

Default: `0`

LSLACK *(boolean)*: Flag for selecting initial basis as Crash-triangular variables plus slacks. ↩

When turned on together with LSTCRS CONOPT will use the triangular crash procedure and select the initial basis as the crash-triangular variables plus slacks in all remaining rows.

Default: `0`

LSPOST *(boolean)*: Pre-processor flag for identifying and using post-triangular equations. ↩

When turned on (the default) CONOPT will try to identify post-triangular equations. Otherwise this phase is ignored.

Default: `1`

LSPRET *(boolean)*: Pre-processor flag for identifying and using pre-triangular equations. ↩

When turned on (the default) CONOPT will try to identify pre-triangular equations. Otherwise this phase is ignored.

Default: `1`

LSSCAL *(boolean)*: Flag for dynamic scaling. ↩

When Lsscal is on (the default) CONOPT will use dynamic scaling of equations and variables. See also LMSCAL.

Default: `1`

LSSQRS *(boolean)*: Flag for Square System. Alternative to defining modeltype=CNS in GAMS ↩

When turned on the modeler declares that this is a square system, i.e. the number of non-fixed variables must be equal to the number of constraints, no bounds must be active in the final solution, and the basis selected from the non-fixed variables must always be nonsingular.

Default: `0`

LSTCRS *(boolean)*: Flag for using the triangular crash method. ↩

When turned on CONOPT will try to crash a triangular basis using ideas by Gould and Reid. The procedure relies on identifying and using good initial values provided by the modeler and only assigning values to variables that are not initialized. Should only be used when several important variables have been given reasonable initial values. The sum of infeasibilities may for some models grow during the crash procedure, so modelers are advised that the option should be used with caution. The option will be ignored if defined variables are forced into the bases Lsusdf.

Default: `0`

LSTRIA *(boolean)*: Flag for triangular or recursive system of equations. ↩

If turned on the equations must form a recursive system. Equations that depend on known variables are allowed as long as they are consistent, e.g. x = 1 and 2∗x = 2. If the equations are not recursive the model is considered infeasible, and the equations with minimum row count are flagged together with the columns they intersect. See also LSISMP.

Default: `0`

LSTRID *(boolean)*: Flag for turning diagnostics on for the post-triangular equations. ↩

If turned on certain diagnostic messages related to the post-triangular equations will be printed. The messages are mainly related to unusual modeling constructs where linear variables for example only appear in the objective or where certain constraints are guarantied redundant.

Default: `0`

LSUQDF *(boolean)*: Flag for requiring defined variables to be unique ↩

When turned on (1) CONOPT will not allow defined variables unless they are unique. We exclude a variable if it can be defined from more than one equation, and we exclude equations if they can be used to define more than one variable.

Default: `1`

LSUSDF *(boolean)*: Flag for forcing defined variables into the basis ↩

When turned on (1) CONOPT will identify defined variables from constraints of the type x(i) = f(x) where x(i) is free or implied free. The largest number of defined variables possible will be made basic and will be assigned initial values that are consistent with their defining constraint. When turned off (0) defined variables and their defining constraints are treated like all other variables and constraints. When turned on the triangular crash (LSTCRS) will not be used

Default: `1`

PRDEF *(boolean)*: Flag for printing the defined variables and their defining constraints. ↩

When turned on (1) CONOPT will print a list of the defined variables and their defining constraints in the order in which they can be evaluated.

Default: `0`

pretri2log *(boolean)*: Send messages about the pre-triangular analyser to the log ↩

Default: `0`

PRPOST *(boolean)*: Flag for printing the post-triangular part of the model ↩

When turned on (1) CONOPT will print a list of the post-triangular constraints and the variables they are solved for in the order in which they can be evaluated.

Default: `0`

PRPREC *(boolean)*: Flag for printing the variables changed by the pre-processor ↩

When turned on (1) CONOPT will print a list of the variables that are changed by the pre-processor. The list includes pre-triangular variables, variables changed by implied bounds, and definitional variables. The three lists can for some models be very long and you can limit the list to variables that are changed by more than (RTPREC)

Default: `0`

PRPRET *(boolean)*: Flag for printing the pre-triangular part of the model ↩

When turned on (1) CONOPT will print a list of the pre-triangular constraints and the variables they are solved for, including the solution values, in the order in which they are solved.

Default: `0`

RTBND1 *(real)*: Bound filter tolerance for solution values close to a bound. ↩

A variable is considered to be at a bound if its distance from the bound is less than Rtbnd1 Max(1,ABS(Bound)). If you need a very small value then your model is probably poorly scaled.

Range: [`3.e-13`, `1.e-5`]

Default: `1.e-7`

RTBNDT *(real)*: Bound tolerance for defining variables as fixed. ↩

A variable is considered fixed if the distance between the bounds is less than Rtbndt $*$ Max(1,Abs(Bound)). The tolerance is also used on implied bounds (from converted inequalities) and these implied bounds may be infeasible up to Rtbndt.

Range: [`3.e-13`, `1.e-5`]

Default: `1.e-7`

RTIPVA *(real)*: Absolute Pivot Tolerance for building initial basis. ↩

Absolute pivot tolerance used during the search for a first logically non-singular basis. The default is fairly large to encourage a better conditioned initial basis.

Range: [`3.e-13`, `1.e-3`]

Default: `1.e-7`

RTIPVR *(real)*: Relative Pivot Tolerance for building initial basis ↩

Relative pivot tolerance used during the search for a first logically non-singular basis.

Range: [`1.e-4`, `0.9`]

Default: `1.e-3`

RTMAXJ *(real)*: Upper bound on the value of a function value or Jacobian element. ↩

Very large values of variables, function value, and derivatives and in particular large variations in the absolute value of the variables, functions, and derivatives makes the model harder to solve and poses problems for both feasibility and optimality tests. CONOPT will usually try to scale the model (see LSSCAL) to remove these problems. However, scaling can also make important aspects of a model appear un-important and there is therefore a limit to how aggressively we can scale a model (see RTMAXS and RTMINS). To avoid serious scaling problems CONOPT poses upper bounds on all variables (see RTMAXV) and all function value and derivatives, RTMAXJ.

Range: [`1.e4`, `1.e30`]

Default: `1.e10`

RTMAXS *(real)*: Upper bound on scale factors. ↩

Scale factors are projected on the interval from Rtmins to Rtmaxs. Is used to prevent very large or very small scale factors due to pathological types of constraints. RTMAXS is silently increased to max(RTMAXV,RTMAXS)/100 if RTMAXV or RTMAXJ have large non-default values.

Range: [`1`, `1.e20`]

Default: `1.e9`

RTMAXV *(real)*: Upper bound on solution values and equation activity levels ↩

See RTMAXJ for a discussion of why CONOPT poses upper bounds on variables and derivatives. If the value of a variable, including the objective function value, exceeds RTMAXV then the model is considered to be unbounded and the optimization process returns the solution with the large variable flagged as unbounded.

Range: [`1.e5`, `1.e30`]

Default: `1.e10`

RTMINA *(real)*: Zero filter for Jacobian elements and inversion results. ↩

Contains the smallest absolute value that an intermediate result can have. If it is smaller, it is set to zero. It must be smaller than RTPIVA/10.

Range: [`1.e-30`, ∞]

Default: `1.e-20`

RTMINJ *(real)*: Filter for small Jacobian elements to be ignored during scaling. ↩

A Jacobian element is considered insignificant if it is less than Rtminj. The value is used to select which small values are scaled up during scaling of the Jacobian.

Range: [`1.e-7`, `1.e-3`]

Default: `1.e-5`

RTMINS *(real)*: Lower bound for scale factors computed from values and 1st derivatives. ↩

Scale factors used to scale variables and equations are projected on the range Rtmins to Rtmaxs. The limits are used to prevent very large or very small scale factors due to pathological types of constraints. The default value for Rtmins is 1 which means that small values are not scaled up. If you need to scale small value up towards 1 then you must define a value of Rtmins < 1.

Range: [`1.e-10`, `1`]

Default: `1`

RTMNS2 *(real)*: Lower bound for scale factors based on large 2nd derivatives. ↩

Scaling of the model is in most cases based on the values of the variables and the first derivatives. However, if the scaled variables and derivatives are reasonable but there are large values in the Hessian of the Lagrangian (the matrix of 2nd derivatives) then the lower bound on the scale factor can be made smaller than Rtmins. CONOPT will try to scale variables with large 2nd derivatives by one over the square root of the diagonal elements of the Hessian. However, the revised scale factors cannot be less than Rtmns2.

Range: [`1.e-9`, `1`]

Default: `1.e-6`

RTMXJ2 *(real)*: Upper bound on second order terms. ↩

The function and derivative debugger (see LKDEBG) tests if derivatives computed using the modelers routine are sufficiently close to the values computed using finite differences. The term for the acceptable difference includes a second order term and uses RTMXJ2 as an upper bound on second order derivatives in the model. Larger RTMXJ2 values will allow larger deviations between the user-defined derivatives and the numerically computed derivatives.

Range: [`1`, ∞]

Default: `1.e4`

RTNWMA *(real)*: Maximum feasibility tolerance (after scaling). ↩

The feasibility tolerance used by CONOPT is dynamic. As long as we are far from the optimal solution and make large steps it is not necessary to compute intermediate solutions very accurately. When we approach the optimum and make smaller steps we need more accuracy. RTNWMA is the upper bound on the dynamic feasibility tolerance and RTNWMI is the lower bound.

Range: [`1.e-10`, `1.e-3`]

Default: `1.e-7`

RTNWMI *(real)*: Minimum feasibility tolerance (after scaling). ↩

See RTNWMA for a discussion of the dynamic feasibility tolerances used by CONOPT.

Range: [4.e-11, 1.e-5]

Default: 4.e-10

RTNWTR *(real)*: Feasibility tolerance for triangular equations. ↩

Triangular equations are usually solved to an accuracy of RTNWMI. However, if a variable reaches a bound or a constraint only has pre-determined variables then the feasibility tolerance can be relaxed to Rtnwtr.

Range: [3.e-13, 1.e-4]

Default: 2.0e-8

RTOBJL *(real)*: Limit for relative change in objective for well-behaved iterations. ↩

The change in objective in a well-behaved iteration is considered small and the iteration counts as slow progress if the change is less than Rtobjl $*$ Max(1,Abs(Objective)). See also LFNICR.

Range: [3.0e-13, 1.0e-5]

Default: 3.0e-12

RTOBJR *(real)*: Relative accuracy of the objective function. ↩

It is assumed that the objective function can be computed to an accuracy of Rtobjr $*$ max(1, abs(Objective)). Smaller changes in objective are considered to be caused by round-off errors.

Range: [3.0e-14, 10.e-6]

Default: 3.0e-13

RTONED *(real)*: Accuracy of One-dimensional search. ↩

The onedimensional search is stopped if the expected decrease in then objective estimated from a quadratic approximation is less than Rtoned times the decrease so far in this onedimensional search.

Range: [0.05, 0.8]

Default: 0.2

RTPIVA *(real)*: Absolute pivot tolerance. ↩

During LU-factorization of the basis matrix a pivot element is considered large enough if its absolute value is larger than Rtpiva. There is also a relative test, see RTPIVR.

Range: [2.2e-16, 1.e-7]

Default: 1.e-10

RTPIVR *(real)*: Relative pivot tolerance during basis factorizations. ↩

During LU-factorization of the basis matrix a pivot element is considered large enough relative to other elements in the column if its absolute value is at least Rtpivr $*$ the largest absolute value in the column. Small values or Rtpivr will often give a sparser basis factorization at the expense of the numerical accuracy. The value used internally is therefore adjusted dynamically between the users value and 0.9, based on various statistics collected during the solution process. Certain models derived from finite element approximations of partial differential equations can give rise to poor numerical accuracy and a larger user-value of Rtpivr may help.

Range: [`1.e-3`, `0.9`]

Default: `0.05`

RTPIVT *(real)*: Absolute pivot tolerance for nonlinear elements in pre-triangular equations. ↩

The smallest pivot that can be used for nonlinear or variable Jacobian elements during the pre-triangular solve. The pivot tolerance for linear or constant Jacobian elements is Rtpiva. The value cannot be less that Rtpiva.

Range: [`2.2e-16`, `1.e-3`]

Default: `1.e-5`

RTPIVU *(real)*: Relative pivot tolerance during basis updates. ↩

During basischanges CONOPT attempts to use cheap updates of the LU-factors of the basis. A pivot is considered large enough relative to the alternatives in the column if its absolute value is at least Rtpivu $*$ the other element. Smaller values of Rtpivu will allow sparser basis updates but may cause accumulation of larger numerical errors.

Range: [`1.e-3`, `0.9`]

Default: `0.05`

RTPREC *(real)*: Tolerance for printing variables changed by the pre-processor ↩

The list of variables changed by the pre-processor (see (PRPREC)) can be limited to the variables that are changed by more than RTPREC, defined as deltaX $>$ RTPREC $*$ max(1, abs(X)).

Range: [`0.0`, ∞]

Default: `0.0`

RTPREL *(real)*: Tolerance for defining large changes in variables in pre-processor ↩

The statistics for the number of variables that are changed in the pro-processor are classified as small and large changes. The change, DeltaX, is defined as large if deltaX $>$ RTPREL $*$ max(1, abs(X)).

Range: [`0.0`, ∞]

Default: `0.01`

RTREDG *(real)*: Optimality tolerance for reduced gradient. ↩

The reduced gradient is considered zero and the solution optimal if the largest superbasic component of the reduced gradient is less than Rtredg.

Range: [3.e-13, 1]

Default: 1.e-7

RTZERN *(real)*: Zero-Noise in external equations ↩

By default CONOPT will debug the derivatives returned from External functions in GAMS in the initial point (see LKDEBG). If external functions have constant derivatives then the constant terms are still part of the external function and this can give rise to small inaccuracies in the contribution of the constant derivatives to the function value. This noise can cause the debugger to incorrectly state that a the external equation depend on the variable with the constant derivative. A larger value of Rtzern should remove the error.

Range: [0.0, 1]

Default: 0.0

RVFILL *(real)*: Fill in factor for basis factorization. ↩

Rvfill is used in the initial allocation of memory for the factorization of the basis. The fill-in (number of new nonzeros) is assumed to be Rvfill-1 times the initial number of nonzeros in the basis. The default is 5 but you may experiment with a smaller value (down to 1.0) for models that use too much memory to get started. If Rvfill is small you may get slower execution due to increased memory movement. And you may still run out of memory later in the optimization.

Range: [1.00, 20]

Default: 5

RVHESS *(real)*: Memory factor for Hessian generation: Skip if #Hessian elements > #Jacobian elements∗Rvhess, 0 means unlimited. ↩

The Hessian of the Lagrangian is considered too dense and is not passed on to CONOPT if the number of nonzero elements in the Hessian of the Lagrangian is greater than the number of nonlinear Jacobian elements multiplied by Rvhess. The assumption is that a very dense Hessian is expensive both to compute and use. If Rvhess = 0.0 then there is no limit on the number of Hessian elements.

Default: 10

RVTIME *(real)*: Time Limit. Overwrites the GAMS Reslim option. ↩

Synonym: reslim

The upper bound on the total number of seconds that can be used in the execution phase. There are only tests for time limit once per iteration. The default value is 10000. Rvtime is overwritten by Reslim when called from GAMS. Rvtime is defined in ProbSize and/or UpdtSize when used as a subroutine library.

Default: GAMS ResLim

5.7.10 APPENDIX C: References

J. Abadie and J. Carpentier, Generalization of the Wolfe Reduced Gradient Method to the case of Nonlinear Constraints, in Optimization, R. Fletcher (ed.), Academic Press, New York, 37–47 (1969).

A. Drud, A GRG Code for Large Sparse Dynamic Nonlinear Optimization Problems, Mathematical Programming 31, 153–191 (1985).

A. S. Drud, CONOPT – A Large-Scale GRG Code, ORSA Journal on Computing 6, 207–216 (1992).

A. S. Drud, CONOPT: A System for Large Scale Nonlinear Optimization, Tutorial for CONOPT Subroutine Library, 16p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark (1995).

A. S. Drud, CONOPT: A System for Large Scale Nonlinear Optimization, Reference Manual for CONOPT Subroutine Library, 69p, ARKI Consulting and Development A/S, Bagsvaerd, Denmark (1996).

J. K. Reid, A Sparsity Exploiting Variant of Bartels-Golub Decomposition for Linear Programming Bases, Mathematical Programming 24, 55–69 (1982).

D. M. Ryan and M. R. Osborne, On the Solution of Highly Degenerate Linear Programmes, Mathematical Programming 41, 385–392 (1988).

U. H. Suhl and L. M. Suhl, Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases, ORSA Journal on Computing 2, 325–335 (1990).

5.8 CONOPT

Author

Arne Drud, ARKI Consulting and Development A/S, Bagsvaerd, Denmark

5.8.1 Introduction

CONOPT4 is an NLP solver derived from CONOPT3 but with many improvements. This documentation is intended to be self-contained, and there will therefore be some repetitions from the CONOPT3 documentation. We will refer to CONOPT4 as CONOPT in the following.

Nonlinear models created with GAMS must be solved with a nonlinear programming (NLP) algorithm. There are many solvers available, and the number is growing. It is almost impossible to predict how easy or difficult it is to solve a particular model with a particular algorithm, especially for NLP models, so GAMS cannot automatically select the best algorithm. The only reliable way to find which solver to use for a particular class of models is to experiment. However, it may be useful to classify different solvers and describe where CONOPT fits into this picture.

The most important distinction between NLP solvers is whether they attempt to find a local or a global solution. Solvers that attempt to find a global solution, called Global Solvers, will usually use some branch and bound technique and they can usually not solve very large models. As a contrast, most Local Solvers that only search for a locally optimal solution, can work with much larger models. If the model has the right mathematical properties, e.g., is convex, then Local Solvers will find a global optimum. Unfortunately, the theory for testing whether a general NLP model is convex is not very developed and is expected to be in the class of hard problems. CONOPT belongs to the class of Local Solvers, and it can be used for very large models.

Within the class of Local Solvers, the main distinction is between solvers based on Active Set methods and Interior Point methods. Active Set solvers are generally good for models with many equality constraints and few bounds. Interior Point solvers are generally good for models with many inequalities and many bounds, where the combinatorial nature of which inequalities and bounds are active play an important role. CONOPT is an Active Set solver.

Finally, within Active Set solvers we distinguish between Feasible Path methods and methods where feasibility only is found in the final point together with optimality. Feasible Path methods iterate through a sequence of feasible points, gradually improving the solution. It is assumed that the feasibility of the intermediate points will give more relevant derivatives and therefore better directions towards the optimal solution, and therefore will reduce the number of iterations relative to methods that do not enforce feasibility. On the other hand, each iteration will be more expensive because maintaining feasibility comes at a cost. CONOPT is a Feasible Path method, and great care has been taken to find an initial feasible solution and to reduce the cost of maintaining feasibility.

In addition to NLP models CONOPT has a good algorithm for CNS (Constrained Nonlinear Systems or Square Systems of equations) and CONOPT has been used to solve CNS models with many million variables and constraints.

This documentation will describe the basic algorithm and how the iteration output can help the user understand the solution progress. It will also describe some options that can be used to control the behavior of the algorithm. It should be emphasized that most users do not need any options. CONOPT has been designed to adjust its behavior based on statistics for the model and information about the progress of the algorithm.

5.8.2 The CONOPT Algorithm

The algorithm used in CONOPT is based on the GRG algorithm first suggested by Abadie and Carpentier (1969). The actual implementation has many modifications to make it efficient for large models. Here we will just give a short verbal description of the major steps in a generic GRG algorithm applied to the model:

```
Min or max f(x)
s.t. g(x) = 0
lb <= x <= ub
```

where x represents the set of variables, f represents the objective function, g represents a vector of constraints, and lb and ub represent vectors of lower and upper bounds. Inequalities are handled by adding properly bounded slack variables. The key steps in any GRG algorithm are:

1. Initialize and find a feasible solution.
2. Compute the Jacobian of the constraints, J.
3. Select a set of n basic variables, x_b, from x such that B, the sub-matrix of basic columns selected from J, is nonsingular. Factorize B. The remaining variables, x_n, are called nonbasic.
4. Solve $B^T u = df/dx_b$ for the multipliers u.
5. Compute the reduced gradient, $rgra = df/dx - J^T u$. `rgra` will be zero for the basic variables.
6. If rgra projected on the bounds is small, then stop. The current point is close to local optimality.
7. Select the set of superbasic variables, xs, as a subset of the nonbasic variables that have a reduced gradient pointing away from the bounds, and find a search direction, ds, for the superbasic variables based on $rgra$ and possibly on second order information.

8. Perform a line search along the direction ds. For each step, xs is changed in the direction ds and the basic variables x_b are subsequently adjusted to satisfy the constraints $g(x_b, x_s) = 0$ using an iterative Newton-like process with the factored B from step 3. When feasible the objective function $f(x)$ is evaluated and used to adjust the step and/or decide to terminate the search.

9. Go to 2.

The individual steps are of course much more detailed in the practical implementation in CONOPT. Step 1 consists of several pre-processing steps that usually create a smaller and easier model to be optimized. Step 1 also has a special Phase 0 and a scaling procedure as described in the following sections. The work involving the basis B, the LU factorization of B, and all operations involving B (step 3, 4, and 8) have been optimized to work even for very large models. The selection of search direction and optimizing step-lengths are specialized according to whether the model appears to be almost linear or not. For almost linear models some of the linear algebra work involving the matrices J and B is done using cheap LP-type updating techniques, second order information is not relevant in step 7, and the line search in step 8 has been improved by observing that the optimal step almost always will be determined by the first variable that reaches a bound. Similarly, when the model appears to be nonlinear, other aspects are optimized: the sets of basic and superbasic variables will often remain constant over several iterations, and the search direction is improved by using 2$^{\text{nd}}$ order information provided by GAMS. The choice of whether to use the almost linear or the nonlinear components is taken dynamically based on observations of the progress of the algorithm. Finally, the one-dimensional search and the feasibility restoring Newton iterations in step 8 have been optimized using quadratic inter- and extrapolations and they take advantage of linearity in the relevant constraints.

5.8.3 Iteration Output

When running CONOPT you will get a log-file that will look like this:

```
 CONOPT 4         44.2.0 7d8c2acc Aug 17, 2023          WEI x86 64bit/MS Window


    C O N O P T   version 4.31
    Copyright (C) ARKI Consulting and Development A/S
                  Bagsvaerdvej 246 A
                  DK-2880 Bagsvaerd, Denmark


    The user model has 610 constraints and 666 variables
    with 2432 Jacobian elements, 1518 of which are nonlinear.
    The Hessian of the Lagrangian has 306 elements on the diagonal,
    714 elements below the diagonal, and 562 nonlinear variables.

    Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
       0   0          1.1198119428E+02 (Input point)

    The pre-triangular part of the model has 4 constraints and 11 variables.
    The post-triangular part of the model has 2 constraints and variables.
    There are 425 definitional constraints and defined variables.

    Preprocessed model has 179 constraints and 228 variables
    with 1953 Jacobian elements, 1828 of which are nonlinear.

    Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                       9.8752676396E+01 (Full preprocessed model)
                       2.1093011202E-01 (After scaling)
                       2.0060649250E-01 (After adjusting individual variables)
```

```
    1   0          1.9323934856E-01                1.0E+00     1 T  T
    2   0          1.9273037524E-01                1.0E+00     1 T  T
    3   0          1.9272947599E-01                1.0E+00     1 T  T
    4   1       1  1.9121491986E-01 2.8E-02       24 6.0E-03    25 F  F
    5   1       1  1.8991312916E-01 7.9E-02        7 5.0E-03     7 F  F
    6   1       1  1.7954812670E-01 8.3E-02       50 7.9E-02     9 T  T
    7   1       1  1.6192877703E-01 1.0E-02       49 9.8E-01    10 F  F

 Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
    8   1       1  1.5263773874E-01 3.8E-01       15 1.8E-01    15 F  F
    9   1       1  9.4548821232E-02 3.9E-01       49 1.0E+00    22 T  T
   10   1       1  5.5194560481E-02 1.0E-01       40 8.6E-01    17 T  T
   11   1       1  3.8130110738E-02 1.2E-01       40 1.0E+00    16 T  T
   12   1       1  2.1971971978E-02 9.0E-02       29 1.0E+00    14 T  T
   13   1       1  6.3168158469E-05 3.7E-01        7 1.2E+00     8 T  T

** Feasible solution. Value of objective =    206.562941019

 Iter Phase   Ninf     Objective     RGmax      NSB   Step  InItr MX OK
   14   3          3.2026635060E+02 1.2E+02       18 2.4E-01    25 T  T
   15   3          4.0972213088E+02 2.1E+02       43 1.8E-01    25 F  F
   16   3          4.7700306262E+02 1.7E+02       44 1.9E-01    25 F  F
   17   3          4.9785498881E+02 6.9E+02       48 1.3E-01    19 F  F
   18   3          5.3401705259E+02 6.5E+02       48 1.0E+00    21 T  T
   19   3          5.5254890008E+02 8.0E+02       46 8.7E-01    17 T  T
   20   3          5.6360530732E+02 5.5E+02       48 1.0E+00     9 T  T
   21   3          5.7367560270E+02 6.5E+02       47 1.0E+00    11 T  T
   22   3          5.8820821496E+02 6.2E+02       47 3.1E+00     8 F  F
   23   3          5.9375570902E+02 7.6E+02       47 1.0E+00    15 T  T

 Iter Phase   Ninf     Objective     RGmax      NSB   Step  InItr MX OK
   24   3          6.2588529990E+02 7.6E+02       46 4.7E+00    14 T  T
   25   3          6.4664801061E+02 8.1E+02       47 3.7E+00    13 F  F
   26   3          6.6175989091E+02 7.4E+02       48 2.9E+00    16 F  T
   27   3          6.7789059174E+02 6.8E+02       48 9.3E+00    14 F  F
   28   3          6.9697407332E+02 7.1E+02       48 1.2E+01    19 T  T
   29   3          7.1098057393E+02 7.1E+02       48 1.4E+01    15 T  T
   30   4          8.6895316171E+02 1.9E+03       48 2.4E-01    18 F  T
   31   4          9.5309471994E+02 1.5E+03       49 8.1E-02    47 F  F
   32   4          1.0261710184E+03 2.1E+03       49 8.9E-02    46 F  F
   33   4          1.0583894431E+03 7.8E+02       49 4.2E-02    47 F  F

 Iter Phase   Ninf     Objective     RGmax      NSB   Step  InItr MX OK
   34   4          1.1225408637E+03 1.2E+04       49 8.8E-02    53 F  F
   35   4          1.1679673256E+03 1.3E+02       49 9.4E-02    47 F  F
   36   4          1.2151335208E+03 9.0E+02       49 1.4E-01    35 F  F
   37   4          1.2291455843E+03 7.3E+01       49 1.2E-01    24 F  F
   38   4          1.2392640422E+03 7.3E+01       49 1.2E-01    14 F  F
   39   4          1.2705894285E+03 7.9E+01       49 6.9E-01    17 F  F
   40   4          1.2785695198E+03 9.9E+01       49 6.0E-01    12 T  T
   41   4          1.2823556479E+03 1.2E+01       48 1.0E+00    12 T  T
   42   4          1.2823990320E+03 1.2E+00       43 1.0E+00    10 F  T
   43   4          1.2823997232E+03 4.8E-02       43 1.0E+00    13 F  T

 Iter Phase   Ninf     Objective     RGmax      NSB   Step  InItr MX OK
   44   4          1.2823997236E+03 9.2E-04       43 1.0E+00    11 F  T
   45   4          1.2823997236E+03 7.4E-06       43 1.0E+00     8 F  T
   46   4          1.2823997236E+03 1.9E-08       43
```

```
** Optimal solution. Reduced gradient less than tolerance.
```

The first few lines identify the version of CONOPT. CONOPT is updated at regular intervals, and if you want to report a problem, it is important to identify the version you are using.

The next lines show the size of the model measured in constraints, variables, and Jacobian elements; these numbers are also shown in the GAMS log. Usually, you will also see information about the Hessian (the matrix of 2nd derivatives) of the Lagrangian (a function combining the objective function and the constraints, $L = f(x) + u^T g(x)$ where u is a vector of dual variables). For some very large or dense models and models with few degrees of freedom, 2nd order information is not automatically available and these lines will be missing. You can force 2nd order information to be used with the option Flg_Hessian.

When CONOPT has received the model from GAMS it evaluates the constraint and determines the initial (sum of) Infeasibilities in the input point. It then executes a preprocessor that tries to reduce the size of the model and to find a solution that is closer to feasibility. The information from the preprocessor depends on the model and is described in the section Preprocessor.

After the preprocessors finishes, the model is scaled, and all later reporting of infeasibilities refers to the scaled model.

The remaining part of the log file describes the optimization iterations with the iteration number in the first column. For very small models you may not see all iterations in the iteration log. The options Frq_Log_SlpSqp and Frq_Log_Simple can be used to increase or decrease the log frequency.

The type of each iteration is described by the value of "Phase" in column 2. During Phase 0, 1, and 2 the model is infeasible and the Sum of Infeasibilities in column 4, labeled "Infeasibility", is being minimized. During Phase 1 and 2 the number in column "Ninf" tells how many constraints are still infeasible. These infeasibilities are minimized subject to the feasible constraints remaining feasible, and Ninf will therefore usually not increase. During Phase 3 and 4 the model is feasible and the actual objective function, also shown in column 4 and now labeled "Objective", is either minimized or maximized according to the Solve statement in GAMS.

Phase 0 iterations are Newton-like iterations. The background is described in the section Phase 0 - Finding an Initial Feasible Solution. During Phase 1 and 3 the model behaves almost linearly and CONOPT applies special linear iterations that take advantage of the linearity. These iterations are usually combined into several inner "Sequential Linear Programming" (SLP) iterations, and the number of these inner iterations are shown in the "InItr" column. During Phase 2 and 4 the model behaves more nonlinearly, either in the objective (Phase 4 only) or in the constraints, and most aspects of the iterations are therefore changed: the line search is more elaborate, and CONOPT needs second order information to improve the convergence. CONOPT uses inner "Sequential Quadratic Programming" (SQP) iterations based on exact second derivatives computed by GAMS. The number of these inner iterations are shown in the "InItr" column.

The algorithm in Phase 1 and 3 is almost the same; the only difference is in the set of constraints (in Phase 3 all constraints and in Phase 1 the feasible constraints) and in the objective function (in Phase 3 the user objective function and in Phase 1 the sum of the residuals in the infeasible constraints). The same applies to Phase 2 vs. Phase 4.

The SLP or SQP iterations will define a search direction on the tangent plane of the nonlinear constraints. CONOPT will follow this direction and use one or more steps to find the best solution in the direction. This process is called the one-dimensional search. Because CONOPT is a Feasible Path Method and the constraints in general are nonlinear, the points on the tangent plane are usually not feasible and CONOPT will therefore adjust some of the variables to make the constraints feasible, before the objective function is evaluated in a feasible point. The feasibility restoring process is based on an adaption of Newton's method where only a subset of the variables, the Basic variables, are adjusted.

The remaining numbers in the iteration log describes the one-dimensional search. The column "NSB" for Number of SuperBasics defines the degree of freedom or the dimension of the current search space,

and "Rgmax" measures the largest reduced gradient among the non-optimal variables. Rgmax should eventually converge to zero, but convergence will in general not be monotone. The last two columns labeled "MX" and "OK" give information about the one-dimensional search. OK = T means that the one-dimensional search was well-behaved, and the step described in column "Step" is close to the local optimum in the selected search direction. OK = F means that the one-dimensional search was stopped short of an optimal step, usually because the Newton iterations used to find a feasible solution only converge for small steps. MX = T means that the one-dimensional search was terminated because a variable reached a bound or an inequality became binding. This is always combined with OK = T. MX = F means that the step length was determined by the nonlinearity of the objective function.

The step suggested by the SLP or SQP procedure is always 1.0. The linear or quadratic approximation is therefore good when the optimal step is 1.0. Iterations with step 1.0 are often faster because it is sufficient to evaluate a single step. The last iteration in phase 1, iteration 11 and 12 in the log-file above, are very good and the last iterations before optimality, iteration 41 to 46 in phase 4, are also very good.

In the initial part of the optimization, when we are far from the final solution, the direction found by the SLP or SQP procedure will often suggest large changes in the variables, and the feasibility restoring Newton iterations may not converge due to large nonlinearities. We will therefore see more iteration with OK = F and Step < 1 in the beginning, e.g., iteration 15 to 17 in phase 3 and iteration 31 to 39 in phase 4.

5.8.4 Termination Messages

When CONOPT has finished it will show a termination message describing why it has finished. This section will show most of these messages followed by a short explanation. It will also show the Model Status returned to GAMS in <model>.ModelStat, where <model> represents the name of the GAMS model. The Solver Status returned in <model>.SolveStat will be given if it is different from 1 (Normal Completion). We will in all cases first show the message from CONOPT followed by a short explanation.

The first 4 messages are used for optimal solutions and CONOPT will return ModelStat = 2 (Locally Optimal), except as noted below:

```
** Optimal solution. There are no superbasic variables.
```

The solution is a locally optimal corner solution. The solution is determined by constraints only, and both the values of the variables and the value of the objective function are usually very accurate. In some cases, CONOPT can determine that the solution is globally optimal, and it will return ModelStat = 1 (Optimal).

```
** Optimal solution. Reduced gradient less than tolerance.
```

The solution is a locally optimal interior solution. The largest component of the reduced gradient is less than the optimality tolerance, Tol_Optimality, with default value $1.e^{-7}$. The value of the objective function is very accurate while the values of the variables can be less accurate due to a flat objective function in the interior of the feasible area.

```
** Optimal solution. The error on the optimal objective function
   value estimated from the reduced gradient and the estimated
   Hessian is less than the minimal tolerance on the objective.
```

The solution is a locally optimal interior solution. The largest component of the reduced gradient is larger than the optimality tolerance, Tol_Optimality. However, when the reduced gradient is scaled with second order information the solution seems optimal. For this to happen the objective must be large or the reduced objective must have large second derivatives, so it is advisable to scale the model if possible.

```
** Optimal solution. Convergence too slow. The change in
   objective has been less than xx.xx for xx consecutive
   iterations.
```

CONOPT stops with a solution that seems optimal. The solution process is stopped because of slow progress. The largest component of the reduced gradient is greater than the optimality tolerance, `Tol_Optimality`, but less than `Tol_Optimality` multiplied by the largest Jacobian element divided by 100. The model must have large derivatives, so it is advisable to scale it.

The four messages above all exist in versions where "Optimal" is replaced by "Infeasible" and ModelStat will be 5 (Locally Infeasible) or 4 (Infeasible). The infeasible messages indicate that the Sum of Infeasibility objective function in Phase 1 or 2 is locally minimal, but positive. If the model is convex, it does not have a feasible solution; if the model is non-convex it may have a feasible solution in a different region. See the section on Initial Values in the Tutorial and Examples section of the GAMS User's Guide for hints on what to do.

```
** Feasible solution. Convergence too slow. The change in
   objective has been less than xx.xx for xx consecutive
   iterations.

** Feasible solution. The tolerances are minimal and
   there is no change in objective although the reduced
   gradient is greater than the tolerance.
```

The two messages above tell that CONOPT stops with a feasible solution. In the first case the solution process is very slow and in the second there is no progress at all and the optimality criteria have not been satisfied. These messages are accompanied by ModelStat = 7 (Feasible Solution) and SolveStat = 4 (Terminated by Solver). The problem can be caused by discontinuities if the model is of type DNLP; in this case you should consider alternative, smooth formulations as discussed in section Reformulating DNLP Models in the Tutorials and Examples. The problem can also be caused by a poorly scaled model. See section Good NLP formulation and in particular the section on Scaling variables and Equations in Tutorials and Examples. Finally, it can be caused by stalling as described in section Stalling. The two messages also exist in a version where "Feasible" is replaced by "Infeasible". ModelStat is in this case 6 (Intermediate Infeasible) and SolveStat is still 4 (Terminated by Solver); these versions tell that CONOPT cannot make progress towards feasibility, but the Sum of Infeasibility objective function does not have a well-defined local minimum.

```
** Unbounded solution. A variable has reached 'infinity'.
   Largest legal value (Lim_Variable) is xx.xx
```

CONOPT considers a solution to be unbounded if a variable exceeds the indicated value of Lim_Variable (default $1.e^{15}$) and it returns ModelStat = 3 (Unbounded). The check for unboundedness is done at every iteration which means that CONOPT will stop if an intermediate solution has a variable that is very large, even if none of the variables in the optimal solution have large values. The variable that has reached 'Infinity' is shown in the listing file. You should check whether the solution appears unbounded, or the problem is caused by the scaling of the unbounded variable. If the model seems correct you are advised to scale it. There is also a lazy solution: you can increase the largest legal value, `Lim_Variable`, as mentioned in the section on options. However, you will most likely pay through reduced reliability or increased solution times. Unlike LP models, where an unbounded model is recognized by an unbounded ray and the iterations are stopped far from "infinity", CONOPT will make a line search and move to a region with large values of the variables. This may lead to bad scaling and derived tolerance and round off problems, including problems of determining whether a solution is feasible or not.

The message above exists in a version where "Unbounded" is replaced by "Infeasible" and ModelStat is 5 (Locally Infeasible). You may also see a message like

```
** Infeasible solution. A free variable exceeds the allowable
   range.  Current value is xx.xx and current upper bound
   (Lim_Variable) is xx.xx
```

These Infeasible messages indicate that some variables become very large before a feasible solution has been found. You should again check whether the problem is caused by the scaling of the unbounded variable. If the model seems correct you should scale it.

```
** The time limit has been reached.
```

The time or resource limit defined in GAMS, either by default (usually 1000 seconds) or by `Option ResLim = xx;` or `<model>.ResLim = xx;` statements, has been reached. CONOPT will return with SolveStat = 3 (Resource Interrupt) and ModelStat either 6 (Locally Infeasible) or 7 (Feasible Solution).

```
** The iteration limit has been reached.
```

The iteration limit defined in GAMS, either by default (usually 2000000000 iterations) or by `Option IterLim = xx;` or `<model>.IterLim = xx;` statements, has been reached. CONOPT will return with SolveStat = 2 (Iteration Interrupt) and ModelStat either 6 (Locally Infeasible) or 7 (Feasible Solution).

```
** Domain error(s) in nonlinear functions.              -or-
** Domain error(s) in nonlinear functions. NaN returned -or-
** Domain error(s) in nonlinear functions. Residual too large.
   Check bounds on variables.
```

The number of function evaluation errors or bad function values has reached the limit defined in GAMS by `Option DomLim = xx;` or `<model>.DomLim = xx;` statements or the default limit of 0 function evaluation errors. CONOPT will return with SolveStat = 5 (Evaluation Error Limit) and ModelStat either 6 (Locally Infeasible) or 7 (Feasible Solution).

Many of the nonlinear functions available with GAMS are not defined for all values of their arguments. Log is not defined for negative arguments, Exp overflows for large arguments, and division by zero is illegal. To avoid evaluating functions outside their domain of definition you should add reasonable variable bounds. CONOPT will in return guarantee that the nonlinear functions never are evaluated with variables outside their bounds. For more advice, see details in section Good NLP Formulations in the Tutorials and Examples section of the GAMS User's Guide.

```
** An initial derivative is too large (larger than xx.xx)
   Scale the variables and/or equations or add bounds.

   <var> appearing in
   <equ>: Initial Jacobian element too large = xx.xx
```

and

```
** A derivative is too large (larger than xx.xx).
   Scale the variables and/or equations or add bounds.

   <var> appearing in
   <equ>: Jacobian element too large = xx.xx
```

These two messages appear if a derivative or Jacobian element is very large, either in the initial point or in a later intermediate point. The relevant variable and equation pair(s) will be shown in the listing file and will guide you where to look. A large derivative means that the function changes very rapidly even after a very small change in the variable and it will most likely create numerical problems for many parts of the optimization algorithm. Instead of attempting to solve a model that most likely will fail, CONOPT will stop, and you are advised to adjust the model.

If the offending derivative is associated with a `Log(x)` or `1/x` term you may try to increase the lower bound on `x`. If the offending derivative is associated with an `Exp(x)` term you must decrease the upper bound on `x`. You may also try to scale the model, either manually or using the variable.Scale and/or equation.Scale option in GAMS as described in the section Scaling variables and Equations in the Tutorials and Examples part of the GAMS documentation.

5.8.5 Preprocessor

The preprocessor in CONOPT3 identifies pre- and post-triangular variables and constraints, and it handles these variables and constraints in a special way to make some internal routines run more efficiently.

CONOPT goes one step further and distinguishes between a 'user model' as defined by the user via the GAMS language, and an 'internal model'. Pre-triangular variables and constraints are simply removed from the user model and are not present in the internal model. Post-triangular variables and constraints are collapsed into a single condensed objective function. And definitional constraints are eliminated. After the internal model has been solved, CONOPT translates the internal solution back into the solution for the user model and reports this solution to the user.

In addition to the simple pre- and post-triangular variables and constraints, the preprocessor in CONOPT looks at more possibilities for simplifying the model. Some of the new features are:

- Fixed variables are removed completely.
- Constraints that represent simple inequalities are identified and changed into simple bounds on the variables and the constraints are removed.
- Simple monotone constraints such as `exp(x) =L= c1 or log(y) =L= c2` are converted into simple bounds on the variables and then removed.
- Forcing constraints such as `x1 + x2 =L= 0` with `x1.lo = 0` and `x2.lo = 0` are identified, the variables are fixed, and the constraints are removed.
- Redundant constraints such as `x1 + x2 =L= 3` with `x1.lo = 0`, `x1.up = 1`, `x2.lo = 0`, and `x2.up = 1` are identified and removed.
- Linear and monotone constraints are used to compute 'implied bounds' on many variables and these bounds can help CONOPT get a better starting point for finding an initial feasible solution. The implied bounds may also change a non-monotone constraint into a monotone constraint, and they may help identify redundant constraints.
- Some non-monotone constraints such as `sqr(x1) + sqr(x2) =L= 1` can also be used to derive implied bounds (here `-1` $<$ `x1` $<$ `+1 and -1` $<$ `x2` $<$ `+1`) that both can improve the starting point and can be used to determine that other terms are monotone or redundant.
- Constraints with exactly two variables, e.g., simple linear identities such as `x1 =E= a*x2 + b` or simple monotone identities such as `x3 =E= exp(x4)`, are used to move bounds between the two variables and this may result in more variables being included in the post-triangle.
- Linear constraints that are identical or proportional to others are identified and removed.
- Pairs of constraints that define a lower and an upper bound on the same linear expression or proportional linear expressions, e.g., `1 =L= x1 + x2` and `2*x1+2*x2 =L= 4`, are turned into a single ranged constraint with a double-bounded slack variable.

- Nonlinear constraints that become linear when the pre-triangular variables are fixed are recognized as being linear with the resulting simplifications.

Some of the new preprocessing steps are useful when solving sub-models in a Branch and Bound environment. A constraint like `x =L= M*y` where `y` is a binary variable fixed at either `0` or `1` is turned into a simple bound on `x`. And a constraint like `sum(i, x(i) ) =L= Cap*y (with x.lo(i) = 0)` combined with `y` fixed at zero will force all `x(i)` to zero.

The preprocessor also identifies constructs that are easy to make feasible. There are currently two types:

- Penalty terms: A penalty constraint is defined as a constraint of the form `f(x1,x2,..) + p - n =E= 0`, where `p` and `n` are positive variables, and where `p` and `n` only appear in post-triangular constraints or in previously identified penalty constraint. For any feasible values of the x-variables it is easy to find values of `p` and `n` that make the penalty constraint feasible: `p = max(0,-f(x))` and `n = max(0,f(x))`. The definition is easily generalized to constraints where p and n have coefficients different from one and nonzero bounds; the essence is the presence of two linear unbounded terms of opposite sign.
- Minimax terms: A minimax group is defined as a group of constraints of the form `eq(i).. fi(x1,x2,..) =L= z` where `z` is common to the group and otherwise only appears in post-triangular constraints, and `z` is unbounded from above. For any feasible value of the x-variables it is easy to find a value of z that makes the minimax group feasible: `z = smin(i: fi(x))`. The definition is easily generalized to groups of constraints where z has coefficients different from one and where the direction of the inequality is reversed.

The preprocessor will also recognize definitional constraints: constraints of the form `x =E= f(y)`, where `x` is a free variable or the bounds on `x` cannot be binding, are called definitional constraints and `x` is called a defined variable. If there are many potential defined variables the preprocessor will select a recursive set and logically eliminate them from the internal model: The values of the defined variables are easily derived from the values of all other variables by solving the definitional constraints in their recursive order. These values are then substituted into the remaining constraints before their residuals are computed. The matrix of derivatives of the remaining constraints is computed from the overall matrix of derivatives via an efficient elimination of the triangular definitional constraints.

The following extract from the log-file for the **otpop.gms** model in the GAMS Library shows the main output from the preprocessor:

```
The user model has 77 constraints and 104 variables
with 285 Jacobian elements, 100 of which are nonlinear.
The Hessian of the Lagrangian has 17 elements on the diagonal,
33 elements below the diagonal, and 66 nonlinear variables.

Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
0   0          4.0939901439E+03 (Input point)

The pre-triangular part of the model has 32 constraints and 43 variables.
The post-triangular part of the model has 9 constraints and variables.
There are 13 definitional constraints and defined variables.

Preprocessed model has 23 constraints and 39 variables
with 88 Jacobian elements, 25 of which are nonlinear.

Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                  1.0510968588E+03 (Full preprocessed model)
                  4.5063072017E+01 (After scaling)
```

The first few lines define the size of the user model. After the description of the initial sum of infeasibilities there are three lines with statistics from the preprocessor. Note that the pre-triangular part of the model has more variables than constraints, in this case because there are fixed variables. The post-triangular part will always have the same number of constraints as variables, and the same is true for definitional constraints and defined variables.

After the statistics from the preprocessor the size of the resulting internal model is shown. The number of constraints in the preprocessed model, here 23, is computed from 77 constraints in the user model minus 32 removed in the pre-triangle, 9 removed in the post-triangle, and 13 removed as definitional constraints. Similarly, the number of variables in the preprocessed model, here 39, is computed from 104 variables in the user model minus 43 removed in the pre-triangle, 9 removed in the post-triangle, and 13 removed as definitional variables. The Jacobian counts cannot be derived easily. And there is no Hessian information for the internal model; it is costly to compute the Hessian for the internal model and it will in most cases be very dense so all use of 2^{nd} order information in the internal model is computed by mapping variables and constraints to the user model and using the Hessian in the user model.

Other examples of output from the preprocessor are explained in the section on Alternative Sub-Models.

There are some options that control the preprocessor:

- You can turn the preprocessor off using the option Flg_Prep = false. Note that CONOPT still will generate an internal model, but it will be very close to the user model.
- You can turn the search for definitional constraints off using option Flg_NoDefc = true. The time used for the search will usually be recovered later by cheaper iterations, except for models that solve in very few iterations.

5.8.6 Adjust Initial Point

If the preprocessed model is infeasible, CONOPT starts with a new 'Adjust Initial Point' procedure. The procedure reduces the sum of infeasibilities by changing individual variables one at a time. The procedure is very cheap because changing a single variable only involves a small part of the overall model. The procedure will as a by-product produce a large part of a good initial basis and many constraints will become feasible. If the Adjust Initial Point procedure reduces the infeasibilities, the log file will have a line as shown below (from otpop.gms):

```
Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                    1.0510968588E+03 (Full preprocessed model)
                    4.5063072017E+01 (After scaling)
                    2.7782397440E+00 (After adjusting individual variables)
  5   0             1.5284282657E+00                 1.0E+00     1 T  T
```

As this example shows, the procedure can in some cases reduce the sum of infeasibilities significantly. In rarer cases it will find a feasible solution.

The time used by the Adjust Initial Point procedure can in rare cases, usually for very dense models, be large. If you experience this, you can turn the procedure off with option Flg_AdjIniP = false.

5.8.7 Phase 0 - Finding an Initial Feasible Solution

CONOPT is a feasible path method, but the initial point defined by the user will not always be feasible, and the Adjust Initial Point procedure may not reduce the infeasibilities to zero.

Phase 0 searches for a feasible solution using a procedure based on Newton's method for solving nonlinear equalities. If a basis has been provided by GAMS (see the GAMS Option Bratio for details) then the initial basis will be as chosen from the GAMS basis with adjustments to make it non-singular. Otherwise, a set of basic variables is selected with preference for variables away from their bounds. It may not be possible to find a full basis with variables away from their bounds so the initial basis may also have some slack variables. The standard Newton's method defines a change direction for these basic variables but bounds and basic slacks will sometimes prevent this direction from reducing the sum of infeasibilities. CONOPT is therefore using an LP-like pricing step to select a subset of the constraints for with Newton's method will reduce the sum of infeasibilities. Newton's method applied to this subset gives a search direction and if a full step can be taken without reaching any bounds the constraints in the subset will usually become feasible rather quickly. If bounds become active the basis is changed and the process is repeated. The constraints that are not in the Newton subset, usually few, will subsequently be made feasible using a phase 1 procedure.

This log file extract showing Phase 0 is from GAMS Library model mathopt3.gms:

```
 Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                    1.7198687345E+03 (Full preprocessed model)
                    7.9643177481E+00 (After scaling)
                    1.0113241089E+00 (After adjusting individual variables)
    1   0          8.6918201909E-01                  2.5E-01     2 F  F
    2   0          4.1584560783E-01                  1.0E+00     1 T  T
    3  S0          9.1316232150E-01                  1.0E+00     1 T  T
    4   0          8.4915160660E-01                  2.5E-01     1 F  F
    5  S0          6.2619863264E-01                  1.0E+00     1 T  T
    6   0          6.1975571905E-01                  1.0E+00     1 T  T
    7   0          6.1974675345E-01                  1.0E+00     1 T  T
    8   1       2  1.5206900053E-01 1.0E+00       2 7.5E-01     2 F  F
```

The `InItr` column shows the number of basis changes for each outer iteration and `Step = 1` indicates that the full solution from Newton's method was used. For some of the iterations Step is less than 1 indicating that the direction found by the inner linear model could not be used in full due to nonlinearities. There are some lines with 'S0' instead of 0 in the Phase column. The S tells that the model was scaled before the iteration and the sum of infeasibilities was increased during this scaling procedure. The sum of infeasibilities is therefore not monotone decreasing, even if each outer iteration does decrease them.

Phase 0 terminates when the Newton subset is made feasible and CONOPT switches to Phase 1 where the remaining constraints are handled. In this model there are just 2 infeasible constraints left.

5.8.8 Transition between SLP and SQP

The transition from SLP (Phase 1 or 3) to SQP (Phase 2 or 4) and back again is in CONOPT3 based on monitoring failure. This logic has been changed in CONOPT, so transition is based on continuous measurements of curvature, both in the general constraints and in the objective function, combined with estimates of computational costs and progress for SLP and SQP.

The continuation of the log file for GAMS Library model otpop.gms shows some of this:

```
 Iter Phase   Ninf     Objective     RGmax      NSB   Step  InItr MX OK
    6   3          2.3861517936E+02 3.6E+02      12 2.1E-01     7 F  T
    7   3          1.4308470370E+02 1.1E+02      16 2.0E-01     9 F  T
    8   3          9.4375606705E+01 1.5E+02      16 1.8E-01     5 F  F
    9   3          2.4652509772E+01 7.4E+01      16 6.7E-01     2 F  T
   10   4          2.4445151316E-02 3.3E+01      16 1.0E+00     6 F  T
   11   4          5.0735392100E-06 4.4E+00      16 1.0E+00     5 F  T
   12   4          1.0276261682E-09 1.9E-02      16 1.0E+00     3 F  T
   13   4          1.4326828955E-13 2.8E-06      16 1.0E+00     1 F  T
   14   4          1.4326828955E-13 4.0E-10      16

** Optimal solution. Reduced gradient less than tolerance.
```

Iterations 6 to 9 are SLP iterations (Phase 3) and iterations 10 to 14 are SQP iterations (Phase 4). The SLP iterations have Step less than 1 due to nonlinear objective terms and CONOPT jumps directly from SLP to SQP. You may in other models see that SQP is replaced by SLP after some iterations where the nonlinearities are measured to be very small.

5.8.9 Bad Iterations

Bad iterations, flagged with "F" in the "OK" column can be a problem. They appear if the output from the SLP or SQP procedure is a search direction where CONOPT cannot move very far because it is difficult to make the nonlinear constraints feasible again. The efforts spent in solving the SLP or SQP procedure is therefore partially wasted. The problem is usually associated with a basis that is ill-conditioned or has Jacobian elements that change very fast.

CONOPT monitors the quality of the basis using information that already has been computed such as the size of elements of the tangent for basic variables relative to similar elements for superbasic variables and intermediate results from the computation of the reduced costs. This information is then used to trigger the search for a better basis, that usually will allow CONOPT to make larger steps in the following one-dimensional searches.

5.8.10 Saddle Points and Directions of Negative Curvature

CONOPT is based on moving in a direction derived from the gradient, or the reduced gradient for models with constraints. If the reduced gradient projected on the bounds is zero, then the solution satisfies the first-order optimality conditions (the KKT or Karush-Kuhn-Tucker conditions), and it is standard procedure to stop. Unfortunately, this means that we can stop in a saddle-point.

It is not very common to move towards a saddle-point and get stuck there. However, it is not uncommon that the initial point, provided by a user or by default, is a saddle point. A simple example is the constraint `x*y =E= 1` started with `x.l = y.l = 0;` this construct can easily end with a locally infeasible solution. Another example is `minimize z`, `z =E= x*y` with the same starting point; it could end locally optimal without moving even though better points exist in the neighborhood.

CONOPT has added a procedure that tries to find a direction of negative curvature that can move the solution point away from a saddle-point. The procedure is only called in points that satisfy the first order optimality conditions and it is therefore usually only called once, and it is a cheap safeguard. The theory behind the method is developed for models without degeneracy and it works very well in practice for these models. Models with some kind of degeneracy (basic variables at bound or nonbasic variables with zero reduced cost) use the same procedure, but it is in this case only a heuristic that cannot be guaranteed to find a direction of negative curvature, even if one exists.

If you know that there are no directions of negative curvature, you can turn the procedure off by setting the logical option Flg_NegCurve to `false`. If the model is known to be convex you can set the logical option Flg_Convex to `true` and it will also turn this procedure off. The saving is usually very small, except for models that solve in very few iterations and for models with very many super basics.

There is no output in the log file for negative curvature. If a direction of negative curvature is found CONOPT will follow this direction and the optimization continues. Otherwise, the solution is declared locally optimal.

5.8.11 Alternative Sub-Models

During an optimization CONOPT can work with up to three different internal sub-models. These models are:

- Full Model: This model consists of the constraints in the user's model excluding all pre- and post-triangular constraints and with the definitional variables eliminated by their defining constraints. The objective function is the user's objective function.
- No-Penalty Model: This model consists of the Full Model excluding all penalty and mini-max constraints. This model does not have an objective function and it is terminated when a feasible solution has been found.
- Linear Feasibility Model: This model consists of the subset of linear constraints of the Full Model. The Linear Feasibility model is either solved without an objective function or minimizing a quadratic distance measure; this is discussed below.

The pre-triangular variables are considered fixed, and they do not appear in any of the sub-models. Their influence comes through their contribution to coefficients and constant terms. The post-triangular variables are considered intermediate variables in the definition of the objective function. They do not appear in the last two models that only are concerned with feasibility, and they only appear indirectly via the objective in the Full Model. The defined variables are considered intermediate variables in the definition of the remaining constraints in the same way as post-triangular variables are intermediate in the objective. The variables in the Full Model are all variables excluding pre- and post-triangular variables and excluding defined variables; this set can include variables that do not appear in any constraints. The constraints of the full models are all constraints excluding pre- and post-triangular constraints and with the definitional constraints logically eliminated. The variables in the Linear Feasibility Model and in the No-Penalty Model are the variables that appear in the constraints of these models, excluding pre-triangular variables.

5.8.11.1 No-Penalty Model

CONOPT always starts by searching for a feasible solution and the sub-models only play a role in this part of the optimization. Therefore, these sub-models are irrelevant if the initial point provided by the modeler is feasible. If there are many penalty and/or minimax constraints then the No-Penalty Model will be much smaller than the Full Model and it is more efficient to use the smaller model while searching for feasibility. The No-Penalty model is therefore only introduced for efficiency reasons. It is by default solved before the Full Model if all the following conditions are satisfied:

- The Flg_NoPen options is true (the default value),
- The model is not a CNS model,
- The user did not provide an initial basis,
- Some of the constraints in the No-Penalty Model are infeasible,
- The number of penalty and minimax constraints is more than the number of constraints in the Full Model multiplied by the value of option Rat_NoPen. The default value of `Rat_NoPen` is `0.1`, i.e., the No-Penalty Model is only defined and solved if it is at least 10% smaller than the Full Model.

An example of a log-file for a model with a No-Penalty model is from the **prolog.gms** model in the GAMS Library:

```
The user model has 23 constraints and 21 variables
with 129 Jacobian elements, 14 of which are nonlinear.
The Hessian of the Lagrangian has 0 elements on the diagonal,
4 elements below the diagonal, and 6 nonlinear variables.

Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
   0   0          1.2977431398E+03 (Input point)

The post-triangular part of the model has 1 constraints and variables.
There are 17 penalty and minimax constraints with 3 variables.

Reduced model without penalty components has 5 constraints and
17 variables with 46 Jacobian elements, 0 of which are nonlinear.


Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                    1.2960000000E+03 (Model without penalty constraints)
                    5.0625000000E+00 (After scaling)
                    0.0000000000E+00 (After adjusting individual variables)

Previous model terminated and penalty components are added back in.
Full preprocessed model has 22 constraints and 20 variables
with 120 Jacobian elements, 8 of which are nonlinear.


Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                    0.0000000000E+00 (After adjusting penalty variables)
```

There are no pre-triangular variables and constraints, and the pre-triangular line is therefore missing. There are also no definitional constraints and the line describing definitional constraints is also missing. On the other hand, there is a line saying that the model has 17 penalty and minimax constraints involving a total of 3 variables.

Since there is a significant number of penalty and minimax constraints, CONOPT starts by generating the No-Penalty model without the penalty and minimax constraints and without the pre- and post-triangular constraints. The 23 constraints in the user model are reduced by 1 post-triangular constraint and 17 minimax and penalty constraints, leaving 5 constraints. Similarly, the 21 variables in the user model are reduced by 1 post-triangular variable and 3 variables in the penalty and minimax constraints, leaving 17 variables.

This No-Penalty model is scaled and made feasible, in this case by the Adjust Initial Point procedure described above. After the No-Penalty model has become feasible CONOPT generates the full internal model, and the 3 variables from the penalty and minimax constraints are adjusted as described in the section on the Preprocessor and the Full Model is then feasible.

5.8.11.2 Linear Feasibility Model

The Linear Feasibility Model is introduced to avoid locally infeasible solutions where some linear constraints are infeasible and some feasible nonlinear constraint block for the linear constraint becoming feasible. The Linear Feasibility Model is used to produce a starting point to one of the nonlinear sub-models (the No-Penalty Model or the Full Model) that satisfies all linear constraints. If the Linear Feasibility Model is proved to be infeasible then the overall model is proved to be globally infeasible (independent of nonlinearities) and there is no reason to proceed with the nonlinear part of the model.

The Linear Feasibility Model is only useful if the model has some linear constraints and if the initial point provided by the modeler does not satisfy these constraints. If the Linear Feasibility Model finds a feasible solution to the linear constraints, it continues in one of four possible ways:

A. Use the first feasible solution directly without using extra effort to improve the solution.

B. Perform an approximate minimization of the weighted distance from the user's initial point. Only the variables that have non-default initial values are included in the distance measure, i.e. variables with an initial value (`xini`) that is different from zero projected on the bounds, i.e. `xini ne min(max(0,x.lo),x.up)`. The distance measure is `sqr( (x-xini) / max(1,abs(xini)) )`.

C. As in B, but with all variables included in the distance measure.

D. As in C, but with xini defined as a value between the lower and upper bounds.

Possibility A is fast, but it may give a starting point for the nonlinear model far from the initial point provided by the user. B is slower but gives a starting point for the nonlinear model that is close to the point provided by the user. The variables included in the distance measure in B are the variables that the user has given a value to and therefore are expected to be important. C and D are also slower, but because they have different objective functions, they may provide different starting points for the nonlinear model and may therefore avoid locally infeasible solutions.

Once the Linear Feasibility Model has been solved, with or without a distance objective function, the No-Penalty Model and/or the Full Model are started. All variables that appear in the Linear Feasibility Model are started from the solution values from this model and the remaining variables, that only appear in nonlinear constraints, are started from the values received from the modeler.

The order in which the sub-models are solved depends on the Linear Feasibility Model strategy, defined with option, Lin_Method:

1. If `Lin_Method` has the default value 1 then the initial point and basis is assumed to be good and CONOPT will start with the No-Penalty Model (only if the conditions mentioned above are satisfied) followed by the Full Model. If the model terminates locally optimal, unbounded, or on some resource limit (time, iterations, function evaluations) CONOPT terminates. We only build and solve the Linear Feasibility Model if the No-Penalty or Full model terminates locally infeasible. If the Linear Feasibility Model is infeasible, the overall model is infeasible and CONOPT terminates. Otherwise, we minimize objective B and use the solution point as a second starting point for the nonlinear model. If this attempt also terminates locally infeasible, we try to generate an alternative initial point with objective C and then with objective D. If all these attempts fail, the model is labeled locally infeasible.
2. With `Lin_Method = 2` CONOPT will start directly with the Linear Feasibility Model with objective A followed by the No-Penalty and Full models. If they are locally infeasible from this starting point, we followed the procedure from above with objective B, C, and then D.
3. `Lin_Method = 3` is like `Lin_Method = 2` except that the first objective A is skipped.

Sometimes the objective functions are identical. If all variables have default initial values, then A and B are the same, and if all variables have non-default initial values, then B and C are the same. CONOPT will check for this and will not solve identical sub-models.

The number of times we restart after a locally infeasible solution is controlled by the option, Num_Rounds. The default value is 4, i.e., we will by default try very hard to find a feasible point as shown in the example below. The value 1 will make CONOPT terminate immediately if a locally infeasible point is found, without going back to the Linear Feasibility Model. Num_Rounds can be used if you are not interested in spending extra time on a model that is likely to be infeasible. This is particularly relevant when CONOPT is used as the sub-solver inside SBB, where infeasible sub-problems are common.

An example of a log-file where all Linear Feasibility objectives are used is taken from `ex5_3_2.gms` in the GlobalLib collection of test models. It shows the repeated starts of the Linear Feasibility Model followed by the Full Preprocessed model:

```
   Preprocessed model has 22 variables and 16 constraints
   with 59 Jacobian elements, 24 of which are nonlinear.

   Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                      4.1200000000E+02 (Full preprocessed model)
                      8.4843750000E+00 (After scaling)
                      6.2500000000E-01 (After adjusting individual variables)
      1   1       1  6.2500000000E-01 0.0E+00       5 0.0E+00       T  T
      4   1       1  6.2500000000E-01 0.0E+00       4

** Infeasible solution. Reduced gradient less than tolerance.

   Initial linear feasibility model has 22 variables and 7 constraints
   with 22 linear Jacobian elements.
   Objective: Distance to initial point (nondefault variables)

   Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                      3.0200000000E+02 (Linear feasibility model)
                      3.1718750000E+00 (After scaling)

** Linear constraints feasible. Distance =    0.00000000000

   Iter Phase   Ninf     Distance      RGmax      NSB   Step  InItr MX OK
      6   4          0.0000000000E+00 0.0E+00       0

   Restarting preprocessed model from a new starting point.

   Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                      1.1000000000E+02 (Full preprocessed model)
                      5.3125000000E+00 (After scaling)
     11   2       1  6.2500000000E-01 0.0E+00       0

** Infeasible solution. There are no superbasic variables.


   Restarting linear feasibility model.
   Objective: Distance to initial point (all variables)

   Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                      0.0000000000E+00 (Linear feasibility model)
                      0.0000000000E+00 (After scaling)

** Linear constraints feasible. Distance =    90002.0000000

   Iter Phase   Ninf     Distance      RGmax      NSB   Step  InItr MX OK
     16   4          2.2501000000E+04 1.8E-12       5

   Restarting preprocessed model from a new starting point.

   Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                      1.8492500000E+02 (Full preprocessed model)
                      1.0775781250E+01 (After scaling)
     21   1       1  6.2500000000E-01 5.4E-03       3 0.0E+00       T  T
     26   1       1  6.2499999982E-01 3.7E-07       3 2.4E+03       T  T
     27   2       1  6.2500000000E-01 0.0E+00       2

** Infeasible solution. Reduced gradient less than tolerance.
```

```
   Restarting linear feasibility model.
   Objective: Distance to point away from bounds

   Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                     1.4210854715E-14 (Linear feasibility model)
                     5.5511151231E-17 (After scaling)

** Linear constraints feasible. Distance =    15.1922028230

   Iter Phase   Ninf     Distance      RGmax      NSB   Step  InItr MX OK
     31   4          2.5570990132E+00 1.1E-04      13 1.0E+00     2 F  T
     32   4          2.5570990132E+00 7.7E-11      13

   Restarting preprocessed model from a new starting point.

   Iter Phase   Ninf   Infeasibility   RGmax      NSB   Step  InItr MX OK
                     6.0836352222E+02 (Full preprocessed model)
                     9.7890253865E+00 (After scaling)
     35  S0       0  2.3859378417E+00                 1.0E+00     3 T  T

** Feasible solution. Value of objective =    1.86415945946

   Iter Phase   Ninf     Objective     RGmax      NSB   Step  InItr MX OK
     40   4          1.8641594595E+00 0.0E+00       0

** Optimal solution. There are no superbasic variables.
```

The Full Model is infeasible when started directly from the values provided by the user. The Linear Feasibility model is then solved to get a different starting point for the Full model where the linear part is feasible. The Full Model is also infeasible from this point, and it is necessary to solve the Linear Feasibility model three times with different objective functions before the full model becomes feasible.

If the model is defined to be convex with option `Flg_Convex = true` then a locally infeasible solution is labeled globally infeasible with ModelStat = 4, and the Linear Feasibility Model will not be used. A locally optimal solution is also labeled (globally) optimal with ModelStat = 1.

5.8.12 Scaling

The Tutorials and Examples section of the GAMS User's Guide has some general recommendations about scaling the user's model. These recommendations also apply to CONOPT.

In addition to the scaling done by the user, CONOPT will also scale the model internally by defining scale factors for both variables and constraints. The purpose of scaling is to get a scaled Jacobian matrix where the nonzero elements are neither very large nor very small, and scaled variables that are neither very large nor very small.

Variables are scaled with the formula xint = xuser / vscale and constraints are scaled with the formula gint = guser / gscale, where 'int' and 'user' represent the user model and the internal (scaled) model and vscale and gscale are scale factors for variables and constraints. As a result, the Jacobian elements changes with Jacint = Jacuser $*$ vscale / gscale.

For LP models the Jacobian is constant and scaling is usually done once. For nonlinear models the Jacobian changes with the solution point and scaling is therefore done at regular intervals.

In a nonlinear model a small variable or a small Jacobian element may be small because it is insignificant, but it may also be significant and just small in the current point. CONOPT is therefore using a scaling

procedure that scales large variables values and large Jacobian terms down to around 1, but it does not scale small variables values and small Jacobian terms up. The modeler is therefore advised to make sure that the expected solution values are not very small and that the terms in constraint are not all very small.

There are two options relevant for scaling: Tol_Scale_Max is the largest scaling factor that CONOPT will use. The default value is $1.e^{15}$ and it is applied to both variables and constraints. Tol_Scale_Min is the smallest scaling factor that CONOPT will use. The default value is 1 and it is also applied to both variables and constraints. The default values imply what was mentioned above, that very large terms can be scaled down, but small terms are not scaled up. If you have small but significant terms you may set Tol_Scale_Min below 1 but notice that insignificant small terms may also be scaled up and may disturb the solution process.

If the largest value of a variable, option Lim_Variable or the internal value of 'infinity', is increased you may consider increasing Tol_Scale_Max as well.

5.8.13 CNS Models

There is a special model class in GAMS called CNS - Constrained Nonlinear System. A constrained nonlinear system is a square system of equations, i.e., a model in which the number of non-fixed variables is equal to the number of constraints. A CNS model can be solved with a solve statement like

```
Solve <model> using CNS;
```

without an objective term. CONOPT will attempt to solve the constraints with respect to the non-fixed variables using Newton's method. When it works, it is often very fast, and it uses less memory than for the corresponding NLP model, but the solution process does not include a lot of the safeguards used for ordinary NLP models. The lack of safeguards means that the solution process will stop with an error message in some difficult situations and return the current intermediate infeasible solution. Some examples are shown below.

Slacks in inequalities are counted as non-fixed variables which effectively means that inequalities should not be binding. Bounds on the variables are allowed, especially to prevent function evaluation errors for functions that only are defined for some arguments, but the bounds should not be binding in the final solution.

Since there is no objective function the solution returned to GAMS will in all cases have marginal values equal to 0 or EPS, both for the variables and the constraints.

The termination messages for CNS models are different from the termination messages for optimization models. The message you hope for is this:

```
** Feasible solution to a square system.
```

that usually will be combined with ModelStat = 16 - Solved. If CONOPT in special cases can guarantee that the solution is unique, for example if the model is linear, then ModelStat will be 15 - Solved Unique.

There are two potential error termination messages related to CNS models. A model with the following two constraints

```
e1 ..   x1 +   x2 =e= 1;
e2 .. 2*x1 + 2*x2 =e= 2;
```

will result in this message

```
** Error in Square System: Pivot too small.
    e2: Pivot too small.
    x1: Pivot too small.
```

"Pivot too small" means that the set of constraints is linearly dependent on the current point and there is no unique search direction for Newton's method, so CONOPT terminates. In the listing file (but not the log-file) the message points to one variable and one constraint. However, this pair of constraint and variable just indicates that the linearly dependent set of constraints and variables include the constraint and variable mentioned. The offending constraint and variable will also be labeled 'DEPND' for linearly dependent in the equation listing. The error will usually be combined with ModelStat = 5 - Locally Infeasible. In cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities ModelStat will be 4 - Infeasible. If the constraints are linearly dependent but the current point satisfies the constraints then the solution status will be 17 - Solved Singular, indicating that the point is feasible, but there is probably a whole ray of feasible solution through the current point.

It should be mentioned that linear dependency and small pivot could be caused by the initial point and that the model could have a solution. An example is:

```
e1.. x1*x2 =E= 1;
e2.. x1+x2 =E= 3;
x1.l = 1; x2.l = 1;
```

A model with these two constraints and the bound

```
e1 .. x1 + x2 =e= 2;
e2 .. x1 - x2 =e= 0;
x1.lo = 1.5;
```

will result in the message

```
** Error in Square System: A variable tries to exceed its bound.
    x1: The variable tries to exceed its bound.
```

because the solution, (x1,x2) = (1,1) violates the lower bound on x1. This error case will also be combined with ModelStat = 5 - Locally Infeasible. In the cases where CONOPT can guarantee that the infeasibility is not caused by nonlinearities ModelStat will be 4 - Infeasible.

CNS can in some cases be used to generate an initial feasible solution: Fix a subset of the variables so the remaining model is uniquely solvable, solve this model with the CNS solver, reset the bounds on the fixed variables, and solve the original model. This two-solve method gives better control over the first feasible solution and can be useful for very large models.

Additional information on CNS can be found in the GAMS User's Guide

5.8.14 Multiple Threads

CONOPT can use multiple threads for some internal computations and GAMS can use multiple threads for function and derivative evaluations. In addition to function and derivative evaluations, multiple threads are only used for certain very large and dense computations and there are not so many of these in the types of models usually built with GAMS. In addition, multiple threads have some overhead, and they are therefore mainly useful for very large models. Currently, the best improvements have been for very large models with more than 100,000 variables or constraints, and especially for very large CNS models.

Threads can be turned on with the GAMS command-line option Threads=n or with the CONOPT option threads.

5.8.15 Loss of Feasibility

During the optimization you may sometimes see a phase 0 iteration and in rare cases you will see the message "Loss of Feasibility - Return to Phase 0". The background for this is as follows:

To work efficiently, CONOPT uses dynamic tolerances for feasibility and during the initial part of the optimization, where the objective changes rapidly, larger infeasibilities are accepted. As the change in objective between iterations becomes smaller it is necessary to solve the constraints more accurately so the uncertainty of the objective value caused by inaccurate constraints will remain smaller than the actual change. The accuracy of the objective is measured as the scalar product of the constraint residuals with the constraint marginals.

Sometimes it is necessary to revise the accuracy of the solution, for example because the algorithmic progress has slowed down or because the marginal of an inaccurate constraint has grown significantly after a basis change, e.g., when an inequality becomes binding. In these cases, CONOPT will reduce the feasibility tolerance and perform one or more Newton iterations on the basic variables. This will usually be very quick, and it happens silently. However, Newton's method may fail, for example in cases where the model is degenerate, and Newton tries to move a basic variable outside a bound. In this case CONOPT uses some special iteration similar to those discussed in section Phase 0 - Finding an Initial Feasible Solution and they are labeled Phase 0.

These Phase 0 iterations may not converge, for example if the degeneracy is significant, if the model is very nonlinear locally, if the model has many product terms involving variables at zero, or if the model is poorly scaled and some constraints contain very large terms. If the iterations do not converge, CONOPT will issue the "Loss of feasibility ..." message, return to the real Phase 0 procedure, find a feasible solution with the smaller tolerance, and resume the optimization.

In rare cases you will see that CONOPT cannot find a feasible solution after the tolerances have been reduced, even though it has declared the model feasible at an earlier stage. In this case CONOPT will stop, restore the last point that was labeled feasible and return this point as an intermediate feasible solution.

Although the problems may appear to be with too tight tolerances it is not a good idea to relax the feasibility tolerances. Relaxed feasibility tolerances will give more inaccurate objective function values that result in difficulties for the progress of the optimization as discussed in the next section.

5.8.16 Stalling

CONOPT will usually make steady progress towards the final solution. A degeneracy breaking strategy and the monotonicity of the objective function in other iterations should ensure that CONOPT cannot cycle. Unfortunately, there are a few places in the code where the objective function may move in the wrong direction and CONOPT may in fact cycle or move very slowly.

The objective value used to compare two points, in the following called the adjusted objective value, is computed as the true objective plus an accuracy adjustment term equal to the scalar product of the residuals with the marginals (see the previous section where this accuracy term also is used). The accuracy adjustment term is very useful in allowing CONOPT to work smoothly with inaccurate intermediate solutions. However, there is a disadvantage: the accuracy adjustment term can change even though the point itself does not change, namely when the marginals change in connection with a basis change. The adjusted objective is therefore not always monotone. When CONOPT loses feasibility and returns to Phase 0 there is an even larger chance of non-monotone behavior.

To prevent infinite loops and to allow the modeler to stop in cases with very slow progress CONOPT has an anti-stalling option. An iteration is counted as a stalled iteration if it is not degenerate and (1) the adjusted objective is worse than the best adjusted objective seen so far, or (2) the step length was zero without being degenerate (see OK = F in section Linear and Nonlinear Mode: Phase 1 to 4). CONOPT will stop if the number of consecutive stalled iterations (again not counting degenerate iterations) exceeds `Lim_StallIter` and `Lim_StallIter` is positive. The default value of `Lim_StallIter` is 100. The message will be:

```
** Feasible solution. The tolerances are minimal and
   there is no change in objective although the reduced
   gradient is greater than the tolerance.
```

Large models with very flat optima can sometimes be stopped prematurely due to stalling. If it is important to find a local optimum accurately then you may have to increase the value of `Lim_StallIter`.

5.8.17 APPENDIX A - Options

The options that ordinary GAMS users can access are listed below.

5.8.17.1 Algorithmic options

Option	Description	Default
Flg_AdjIniP	Flag for calling Adjust Initial Point	1
Flg_Convex	Flag for defining a model to be convex	0
Flg_Crash_Slack	Flag for pre-selecting slacks for the initial basis.	0
Flg_Dbg_Intv	Flag for debugging interval evaluations.	0
Flg_NegCurve	Flag for testing for negative curvature when apparently optimal	1
Flg_NoDefc	Flag for turning definitional constraints off. The default is false.	0
Flg_NoPen	Flag for allowing the Model without penalty constraints	1
Flg_SLPMode	Flag for enabling SLP mode.	1
Flg_SQPMode	Flag for enabling of SQP mode.	1
Flg_Square	Flag for Square System. Alternative to defining modeltype=CNS in GAMS	0
Flg_TraceCNS	Flag for tracing a CNS solution.	0
Frq_Rescale	Rescaling frequency.	5
Lim_Err_2DDir	Limit on errors in Directional Second Derivative evaluation.	10
Lim_Err_Fnc_Drv	Limit on number of function evaluation errors. Overwrites GAMS Domlim option	GAMS DomLim
Lim_Err_Hessian	Limit on errors in Hessian evaluation.	10
Lim_Iteration	Maximum number of iterations. Overwrites GAMS Iterlim option.	GAMS IterLim
Lim_NewSuper	Maximum number of new superbasic variables added in one iteration.	auto
Lim_RedHess	Maximum number of superbasic variables in the approximation to the Reduced Hessian.	auto
Lim_SlowPrg	Limit on number of iterations with slow progress (relative less than Tol_Obj_Change).	20
Lim_StallIter	Limit on the number of stalled iterations.	100
Lim_Start_Degen	Limit on number of degenerate iterations before starting degeneracy breaking strategy.	100

Option	Description	Default
Lim_Time	Time Limit. Overwrites the GAMS Reslim option.	GAMS ResLim
Lim_Variable	Upper bound on solution values and equation activity levels	1.e15
Lin_Method	Method used to determine if and/or which Linear Feasibility Models to use	1
Mtd_Dbg_1Drv	Method used by the function and derivative debugger.	0
Mtd_RedHess	Method for initializing the diagonal of the approximate Reduced Hessian	0
Mtd_Scale	Method used for scaling.	3
Mtd_Step_Phase0	Method used to determine the step in Phase 0.	auto
Mtd_Step_Tight	Method used to determine the maximum step while tightening tolerances.	0
Num_Rounds	Number of rounds with Linear Feasibility Model	4
Rat_NoPen	Limit on ratio of penalty constraints for the No_Penalty model to be solved	0.1
Tol_Bound	Bound filter tolerance for solution values close to a bound.	1.e-7
Tol_BoxSize	Initial box size for trust region models for overall model	10
Tol_BoxSize_Lin	Initial box size for trust region models for linear feasibility model	1000
Tol_Box_LinFac	Box size factor for linear variables applied to trust region box size	10
Tol_Def_Mult	Largest growth factor allowed in the block of definitional constraints	1.e4
Tol_Feas_Max	Maximum feasibility tolerance (after scaling).	1.e-7
Tol_Feas_Min	Minimum feasibility tolerance (after scaling).	4.e-10
Tol_Feas_Tria	Feasibility tolerance for triangular equations.	1.0e-8
Tol_Fixed	Tolerance for defining variables as fixed based on initial or derived bounds.	4.e-10
Tol_Jac_Min	Filter for small Jacobian elements to be ignored during scaling.	1.e-5
Tol_Linesearch	Accuracy of One-dimensional search.	0.2
Tol_Obj_Acc	Relative accuracy of the objective function.	3.0e-13
Tol_Obj_Change	Limit for relative change in objective for well-behaved iterations.	3.0e-12
Tol_Optimality	Optimality tolerance for reduced gradient when feasible.	1.e-7
Tol_Opt_Infeas	Optimality tolerance for reduced gradient when infeasible.	1.e-7
Tol_Opt_LinF	Optimality tolerance when infeasible in Linear Feasibility Model	1.e-10
Tol_Piv_Abs	Absolute pivot tolerance.	1.e-10
Tol_Piv_Abs_Ini	Absolute Pivot Tolerance for building initial basis.	1.e-7
Tol_Piv_Abs_NLTr	Absolute pivot tolerance for nonlinear elements in pre-triangular equations.	1.e-5
Tol_Piv_Ratio	Relative pivot tolerance during ratio-test	1.e-8
Tol_Piv_Rel	Relative pivot tolerance during basis factorizations.	0.05
Tol_Piv_Rel_Ini	Relative Pivot Tolerance for building initial basis	1.e-3

Option	Description	Default
Tol_Piv_Rel_Updt	Relative pivot tolerance during basis updates.	0.05
Tol_Scale2D_Min	Lower bound for scale factors based on large 2nd derivatives.	1.e-6
Tol_Scale_Max	Upper bound on scale factors.	1.e25
Tol_Scale_Min	Lower bound for scale factors computed from values and 1st derivatives.	1
Tol_Scale_Var	Lower bound on x in x∗Jac used when scaling.	1.e-5
Tol_Zero	Zero filter for Jacobian elements and inversion results.	1.e-20
Trace_MinStep	Minimum step between Reinversions when using TraceCNS.	0.001

5.8.17.2 Debugging options

Option	Description	Default
Flg_Interv	Flag for using intervals in the Preprocessor	1
Flg_Prep	Flag for using the Preprocessor	1
Flg_Range	Flag for identifying sets of ranged constraints	1
Lim_Dbg_1Drv	Flag for debugging of first derivatives	0
Lim_Hess_Est	Upper bound on second order terms.	1.e4
Lim_Msg_Dbg_1Drv	Limit on number of error messages from function and derivative debugger.	10

5.8.17.3 Output options

Option	Description	Default
Frq_Log_Simple	Frequency for log-lines for non-SLP/SQP iterations.	auto
Frq_Log_SlpSqp	Frequency for log-lines for SLP or SQP iterations.	auto
Lim_Msg_Large	Limit on number of error messages related to large function value and Jacobian elements.	10
Lim_Pre_Msg	Limit on number of error messages related to infeasible pre-triangle.	25

5.8.17.4 Interface options

Option	Description	Default
cooptfile		
Flg_2DDir	Flag for computing and using directional 2nd derivatives.	auto
Flg_Hessian	Flag for computing and using 2nd derivatives as Hessian of Lagrangian.	auto
HEAPLIMIT	Maximum Heap size in MB allowed	1e20
HessianMemFac	Memory factor for Hessian generation: Skip if Hessian elements > Nonlinear Jacobian elements∗HessianMemFac, 0 means unlimited.	0

Option	Description	Default
THREADC	Number of compatibility threads used for comparing different values of THREADS	1
THREADF	Number of threads used for function evaluation	1
threads	Number of threads used by Conopt internally	GAMS Threads

cooptfile *(string)*: ↩

Flg_2DDir *(boolean)*: Flag for computing and using directional 2nd derivatives. ↩

If turned on, make directional second derivatives (Hessian matrix times directional vector) available to CONOPT. The default is on, but it will be turned off of the model has external equations (defined with =X=) and the user has not provided directional second derivatives. If both the Hessian of the Lagrangian (see Flg_Hessian) and directional second derivatives are available then CONOPT will use both: directional second derivatives are used when the expected number of iterations in the SQP sub-solver is low and the Hessian is used when the expected number of iterations is large.

Default: auto

Flg_AdjIniP *(boolean)*: Flag for calling Adjust Initial Point ↩

If Flg_AdjIniP is on (the default) then the Adjust Initial Point routine is called after the pre-processor. Can be turned off if the routine is very slow.

Default: 1

Flg_Convex *(boolean)*: Flag for defining a model to be convex ↩

When turned on (the default is off) CONOPT knows that a local solution is also a global solution, whether it is optimal or infeasible, and it will be labeled appropriately. At the moment, Flg_NegCurve will be turned off. Other parts of the code will gradually learn to take advantage of this flag.

Default: 0

Flg_Crash_Slack *(boolean)*: Flag for pre-selecting slacks for the initial basis. ↩

When turned on (1) CONOPT will select all infeasible slacks as the first part of the initial basis.

Default: 0

Flg_Dbg_Intv *(boolean)*: Flag for debugging interval evaluations. ↩

Flg_Dbg_Intv controls whether interval evaluations are debugged. Currently we check that the lower bound does not exceed the upper bound for all intervals returned, both for function values and for derivatives.

Default: 0

Flg_Hessian *(boolean)*: Flag for computing and using 2nd derivatives as Hessian of Lagrangian. ↩

If turned on, compute the structure of the Hessian of the Lagrangian and make it available to CONOPT. The default is usually on, but it will be turned off if the model has external equations (defined with =X=) or cone constraints (defined with =C=) or if the Hessian becomes too dense. See also Flg_2DDir and HessianMemFac.

Default: auto

Flg_Interv *(boolean)*: Flag for using intervals in the Preprocessor ↩

If turned on (default), CONOPT will attempt to use interval evaluations in the preprocessor to determine if functions are monotone or if intervals for some of the variables can be excluded as infeasible.

Default: 1

Flg_NegCurve *(boolean)*: Flag for testing for negative curvature when apparently optimal ↩

When turned on (the default) CONOPT will try to identify directions with negative curvature when the model appears to be optimal. The objective is to move away from saddlepoints. Can be turned off when the model is known to be convex and cannot have negative curvature.

Default: 1

Flg_NoDefc *(boolean)*: Flag for turning definitional constraints off. The default is false. ↩

If Flg_NoDefc is on, the Preprocessor will not look for definitional constraints and variables.

Default: 0

Flg_NoPen *(boolean)*: Flag for allowing the Model without penalty constraints ↩

When turned on (the default) CONOPT will create and solve a smaller model without the penalty constraints and variables and the minimax constraints and variables if the remaining constraints are infeasible in the initial point. This is often a faster way to start the solution process.

Default: 1

Flg_Prep *(boolean)*: Flag for using the Preprocessor ↩

If turned on (default), CONOPT will use its preprocessor to try to determine pre- and post-triangluar components of the model and find definitional constraints.

Default: 1

Flg_Range *(boolean)*: Flag for identifying sets of ranged constraints ↩

If turned on (default), CONOPT will as part of its preprocessor look for sets of parallel linear constraints and turn each set into a single ranged constraints. There is currently a potential problem with the duals on these constraints and if duals are important ranges can be turned off with this flag.

Default: 1

Flg_SLPMode *(boolean)*: Flag for enabling SLP mode. ↩

If Flg_SLPMode is on (the default) then the SLP (sequential linear programming) sub-solver can be used, otherwise it is turned off.

Default: `1`

Flg_SQPMode *(boolean)*: Flag for enabling of SQP mode. ↩

If Flg_SQPMode is on (the default) then the SQP (sequential quadratic programming) sub-solver can be used, otherwise it is turned off.

Default: `1`

Flg_Square *(boolean)*: Flag for Square System. Alternative to defining modeltype=CNS in GAMS ↩

When turned on the modeler declares that this is a square system, i.e. the number of non-fixed variables must be equal to the number of constraints, no bounds must be active in the final solution, and the basis selected from the non-fixed variables must always be nonsingular.

Default: `0`

Flg_TraceCNS *(boolean)*: Flag for tracing a CNS solution. ↩

When turned on the model must, for fixed value of the objective variable, be a CNS model and must satisfy the conditions of a CNS. The model is first solved as a CNS with the initial value of the objective fixed and the objective is then minimized or maximized subject to the CNS constraints.

Default: `0`

Frq_Log_Simple *(integer)*: Frequency for log-lines for non-SLP/SQP iterations. ↩

Frq_Log_Simple and Frq_Log_SlpSqp can be used to control the amount of iteration send to the log file. The non-SLP/SQP iterations, i.e. iterations in phase 0, 1, and 3, are usually fast and writing a log line for each iteration may be too much, especially for smaller models. The default value for the log frequency for these iterations is therefore set to 10 for small models, 5 for models with more than 500 constraints or 1000 variables and 1 for models with more than 2000 constraints or 3000 variables.

Default: `auto`

Frq_Log_SlpSqp *(integer)*: Frequency for log-lines for SLP or SQP iterations. ↩

Frq_Log_Simple and Frq_Log_SlpSqp can be used to control the amount of iteration send to the log file. Iterations using the SLP and/or SQP sub-solver, i.e. iterations in phase 2 and 4, may involve several inner iterations and the work per iteration is therefore larger than for the non-SLP/SQP iterations and it may be relevant to write log lines more frequently. The default value for the log frequency is therefore 5 for small models and 1 for models with more than 500 constraints or 1000 variables.

Default: `auto`

Frq_Rescale *(integer)*: Rescaling frequency. ↩

The row and column scales are recalculated at least every Frq_Rescale new point (degenerate iterations do not count), or more frequently if conditions require it.

Default: `5`

HEAPLIMIT *(real)*: Maximum Heap size in MB allowed ↩

Range: [0, ∞]

Default: 1e20

HessianMemFac *(real)*: Memory factor for Hessian generation: Skip if Hessian elements > Nonlinear Jacobian elements∗HessianMemFac, 0 means unlimited. ↩

The Hessian of the Lagrangian is considered too dense therefore too expensive to evaluate and use, and it is not passed on to CONOPT if the number of nonzero elements in the Hessian of the Lagrangian is greater than the number of nonlinear Jacobian elements multiplied by HessianMemFac. See also Flg_Hessian. If HessianMemFac = 0.0 (the default value) then there is no limit on the number of Hessian elements.

Default: 0

Lim_Dbg_1Drv *(integer)*: Flag for debugging of first derivatives ↩

Lim_Dbg_1Drv controls how often the derivatives are tested. Debugging of derivatives is only relevant for user-written functions in external equations defined with =X=. The amount of debugging is controlled by Mtd_Dbg_1Drv. See Lim_Hess_Est for a definition of when derivatives are considered wrong.

Default: 0

value	meaning
-1	The derivatives are tested in the initial point only.
0	No debugging
+n	The derivatives are tested in all iterations that can be divided by Lim_Dbg_1Drv, provided the derivatives are computed in this iteration. (During phase 0, 1, and 3 derivatives are only computed when it appears to be necessary.)

Lim_Err_2DDir *(integer)*: Limit on errors in Directional Second Derivative evaluation. ↩

If the evaluation of Directional Second Derivatives (Hessian information in a particular direction) has failed more than Lim_Err_2DDir times CONOPT will not attempt to evaluate them any more and will switch to methods that do not use Directional Second Derivatives. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: 10

Lim_Err_Fnc_Drv *(integer)*: Limit on number of function evaluation errors. Overwrites GAMS Domlim option ↩

Function values and their derivatives are assumed to be defined in all points that satisfy the bounds of the model. If the function value or a derivative is not defined in a point CONOPT will try to recover by going back to a previous safe point (if one exists), but it will not do it more than at most Lim_Err_Fnc_Drv times. If CONOPT is stopped by functions or derivatives not being defined it will return with a intermediate infeasible or intermediate non-optimal model status.

Default: GAMS DomLim

Lim_Err_Hessian *(integer)*: Limit on errors in Hessian evaluation. ↩

If the evaluation of Hessian information has failed more than Lim_Err_Hessian times CONOPT will not attempt to evaluate it any more and will switch to methods that do not use the Hessian. Note that second order information may not be defined even if function and derivative values are well-defined, e.g. in an expression like power(x,1.5) at x=0.

Default: `10`

Lim_Hess_Est *(real)*: Upper bound on second order terms. ↩

The function and derivative debugger (see `Lim_Dbg_1Drv`) tests if derivatives computed using the modelers routine are sufficiently close to the values computed using finite differences. The term for the acceptable difference includes a second order term and uses Lim_Hess_Est as an estimate of the upper bound on second order derivatives in the model. Larger Lim_Hess_Est values will allow larger deviations between the user-defined derivatives and the numerically computed derivatives.

Default: `1.e4`

Lim_Iteration *(integer)*: Maximum number of iterations. Overwrites GAMS Iterlim option. ↩

The iteration limit can be used to prevent models from spending too many resources. You should note that the cost of the different types of CONOPT iterations (phase 0 to 4) can be very different so the time limit (GAMS Reslim or option `Lim_Time`) is often a better stopping criterion. However, the iteration limit is better for reproducing solution behavior across machines.

Default: `GAMS IterLim`

Lim_Msg_Dbg_1Drv *(integer)*: Limit on number of error messages from function and derivative debugger. ↩

The function and derivative debugger (see `Lim_Dbg_1Drv`) may find a very large number of errors, all derived from the same source. To avoid very large amounts of output CONOPT will stop the debugger after Lim_Msg_Dbg_1Drv error(s) have been found.

Default: `10`

Lim_Msg_Large *(integer)*: Limit on number of error messages related to large function value and Jacobian elements. ↩

Very large function value or derivatives (Jacobian elements) in a model will lead to numerical difficulties and most likely to inaccurate primal and/or dual solutions. CONOPT is therefore imposing an upper bound on the value of all function values and derivatives. This bound is 1.e30. If the bound is violated CONOPT will return with an intermediate infeasible or intermediate non-optimal solution and it will issue error messages for all the violating Jacobian elements, up to a limit of Lim_Msg_Large error messages.

Default: `10`

Lim_NewSuper *(integer)*: Maximum number of new superbasic variables added in one iteration. ↩

When there has been a sufficient reduction in the reduced gradient in one subspace new non-basics can be selected to enter the superbasis. The ones with largest reduced gradient of proper sign are selected, up to a limit. If Lim_NewSuper is positive then the limit is min(500,Lim_NewSuper). If Lim_NewSuper is zero (the default) then the limit is selected dynamically by CONOPT depending on model characteristics.

Default: `auto`

Lim_Pre_Msg *(integer)*: Limit on number of error messages related to infeasible pre-triangle. ↩

If the pre-processor determines that the model is infeasible it tries to define a minimal set of variables and constraints that define the infeasibility. If this set is larger than Lim_Pre_Msg elements the report is considered difficult to use and it is skipped.

Default: `25`

Lim_RedHess *(integer)*: Maximum number of superbasic variables in the approximation to the Reduced Hessian. ↩

CONOPT uses and stores a dense lower-triangular matrix as an approximation to the Reduced Hessian. The rows and columns correspond to the superbasic variable. This matrix can use a large amount of memory and computations involving the matrix can be time consuming so CONOPT imposes a limit on on the size. The limit is Lim_RedHess if Lim_RedHess is defined by the modeler and otherwise a value determined from the overall size of the model. If the number of superbasics exceeds the limit, CONOPT will switch to a method based on a combination of SQP and Conjugate Gradient iterations assuming some kind of second order information is available. If no second order information is available CONOPT will use a quasi-Newton method on a subset of the superbasic variables and rotate the subset as the reduced gradient becomes small.

Default: `auto`

Lim_SlowPrg *(integer)*: Limit on number of iterations with slow progress (relative less than Tol_Obj_Change). ↩

The optimization is stopped if the relative change in objective is less than `Tol_Obj_Change` for Lim_SlowPrg consecutive well-behaved iterations.

Default: `20`

Lim_StallIter *(integer)*: Limit on the number of stalled iterations. ↩

An iteration is considered a stalled iteration if there is no change in objective because the linesearch is limited by nonlinearities or numerical difficulties. Stalled iterations will have Step = 0 and F in the OK column of the log file. After a stalled iteration CONOPT will try various heuristics to get a better basis and a better search direction. However, the heuristics may not work as intended or they may even return to the original bad basis, especially if the model does not satisfy standard constraints qualifications and does not have a KKT point. To prevent cycling CONOPT will therefore stop after Lim_StallIter stalled iterations and returns an Intermediate Infeasible or Intermediate Nonoptimal solution.

Default: `100`

Lim_Start_Degen *(integer)*: Limit on number of degenerate iterations before starting degeneracy breaking strategy. ↩

The default CONOPT pivoting strategy has focus on numerical stability, but it can potentially cycle. When the number of consecutive degenerate iterations exceeds Lim_Start_Degen CONOPT will switch to a pivoting strategy that is guaranteed to break degeneracy but with slightly weaker numerical properties.

Default: `100`

Lim_Time *(real)*: Time Limit. Overwrites the GAMS Reslim option. ↩

The upper bound on the total number of seconds that can be used in the execution phase. There are only tests for time limit once per iteration. The default value is 10000. Lim_Time is overwritten by Reslim when called from GAMS.

Range: [0, ∞]

Default: GAMS ResLim

Lim_Variable *(real)*: Upper bound on solution values and equation activity levels ↩

If the value of a variable, including the objective function value and the value of slack variables, exceeds Lim_Variable then the model is considered to be unbounded and the optimization process returns the solution with the large variable flagged as unbounded. A bound cannot exceed this value.

Range: [1.e5, 1.e30]

Default: 1.e15

Lin_Method *(integer)*: Method used to determine if and/or which Linear Feasibility Models to use ↩

The Linear Feasibility Model can use different objectives: Objective 1 is no objective, i.e. the first point that satisfies the Linear Feasibility Model is used as a starting point for the Full Model. Objective 2 minimizes a scaled distance from the initial point for all variables defined by the modeler. Objective 3 minimizes a scaled distance from the initial point for all variables including those not defined by the modeler. Objective 4 minimizes a scaled distance from random a point selected away from bounds.

Default: 1

value	meaning
1	Ignore Linear Feasibility Model in the first round and use objective 2, 3, and 4 (see above) in the following rounds as long as model is locally infeasible. This is the default method.
2	Use Linear Feasibility Model with objective 1 in the first round and continue with objective 2, 3, and 4 in the following rounds as long as model is locally infeasible.
3	Use Linear Feasibility Model with objective 2 in the first round and continue with objective 3 and 4 in the following rounds as long as model is locally infeasible.

Mtd_Dbg_1Drv *(integer)*: Method used by the function and derivative debugger. ↩

The function and derivative debugger (turned on with Lim_Dbg_1Drv) can perform a fairly cheap test or a more extensive test, controlled by Mtd_Dbg_1Drv. See Lim_Hess_Est for a definition of when derivatives are considered wrong. All tests are performed in the current point found by the optimization algorithm.

Default: 0

value	meaning
0	Perform tests for sparsity pattern and tests that the numerical values of the derivatives appear to be correct. This is the default.
1	As 0 plus make extensive test to determine if the functions and their derivatives are continuous around the current point. These tests are much more expensive and should only be used of the cheap test does not find an error but one is expected to exist.

Mtd_RedHess *(integer)*: Method for initializing the diagonal of the approximate Reduced Hessian ↩

Each time a nonbasic variable is made superbasic a new row and column is added to the approximate Reduced Hessian. The off-diagonal elements are set to zero and the diagonal to a value controlled by Mtd_RedHess:

Default: `0`

value	meaning
`0`	The new diagonal element is set to the geometric mean of the existing diagonal elements. This gives the new diagonal element an intermediate value and new superbasic variables are therefore not given any special treatment. The initial steps should be of good size, but build-up of second order information in the new sub-space may be slower. The larger diagonal element may also in bad cases cause premature convergence.
`1`	The new diagonal elements is set to the minimum of the existing diagonal elements. This makes the new diagonal element small and the importance of the new superbasic variable will therefore be high. The initial steps can be rather small, but better second order information in the new sub-space should be build up faster.

Mtd_Scale *(integer)*: Method used for scaling. ↩

CONOPT will by default use scaling of the equations and variables of the model to improve the numerical behavior of the solution algorithm and the accuracy of the final solution (see also `Frq_Rescale`.) The objective of the scaling process is to reduce the values of all large primal and dual variables as well as the values of all large first derivatives so they become closer to 1. Small values are usually not scaled up, see `Tol_Scale_Max` and `Tol_Scale_Min`. Scaling method 3 is recommended. The others are only kept for backward compatibility.

Default: `3`

value	meaning
`0`	Scaling is based on repeatedly dividing the rows and columns by the geometric means of the largest and smallest elements in each row and column. Very small elements less than Tol_Jac_Min are considered equal to Tol_Jac_Min.
`1`	Similar to 3 below, but the projection on the interval [Tol_Scale_Min,Tol_Scale_Max] is applied at a different stage. With method 1, abs(X)∗abs(Jac) with small X and very large Jac is scaled very aggressively with a factor abs(Jac). With method 3, the scale factor is abs(X)∗abs(Jac). The difference is seen in models with terms like Sqrt(X) close to X = 0.
`2`	As 1 but the terms are computed based on a moving average of the squares X and Jac. The purpose of the moving average is to keep the scale factor more stable. This is often an advantage, but for models with very large terms (large variables and in particular large derivatives) in the initial point the averaging process may not have enough time to bring the scale factors into the right region.
`3`	Rows are first scaled by dividing by the largest term in the row, then columns are scaled by dividing by by the maximum of the largest term and the value of the variable. A term is here defined as abs(X)∗abs(Jac) where X is the value of the variable and Jac is the value of the derivative (Jacobian element). The scale factor are then projected on the interval between Tol_Scale_Min and Tol_Scale_Max.

Mtd_Step_Phase0 *(integer)*: Method used to determine the step in Phase 0. ↩

The steplength used by the Newton process in phase 0 is computed by one of two alternative methods controlled by Mtd_Step_Phase0:

Default: `auto`

value	meaning
`0`	The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this method. It is the default method for optimization models.
`1`	A method based on bending (projecting the target values of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum. Linear constraints that initially are feasible may become infeasible due to bending.

Mtd_Step_Tight *(integer)*: Method used to determine the maximum step while tightening tolerances. ↩

The steplength used by the Newton process when tightening tolerances is computed by one of two alternative methods controlled by Mtd_Step_Tight:

Default: `0`

value	meaning
`0`	The standard ratio test method known from the Simplex method. CONOPT adds small perturbations to the bounds to avoid very small pivots and improve numerical stability. Linear constraints that initially are feasible will remain feasible with this default method.
`1`	A method based on bending (projecting the target value of the basic variables on the bounds) until the sum of infeasibilities is close to its minimum.

Num_Rounds *(integer)*: Number of rounds with Linear Feasibility Model ↩

Lin_Method defined which Linear Feasibility Model are going to be solved if the previous models end Locally Infeasible. The number of rounds is limited by Num_Rounds.

Range: {`1`, ..., `4`}

Default: `4`

Rat_NoPen *(real)*: Limit on ratio of penalty constraints for the No_Penalty model to be solved ↩

The No-Penalty model can only be generated and solved if the number of penalty and minimax constraints exceed Rat_NoPen times the constraints in the Full Model.

Default: `0.1`

THREADC *(integer)*: Number of compatibility threads used for comparing different values of THREADS ↩

Range: {`0`, ..., ∞}

Default: `1`

THREADF *(integer)*: Number of threads used for function evaluation ↩

Range: {`0`, ..., ∞}

Default: `1`

threads *(integer)*: Number of threads used by Conopt internally ↩

Range: {0, ..., ∞}

Default: GAMS Threads

Tol_Bound *(real)*: Bound filter tolerance for solution values close to a bound. ↩

A variable is considered to be at a bound if its distance from the bound is less than Tol_Bound * Max(1,ABS(Bound)). The tolerance is used to build the initial bases and is used to flag variables during output.

Range: [3.e-13, 1.e-5]

Default: 1.e-7

Tol_BoxSize *(real)*: Initial box size for trust region models for overall model ↩

The new Phase 0 methods solves an LP model based on a scaled and linearized version of the model with an added trust region box constraint around the initial point. Tol_BoxSize defines the size of the initial trust region box. During the optimization the trust region box is adjusted based on how well the linear approximation fits the real model.

Range: [0.01, 1.e6]

Default: 10

Tol_BoxSize_Lin *(real)*: Initial box size for trust region models for linear feasibility model ↩

Similar to Tol_BoxSize but applied to the linear feasibility model. Since this model has linear constraints the default initial box size is larger.

Range: [0.01, 1.e8]

Default: 1000

Tol_Box_LinFac *(real)*: Box size factor for linear variables applied to trust region box size ↩

The trust region box used in the new Phase 0 method limits the change of variables so 2nd order terms will not become too large. Variables that appear linearly do not have 2nd order terms and the initial box size is therefore larger by a factor Tol_Box_LinFac.

Range: [1, 1.e4]

Default: 10

Tol_Def_Mult *(real)*: Largest growth factor allowed in the block of definitional constraints ↩

The block of definitional constraints form a triangular matrix. This triangular matrix can hide large accumulating growth factors that can lead to increases in the initial sum of infeasibilities and to numerical instability. Tol_Def_Mult is an upper bound on these growth factors. If it is exceeded some critical chains of definitional constraints will be broken leading to a larger internal model, that should be numerically better behaved.

Range: [1.01, ∞]

Default: 1.e4

Tol_Feas_Max *(real)*: Maximum feasibility tolerance (after scaling). ↩

The feasibility tolerance used by CONOPT is dynamic. As long as we are far from the optimal solution and make large steps it is not necessary to compute intermediate solutions very accurately. When we approach the optimum and make smaller steps we need more accuracy. Tol_Feas_Max is the upper bound on the dynamic feasibility tolerance and Tol_Feas_Min is the lower bound. It is NOT recommended to use loose feasibility tolerances since the objective, including the sum of infeasibility objective, will be less accurate and it may prevent convergence.

Range: [1.e-10, 1.e-3]

Default: 1.e-7

Tol_Feas_Min *(real)*: Minimum feasibility tolerance (after scaling). ↩

See Tol_Feas_Max for a discussion of the dynamic feasibility tolerances used by CONOPT.

Range: [3.e-13, 1.e-5]

Default: 4.e-10

Tol_Feas_Tria *(real)*: Feasibility tolerance for triangular equations. ↩

Triangular equations are usually solved to an accuracy of Tol_Feas_Min. However, if a variable reaches a bound or if a constraint only has pre-determined variables then the feasibility tolerance can be relaxed to Tol_Feas_Tria.

Range: [3.e-13, 1.e-4]

Default: 1.0e-8

Tol_Fixed *(real)*: Tolerance for defining variables as fixed based on initial or derived bounds. ↩

A variable is considered fixed if the distance between the bounds is less than Tol_Fixed $*$ Max(1,Abs(Bound)). The tolerance is used both on the users original bounds and on the derived bounds that the preprocessor implies from the constraints of the model.

Range: [3.e-13, 1.e-8]

Default: 4.e-10

Tol_Jac_Min *(real)*: Filter for small Jacobian elements to be ignored during scaling. ↩

A Jacobian element is considered insignificant if it is less than Tol_Jac_Min. The value is used to select which small values are scaled up during scaling of the Jacobian. Is only used with scaling method Mtd_Scale = 0.

Range: [1.e-7, 1.e-3]

Default: 1.e-5

Tol_Linesearch *(real)*: Accuracy of One-dimensional search. ↩

The onedimensional search is stopped if the expected decrease in then objective estimated from a quadratic approximation is less than Tol_Linesearch times the decrease so far in this onedimensional search.

Range: [0.05, 0.8]

Default: 0.2

Tol_Obj_Acc *(real)*: Relative accuracy of the objective function. ↩

It is assumed that the objective function can be computed to an accuracy of Tol_Obj_Acc ∗ max(1, abs(Objective)). Smaller changes in objective are considered to be caused by round-off errors.

Range: [3.0e-14, 1.e-6]

Default: 3.0e-13

Tol_Obj_Change *(real)*: Limit for relative change in objective for well-behaved iterations. ↩

The change in objective in a well-behaved iteration is considered small and the iteration counts as slow progress if the change is less than Tol_Obj_Change ∗ Max(1,Abs(Objective)). See also Lim_SlowPrg.

Range: [3.0e-13, 1.0e-5]

Default: 3.0e-12

Tol_Optimality *(real)*: Optimality tolerance for reduced gradient when feasible. ↩

The reduced gradient is considered zero and the solution optimal if the largest superbasic component of the reduced gradient is less than Tol_Optimality.

Range: [3.e-13, 1]

Default: 1.e-7

Tol_Opt_Infeas *(real)*: Optimality tolerance for reduced gradient when infeasible. ↩

The reduced gradient is considered zero and the solution infeasible if the largest superbasic component of the reduced gradient is less than Tol_Opt_Infeas.

Range: [3.e-13, 1]

Default: 1.e-7

Tol_Opt_LinF *(real)*: Optimality tolerance when infeasible in Linear Feasibility Model ↩

This is a special optimality tolerance used when the Linear Feasibility Model is infeasible. Since the model is linear the default value is smaller than for nonlinear submodels.

Range: [3.e-13, 1.e-4]

Default: 1.e-10

Tol_Piv_Abs *(real)*: Absolute pivot tolerance. ↩

During LU-factorization of the basis matrix a pivot element is considered large enough if its absolute value is larger than Tol_Piv_Abs. There is also a relative test, see Tol_Piv_Rel.

Range: [2.2e-16, 1.e-7]

Default: 1.e-10

Tol_Piv_Abs_Ini *(real)*: Absolute Pivot Tolerance for building initial basis. ↩

Absolute pivot tolerance used during the search for a first logically non-singular basis. The default is fairly large to encourage a better conditioned initial basis.

Range: [3.e-13, 1.e-3]

Default: 1.e-7

Tol_Piv_Abs_NLTr *(real)*: Absolute pivot tolerance for nonlinear elements in pre-triangular equations. ↩

The smallest pivot that can be used for nonlinear or variable Jacobian elements during the pre-triangular solve. The pivot tolerance for linear or constant Jacobian elements is Tol_Piv_Abs. The value cannot be less that Tol_Piv_Abs.

Range: [2.2e-16, 1.e-3]

Default: 1.e-5

Tol_Piv_Ratio *(real)*: Relative pivot tolerance during ratio-test ↩

During ratio-rests, the lower bound on the slope of a basic variable to potentially leave the basis is Tol_Piv_Ratio $*$ the largest term in the computation of the tangent.

Range: [1.e-10, 1.e-6]

Default: 1.e-8

Tol_Piv_Rel *(real)*: Relative pivot tolerance during basis factorizations. ↩

During LU-factorization of the basis matrix a pivot element is considered large enough relative to other elements in the column if its absolute value is at least Tol_Piv_Rel $*$ the largest absolute value in the column. Small values or Tol_Piv_Rel will often give a sparser basis factorization at the expense of the numerical accuracy. The value used internally is therefore adjusted dynamically between the users value and 0.9, based on various statistics collected during the solution process. Certain models derived from finite element approximations of partial differential equations can give rise to poor numerical accuracy and a larger user-value of Tol_Piv_Rel may help.

Range: [1.e-3, 0.9]

Default: 0.05

Tol_Piv_Rel_Ini *(real)*: Relative Pivot Tolerance for building initial basis ↩

Relative pivot tolerance used during the search for a first logically non-singular basis.

Range: [1.e-4, 0.9]

Default: 1.e-3

Tol_Piv_Rel_Updt *(real)*: Relative pivot tolerance during basis updates. ↩

During basischanges CONOPT attempts to use cheap updates of the LU-factors of the basis. A pivot is considered large enough relative to the alternatives in the column if its absolute value is at least Tol_Piv_Rel_Updt $*$ the other element. Smaller values of Tol_Piv_Rel_Updt will allow sparser basis updates but may cause accumulation of larger numerical errors.

Range: [1.e-3, 0.9]

Default: 0.05

Tol_Scale2D_Min *(real)*: Lower bound for scale factors based on large 2nd derivatives. ↩

Scaling of the model is in most cases based on the values of the variables and the first derivatives. However, if the scaled variables and derivatives are reasonable but there are large values in the Hessian of the Lagrangian (the matrix of 2nd derivatives) then the lower bound on the scale factor can be made smaller than Tol_Scale_Min. CONOPT will try to scale variables with large 2nd derivatives by one over the square root of the diagonal elements of the Hessian. However, the revised scale factors cannot be less than Tol_Scale2D_Min.

Range: [`1.e-9`, `1`]

Default: `1.e-6`

Tol_Scale_Max *(real)*: Upper bound on scale factors. ↩

Scale factors are projected on the interval from Tol_Scale_Min to Tol_Scale_Max. Is used to prevent very large or very small scale factors due to pathological types of constraints. The upper limit is selected such that Square(X) can be handled for X close to Lim_Variable. More nonlinear functions may not be scalable for very large variables.

Range: [`1`, `1.e30`]

Default: `1.e25`

Tol_Scale_Min *(real)*: Lower bound for scale factors computed from values and 1st derivatives. ↩

Scale factors used to scale variables and equations are projected on the range Tol_Scale_Min to Tol_Scale_Max. The limits are used to prevent very large or very small scale factors due to pathological types of constraints. The default value for Tol_Scale_Min is 1 which means that small values are not scaled up. If you need to scale small value up towards 1 then you must define a value of Tol_Scale_Min < 1.

Range: [`1.e-10`, `1`]

Default: `1`

Tol_Scale_Var *(real)*: Lower bound on x in x∗Jac used when scaling. ↩

Rows are scaled so the largest term x∗Jac is around 1. To avoid difficulties with models where Jac is very large and x very small a lower bound of Tol_Scale_Var is applied to the x-term.

Range: [`1.e-8`, `1`]

Default: `1.e-5`

Tol_Zero *(real)*: Zero filter for Jacobian elements and inversion results. ↩

Contains the smallest absolute value that an intermediate result can have. If it is smaller, it is set to zero. It must be smaller than `Tol_Piv_Abs`/10.

Default: `1.e-20`

Trace_MinStep *(real)*: Minimum step between Reinversions when using TraceCNS. ↩

The optimization is stopped with a slow convergence message if the change in trace variable or objective is less than this tolerance between reinversions for more than two consecutive reinversions. The step is scaled by the distance from the initial value to the critical bound.

Range: [`0`, `1`]

Default: `0.001`

5.9 CONVERT

5.9.1 Introduction

CONVERT is a utility which transforms a GAMS model instance into a scalar model where all confidential information has been removed or into formats used by other modeling and solution systems. CONVERT is designed to achieve the following goals:

- Permit users to convert a confidential model into GAMS scalar format so that any idenifiable structure is removed. It can then be passed on to others for investigation without confidentiality being lost.
- A way of sharing GAMS test problems for use with other modeling systems or solvers.

CONVERT comes free of charge with any licensed GAMS system and can convert GAMS models into a number of formats, see Section Target languages and formats for a list.

5.9.2 How to use CONVERT

CONVERT is run like any other GAMS solver. From the command line this is:

```
>> gams modelname modeltype=convert
```

where `modelname` is the GAMS model name and `modeltype` the solver indicator for a particular model type (e.g. LP, MIP, RMIP, QCP, MIQCP, RMIQCP, NLP, DNLP, CNS, MINLP, or MCP). CONVERT can also be specified via the option statement within the model itself before the solve statement:

```
option modeltype=convert;
```

5.9.3 The GAMS Scalar Format

By default, CONVERT generates a scalar GAMS model (`gams.gms`) from the input model. The scalar model exhibits the following characteristics:

- A model without sets or indexed parameters. It does not exhibit any of the advanced characteristics of modeling systems and is easily transformable.
- A model with a new set of individual variables, depicting each variable in the GAMS model as one of 3 types: positive, integer or binary. Each variable is numbered sequentially, i.e. all positive GAMS variables are mapped into n single variables `x1, x2, ..., xn`.
- A model with individual equations depicting each variable in the GAMS model. All equations are also numbered sequentially, that is equations `e1, e2, ..., em`.

Equation and variable bounds, as well as variable starting values are preserved from the original GAMS formulation.

As an example, suppose the user wishes to translate the GAMS Model Library model **trnsport** into scalar format, One would run

```
gams trnsport.gms lp=convert
```

which would generate the following scalar model `gams.gms`:

```
* LP written by GAMS Convert at 11/19/20 15:28:05
*
* Equation counts
*     Total        E        G        L        N        X        C        B
*         6        1        3        2        0        0        0        0
*
* Variable counts
*                  x        b        i      s1s      s2s       sc       si
*     Total     cont   binary  integer     sos1     sos2    scont     sint
*         7        7        0        0        0        0        0        0
* FX      0
*
* Nonzero counts
*     Total    const       NL
*        19       19        0

* Solve m using LP minimizing x7;

Variables
    x1,x2,x3,x4,x5,x6,x7;

Positive Variables
    x1,x2,x3,x4,x5,x6;

Equations
    e1,e2,e3,e4,e5,e6;

e1..  -0.225 * x1 - 0.153 * x2 - 0.162 * x3 - 0.225 * x4 - 0.162 * x5 - 0.126 *
      x6 + x7 =E= 0;
e2..  x1 + x2 + x3 =L= 350;
e3..  x4 + x5 + x6 =L= 600;
e4..  x1 + x4 =G= 325;
e5..  x2 + x5 =G= 300;
e6..  x3 + x6 =G= 275;

Model m / all /;

m.limrow = 0;
m.limcol = 0;

Solve m using LP minimizing x7;
```

Note that the resulting scalar model does not contain any of the descriptive information about the data or the context of the constraints.

Additionally, a dictionary file (`dict.txt`) is created by default which specifies a mapping between the variable and equation names in the scalar model and their corresponding names in the original model.

For the above example, the dictionary file is

```
LP written by GAMS Convert at 11/19/20 15:28:05
```

```
Equation counts
    Total        E        G        L        N        X        C        B
        6        1        3        2        0        0        0        0

Variable counts
                 x        b        i      s1s      s2s       sc       si
    Total     cont   binary  integer     sos1     sos2    scont     sint
        7        7        0        0        0        0        0        0
FX      0

Nonzero counts
    Total    const       NL
       19       19        0

Equations 1 to 6
  e1  cost
  e2  supply(seattle)
  e3  supply(san-diego)
  e4  demand(new-york)
  e5  demand(chicago)
  e6  demand(topeka)

Variables 1 to 7
  x1  x(seattle,new-york)
  x2  x(seattle,chicago)
  x3  x(seattle,topeka)
  x4  x(san-diego,new-york)
  x5  x(san-diego,chicago)
  x6  x(san-diego,topeka)
  x7  z
```

Conversion of a GAMS model to a scalar one may be handy for model debugging. However, in this case, it may be good to retain the original variable and equation names. The following simple sed command attempts to achieve this:

```
sed -n -e "s:^ *\([exbi][0-9][0-9]*\)  \(.*\):s/\1/\2/g:gp" dict.txt | sed -n '1!G;h;$p' > mod.txt
sed -f mod.txt gams.gms
```

For the above example, this outputs:

```
Variables
    x(seattle,new-york),x(seattle,chicago),x(seattle,topeka),x(san-diego,new-york),x(san-diego,chicag

Positive Variables
    x(seattle,new-york),x(seattle,chicago),x(seattle,topeka),x(san-diego,new-york),x(san-diego,chicag

Equations
    cost,supply(seattle),supply(san-diego),demand(new-york),demand(chicago),demand(topeka);

cost..  -0.225 * x(seattle,new-york) - 0.153 * x(seattle,chicago) - 0.162 * x(seattle,topeka) - 0.225
      x(san-diego,topeka) + z =E= 0;
supply(seattle)..  x(seattle,new-york) + x(seattle,chicago) + x(seattle,topeka) =L= 350;
supply(san-diego)..  x(san-diego,new-york) + x(san-diego,chicago) + x(san-diego,topeka) =L= 600;
demand(new-york)..  x(seattle,new-york) + x(san-diego,new-york) =G= 325;
demand(chicago)..  x(seattle,chicago) + x(san-diego,chicago) =G= 300;
demand(topeka)..  x(seattle,topeka) + x(san-diego,topeka) =G= 275;
```

Of course, this is not a valid GAMS code and cannot be compiled, but it may be sufficient to view the model algebra as generated by the GAMS compiler.

By using

```
sed -n -e "y/(),-/____/" -e "s:^ *\([exbi][0-9][0-9]*\)  \(.*\):s/\1/\2/g:gp" dict.txt | sed -n '1!G
sed -f mod.txt gams.gms
```

one gets for this example

```
Variables
    x_seattle_new_york_,x_seattle_chicago_,x_seattle_topeka_,x_san_diego_new_york_,x_san_diego_chica

Positive Variables
    x_seattle_new_york_,x_seattle_chicago_,x_seattle_topeka_,x_san_diego_new_york_,x_san_diego_chica

Equations
    cost,supply_seattle_,supply_san_diego_,demand_new_york_,demand_chicago_,demand_topeka_;

cost..  -0.225 * x_seattle_new_york_ - 0.153 * x_seattle_chicago_ - 0.162 * x_seattle_topeka_ - 0.225
      x_san_diego_topeka_ + z =E= 0;
supply_seattle_..  x_seattle_new_york_ + x_seattle_chicago_ + x_seattle_topeka_ =L= 350;
supply_san_diego_..  x_san_diego_new_york_ + x_san_diego_chicago_ + x_san_diego_topeka_ =L= 600;
demand_new_york_..  x_seattle_new_york_ + x_san_diego_new_york_ =G= 325;
demand_chicago_..  x_seattle_chicago_ + x_san_diego_chicago_ =G= 300;
demand_topeka_..  x_seattle_topeka_ + x_san_diego_topeka_ =G= 275;
```

This can even be compiled by GAMS and gives the correct solution.

The proposed commands come with several limitations and may not produce in all cases the desired output. For example, wrong results would be printed if the original model contains variable or equation names that start with `{b,i,e,x}[digit]`. Also semicontinuous or semiinteger variables or special ordered sets are not supported by the above. We leave it to the experienced user to extend the command appropriately.

5.9.4 The OSiL Format

The Optimization Services Instance Language (OSiL) [70] specifies an XML-based format to represent optimization problem instances. GAMS/CONVERT can write MINLP model instances in `OSiL format`. Expression trees are written in `OSnL format`.

Next to the indexed operations for sum, product, minimum, and maximum, and the operations for subtraction and division, the following intrinsic functions are mapped to their `OSnL` counterparts: `sqr`, `sqrt`, `exp`, `log`, `log2`, `log10`, `abs`, `cos`, `sin`, `tan`, `arccos`, `arcsin`, `arctan`, `sinh`, `cosh`, `tanh`, `pi`, `div`, `gamma`, `loggamma`, `floor`, `ceil`, `round`, `trunc`, `sign fact`, `binomial`. The functions `cvPower`, `power`, `rpower`, `vcpower` are all mapped to OSnL's `power` operator, thus conditions on arguments are not preserved. Functions `arctan2`, `centropy`, `edist`, `errorf`, and `poly` are represented by an expression according to their algebraic definition. The intrinsic functions `signpower`, `entropy`, `sigmoid`, `gammareg`, `beta`, `logbeta`, and `betareg` are also written to OSiL files, but do not follow the OSnL standard (as it currently does not offer these functions). Thus, OSiL readers may reject XML files that use these functions. Finally, also the logical functions are written to OSiL by using their OSnL counterpart.

5.9.5 User-Specified Options

CONVERT options are passed on through option files. If you specify <modelname>.optfile = 1; before the SOLVE statement in your GAMS model, CONVERT will look for and read an option file with the name convert.opt (see The Solver Options File for general use of solver option files). The syntax for the CONVERT option file is

```
optname value
```

with one option on each line. For example,

```
ampl
```

This option file would tell CONVERT to produce an AMPL input file. For file format options, the user can specify the filename for the file to be generated. For example, the option file entry

```
lingo myfile.lng
```

would generate a LINGO input file format called myfile.lng. Using the option lingo by itself, would produce the default output file for that option (lingo.lng).

All available options are listed in the following tables.

5.9.5.1 Target languages and formats

Option	Description	Default
All	Generates all supported file formats	
Ampl	Generates Ampl input file	ampl.mod
AmplNL	Generates Ampl .nl file	ampl.nl
CplexLP	Generates CPLEX LP format input file	cplex.lp
CplexMPS	Generates CPLEX MPS format input file	cplex.mps
Dict	Generates Convert to GAMS Dictionary	dict.txt
DictMap	Generates Convert to GAMS Dictionary Map	dictmap.gdx
DumpGDX	Generates GDX with model data incl. Jacobian and Hessian evaluated at current point	dump.gdx
FileList	Generates file list of file formats generated	files.txt
FixedMPS	Generates fixed format MPS file	fixed.mps
Gams	Generates GAMS scalar model. This is the default conversion format used.	gams.gms
JuMP	Generates JuMP scalar model	jump.jl
Lingo	Generates Lingo input file	lingo.lng
NLP2dual	Generates the Wolfe dual of a smooth optimization model	gamsdual.gms
NLP2MCP	Generates GAMS scalar MCP model	gamsmcp.gms
OSiL	Generates Optimization Services instance Language (OSiL) file	osil.xml
Pyomo	Generates Pyomo Concrete scalar model	pyomo.py

5.9.5.2 Other options

Option	Description	Default
AmplNLBin	Enables binary .nl file	0
AmplNlInitDual	Which initial equation marginal values to write to .nl file 0: Write no values 1: Write only nondefault values 2: Write all values	1
AmplNlInitPrimal	Which initial variable level values to write to .nl file 0: Write no values 1: Write only nondefault values 2: Write all values	2
DualType	Controls type of Wolfe dual to generate in NLP2dual None: No Wolfe dual generated NLPScalarBounds: NLP dual where variable bounds become scalars used in equations NLPConstantBounds: NLP dual where finite variable bounds become constants in equations BiLevel: Bilevel model with outer problem optimizing over the duals MPEC: MPEC obtained by explicitly including FOC of BiLevel inner problem	None
GDXHessian	Enable hessian information for DumpGDX	0
GDXNames	Enable variable and equation names for DumpGDX	1
GDXQuadratic	Enable quadratic information for DumpGDX	0
GDXUELs	Enable UELs for DumpGDX	1
GmsInsert	Line to be inserted before the solve statement	$if NOT 'gams.u1' == '' $include 'gams.u1'
HeaderTimeStamp	Control format of time stamp in header of output file None: Use no timestamp default: Use the traditional default timestamp	default
IntervalEval	Include interval evaluations in DumpGDX	0
ObjVar	Name of objective variable	GAMS index name, e.g. x1
PermuteEqus	Random seed for permutation of equations (0: no permutation)	0
PermuteVars	Random seed for permutation of variables (0: no permutation)	0

Option	Description	Default
QExtractAlg	quadratic extraction algorithm in GAMS interface 0: Automatic 1: ThreePass: Uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage. 2: DoubleForward: Uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists. 3: Concurrent: Uses ThreePass and DoubleForward in parallel. As soon as one finishes, the other one stops.	0
Reform	Force reformulations	100
SkipNRows	Skip constraints of type =N=	0
Width	Max line width of output format Range: {40, ..., ∞}	80

5.10 COPT

Cardinal Optimizer by Cardinal Operations is a solver for mixed-integer linear and convex quadratic programming problems.

5.10.1 Usage

A GAMS/COPT or GAMS/COPT-Link license is required to use the GAMS/COPT interface that is distributed with GAMS. If only a GAMS/COPT-Link license is available, also a separate COPT license needs to be installed on the system, see the "Cardinal Operations User Guide" for details.

The following statement can be used inside your GAMS program to specify using COPT

```
Option MIP = COPT;     { or LP, QCP, MIQCP, ... }
```

The above statement should appear before the Solve statement. If COPT was specified as the default solver during GAMS installation, the above statement is not necessary.

COPT supports special ordered sets (SOS), but no semicontinuous or semiinteger variables.

5.10.1.1 Specification of COPT Options

The following GAMS parameters are currently supported by GAMS/COPT: reslim, iterlim, nodlim, optcr, optca, tryint, and threads.

Further options can be set via a solver options file, see Section The Solver Option File for further information. Following is an example options file `copt.opt`:

```
TreeCutLevel 0
SimplexThreads 4
```

It will cause COPT to use cut generators only in the root node (for a MI(QC)P solve) and specifies to use 4 threads in the simplex algorithm.

5.10.1.2 Specification of Indicators

Indicators are a modeling tool to specify that certain equations in a model must only be satisfied if certain binary variables take a specified value. Indicators are not supported by the GAMS language, but can be passed to COPT via a GAMS/COPT solver options file, see Indicator Constraints for more details on its syntax.

5.10.1.3 MIP Starting Point

When solving a MIP, by default all values that are specified as variable levels in the GAMS model are passed as starting point to COPT (unless MipStartMode = 0). However, if GAMS option tryint is set to a nonzero value or MipStartMode = 2, then only values of binary and integer variables which fractionality is at most the value tryint are passed as starting point to COPT (possible fractional values are rounded to integers). COPT will then try to complete this partial solution to a feasible solution.

5.10.1.4 Solution Pool

When COPT solves a problem, it may find several solutions, whereof only the best one is available to the GAMS user via the variable level values in the GAMS model. If the option solnpool is specified, then all alternative solutions found by COPT are written into GDX files and an index file with the name given by the this option is written. If the option solnpoolmerge is specified, then all alternative solutions found by COPT are written into a single GDX file, which name is given by the this option.

The GAMS testlib model **dumpsol** shows an example use for this option.

5.10.1.5 Infeasible and unbounded LPs

If an LP is found infeasible by COPT, a primal infeasibility certificate in form of a Farkas proof (see also Mosek manual) is computed. The GAMS/COPT link returns the values of this certificate in the equations marginal values and sets the `INFES` markers (see solution listing) for those equations that are included in the Farkas proof.

If an LP is found unbounded by COPT, a dual infeasibility certificate in form of a primal ray is computed. If the problem is feasible, then the primal ray gives a direction in which the objective function can be infinitely improved. The GAMS/COPT link returns the values of the primal ray in the variable level values and sets `UNBND` markers (see solution listing) for those variables that are included in the ray.

COPT option ReqFarkasRay can be used to disable the calculation of these infeasibility certificates.

5.10.1.6 Finding an Irreducible Inconsistent Subsystem (IIS) of Constraints

If an LP or MIP is infeasible, COPT has the capability to identify a subset of the constraints that are infeasible and become feasible once any one of them is eliminated. Option iis can be used to enable finding an IIS.

As an example, consider a GAMS model containing

```
Positive Variables x, y;
Equation e1;
e1.. x + y =l= -1;
```

The corresponding COPT output when iis has been set to 1 will look like

```
Starting the simplex solver using 1 thread

Method   Iteration           Objective  Primal.NInf   Dual.NInf        Time
Dual             0    0.0000000000e+00            1           0       0.00s

Solving finished
Status: Infeasible  Objective: -  Iterations: 0  Time: 0.00s
Start the IIS computation for an LP

 Iteration  Min RowBnd  Max RowBnd  Min ColBnd  Max ColBnd      Time
         0           0           1           0           2      0.00s
         1           0           1           0           2      0.00s
         2           1           1           0           2      0.00s
         3           1           1           1           2      0.00s
         4           1           1           2           2      0.00s

IIS summary: 1 rows, 2 bounds of columns
IIS computation finished (0.000s)
IIS is minimal
Number of equations in IIS: 1
  Upper: e1 <= -1
Number of variables in IIS: 2
  Lower: x >= 0
  Lower: y >= 0
```

That is, COPT finds the problem infeasible. Then the IIS computation is started and an IIS is found. The IIS is then printed to the log and listing file. It consists of the lower bounds of variables `x` and `y` and equation `e1`. This suggests that the entire model can be made feasible by lowering the lower bound of `x` or `y` or by increasing the right-hand side of `e1`.

If one knows in advance that the problem is infeasible, then option iis can be set to 2 to skip the initial solve.

5.10.1.7 Using the Feasibility Relaxation

The feasibility relaxation is enabled by the FeasRelax parameter in a COPT solver option file.

With the FeasRelax option COPT selectively relaxes the variable bounds and constraint sides of an infeasible model instance in a way that a weighted penalty function is minimized. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

Attention

> At the moment, COPT can only relax sides of linear constraints. Indicator and quadratic constraints are not considered for relaxation.

By default all equation sides are candidates for relaxation and weighted equally but none of the variable bounds can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the dot option feaspref. A zero preference means that the associated bound or constraint side is not to be modified. The weighted penalty function is constructed from these preferences. The larger the preference, the more likely it will be that a given variable bound or constraint side will be relaxed. However, it is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a COPT solver option file. The syntax is:

(variable or equation) `.feaspref` *(value)*

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the `copt.opt` file can be specified by:

```
feasrelax 1
v.feaspref            1
v.feaspref('i1',*)    2
v.feaspref('i1','j2') 0

e.feaspref(*,'j1')    0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxation on. Furthermore, we specify that all variables `v(i,j)` have preference of 1, except variables over set element `i1`, which have a preference of 2. The variable over set element `i1` and `j2` has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords `variables` or `equations` instead of the individual variable and equations names (e.g. `variables.feaspref 1`).

The parameter FeasRelaxMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasRelaxMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations). Also check model **feasopt1** in the GAMS model library.

5.10.1.8 Parameter Tuning Tool

The COPT tuning tool performs multiple solves on a model instance, choosing different parameter settings for each run, in a search for settings that improve runtime or other quality measures. The longer it is run, the more likely it is to find a significant improvement. For notes on the limitation of parameter tuning tools, see also the GAMS/Gurobi manual.

A number of tuning-related parameters allow to control the operation of the tuning tool. To enable tuning, option Tuning needs to be set. The value of `Tuning` will be used as a prefix for the name of option files written by the solver link. The amount of time spend for tuning can be controlled by option TuneTimeLimit. Options TuneMode and TuneMeasure allow to specify whether solving time or bounds on the optimal value should be used as quality measure for the tuning time.

Options that are set in a GAMS/COPT options file are considered as fixed parameters and are not changed by the tuning tool. Since the GAMS/COPT link changes the default for a few COPT options, please check the log for the fixed parameters.

Using option TuneParams, a COPT options file with parameters that should be tuned can be specified. Note that this parameter file is read by COPT directly, so that its syntax may deviate from the one of GAMS/COPT solver option files. The file allows multiple values for each parameter name. For example,

if the file contains the lines `RootCutLevel 0 1 2 3` and `TreeCutLevel 0 1 2 3`, then the tuner will optimize only the amount of cutting plane generation for the given instance. If `TuneParams` is not set, the tuner will generate tuning parameter sets automatically.

If the tuning tool found one or several parameter sets that improve performance, then these will be written into files named <`Tuning`>.<`idx`>, where `Tuning` is the value of option Tuning and `idx` the index of the parameter set. The parameter set that yields best performance has index `000`. GAMS/COPT does not return a solution if the tuning tool is used. Tuning cannot be used at the same time as other advanced features, like feasibility relaxation.

5.10.2 List of COPT Options

In the following, we give a detailed list of available COPT options.

5.10.2.1 Limits and Tolerances

Option	Description	Default
AbsGap	The absolute gap for MIP Range: [0, ∞]	GAMS optca
BarIterLimit	Barrier iteration limit Range: {0, ..., ∞}	GAMS iterlim
DualTol	The tolerance for dual solutions and reduced cost Range: [1e-09, 0.0001]	1e-06
FeasTol	The feasibility tolerance Range: [1e-09, 0.0001]	1e-06
IntTol	The integer feasibility tolerance Range: [1e-09, 0.1]	1e-06
MatrixTol	The input matrix coefficient tolerance Range: [0, 1e-07]	1e-10
NodeLimit	Limit of nodes for MIP Range: {-1, ..., ∞}	GAMS nodlim
RelGap	The relative gap for MIP Range: [0, ∞]	GAMS optcr
SolTimeLimit	Time limit that is only effictive after a primal feasible solution has been found Range: [0, 1e+20]	1e+20
TimeLimit	Time limit of the optimization Range: [0, 1e+20]	GAMS reslim

5.10.2.2 Presolving and Scaling

Option	Description	Default
Dualize	Whether to dualize a problem before solving it -1: Choose automatically. 0: No dualizing. 1: Dualizing the problem.	-1

Option	Description	Default
Presolve	Level of presolving performed before solving a problem -1: Choose automatically. 0: No presolving. 1: Fast presolving. 2: Normal presolving. 3: Aggressive presolving.	-1
Scaling	Whether to perform scaling before solving a problem -1: Choose automatically. 0: No scaling. 1: Apply scaling.	-1

5.10.2.3 LP solving

Option	Description	Default
BarHomogeneous	Whether to use homogeneous self-dual form in barrier -1: Choose automatically. 0: No. 1: Yes.	-1
BarOrder	Ordering method for barrier -1: Choose automatically. 0: Approximate Minimum Degree (AMD). 1: Nested Dissection (ND).	-1
BarStart	Starting Point method for barrier -1: Choose automatically. 0: Simple. 1: Mehrotra. 2: Modified Mehrotra.	-1
Crossover	Whether to run crossover after barrier -1: Choose automatically. Only run crossover when the LP solution is not primal-dual feasible. 0: No. 1: Yes.	1
DualPerturb	Whether to allow the objective function perturbation -1: Choose automatically. 0: No perturbation. 1: Allow objective function perturbation.	-1
DualPrice	Specifies the dual simplex pricing algorithm -1: Choose automatically. 0: Using Devex pricing algorithm. 1: Using dual steepest-edge pricing algorithm .	-1
LpMethod	Specifies the LP method -1: Dual Simplex for LP. Automatic choice between Dual Simplex and Barrier for MIP. 1: Dual simplex. 2: Barrier. 3: Crossover. 4: Concurrent (Use simplex and barrier simultaneously). 5: Choose automatically.	-1
ReqFarkasRay	Whether to return a Farkas or Ray when an LP is infeasible or unbounded Range: boolean	1

5.10.2.4 MIP solving

Option	Description	Default
ConflictAnalysis	Whether to perform conflict analysis -1: Automatic. 0: No. 1: Yes.	-1
CutLevel	Level of cutting planes generation -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1
DivingHeurLevel	Level of diving heuristics -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1
FAPHeurLevel	Level of fix-and-propagate heuristics -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1
HeurLevel	Level of heuristics -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1
MipStartMode	Specifies MIP start mode -1: Automatic. 0: Disable MIP starts. 1: Only use full and feasible MIP starts. 2: Try to complete partial feasible MIP starts by solving subMIPs.	-1, if GAMS tryint = 0, otherwise 2
MipStartNodeLimit	Limit of nodes for MIP start sub-MIP (to complete partial MIP starts) (-1: unlimited) Range: {-1, ..., ∞}	-1
NodeCutRounds	Maximum cut rounds in a local node Range: {-1, ..., ∞}	-1
RootCutLevel	Level of root cutting planes generation -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1
RootCutRounds	Maximum cut rounds in the root (-1: unlimited) Range: {-1, ..., ∞}	-1

Option	Description	Default
RoundingHeurLevel	Level of rounding heuristics -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1
StrongBranching	Level of strong branching -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1
SubMipHeurLevel	Level of sub-MIP heuristics -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1
TreeCutLevel	Level of tree cutting planes generation -1: Automatic. 0: Off. 1: Fast. 2: Normal. 3: Aggressive.	-1

5.10.2.5 Multithreading

Option	Description	Default
BarThreads	Number of threads to use in the barrier solver Range: {-1, ..., 128}	-1
CrossoverThreads	Number of threads to use in the crossover Range: {-1, ..., 128}	-1
MipTasks	Number of parallel tasks in MIP solving (-1: automatic) Range: {-1, ..., 256}	-1
SimplexThreads	Number of threads to use in the simplex solver Range: {-1, ..., 128}	-1
Threads	Number of threads to use (-1: automatic, 0: 1 thread) GAMS threads = 0 sets the default for COPT threads to -1, i.e., let COPT choose the number of processors to use. Range: {-1, ..., 128}	GAMS threads

5.10.2.6 Infeasibility Analysis

Option	Description	Default
.feaspref	preference to include variable bound or equation side into feasibility relaxation The higher the value, the more likely it will be that associated variable bounds or equation sides are relaxed when computing a feasible relaxation. If zero, then the associated variable bounds or equation sides are not to considered for relaxation. Range: [0, ∞)	1.0 for equations, 0.0 for variables
FeasRelax	whether to compute a feasible relaxation instead of solving the problem Range: boolean Synonyms: feasopt	0
FeasRelaxMode	Specifies the feasibility relaxation mode 0: Minimize sum of violations. 1: Optimize original objective function under minimal sum of violations. 2: Minimize number of violations. 3: Optimize original objective function under minimal number of violations. 4: Minimize sum of squared violations. 5: Optimize original objective function under minimal sum of squared violations. Synonyms: feasoptmode	0
iis	whether to compute an Irreducible Inconsistent Subsystem (IIS) if the problem is infeasible 0: no 1: compute IIS after solve if infeasible 2: compute IIS without solving original problem	0
IISMethod	Specifies the IIS method -1: Automatic. 0: Find smaller IIS. 1: Find IIS quickly.	-1

5.10.2.7 Parameter Tuning

Option	Description	Default
TuneMeasure	Method to aggregate multiple permutation solves into a single measure for tuner -1: Choose automatically. 0: Average value. 1: Maximum value.	-1
TuneMethod	Tuning method -1: Choose automatically. 0: Greedy search strategy. 1: Broader search strategy.	-1
TuneMode	Specifies tuning mode for tuner -1: Choose automatically. 0: Solving time. 1: Relative optimality gap. 2: Primal bound (objective function value of incumbent). 3: Dual bound (bound on optimal value).	-1

Option	Description	Default
TuneOutputLevel	Output level for tuner 0: No tuning log. 1: Summary of the improved parameters only. 2: Summary of each tuning attempt. 3: Detailed log of each tuning attempt.	2
TuneParams	COPT parameter file with options to be tuned For each parameter, multiple values can be specified. Range: string	
TunePermutes	Number of permuted solves for each parameter set by tuner (0: auto) Range: {0, ..., ∞}	0
TuneTargetRelGap	Target relative gap for tuning Range: [0, ∞]	0.0001
TuneTargetTime	Target elapsed time for tuning Range: [0, ∞]	0.01
TuneTimeLimit	Time limit of tuning (0: auto) Range: [0, 1e+20]	0
Tuning	If set, enables tuning tool. The option value should be the prefix for names of tuned option files that are written. Range: string	

5.10.2.8 GAMS/COPT link

Option	Description	Default
readparams	read COPT parameter file Range: string	
solnpool	Solution pool file name The name of a solutions index gdx file for writing alternate solutions found by COPT. The GDX file specified by this option will contain a set called index that contains the names of GDX files with the individual solutions. Range: string	
solnpoolmerge	Solution pool file name for merged solutions Range: string	
solvefinal	whether to solve the LP obtained from fixing discrete variables and variables in SOS after a MIP solve If marginals are not accessed after a MIP solve, it is advised to disable this option. Range: boolean	1
solvetrace	name of file for writing solving progress information during solve Range: string	
solvetracenodefreq	frequency in number of nodes for writing to solve trace file The node frequency does not work when COPT is run with multiple threads. Range: {0, ..., ∞}	100
solvetracetimefreq	frequency in seconds for writing to solve trace file Range: (0, ∞)	5
writebas	name of file to which to write advanced starting basis in basis format Range: string	
writebin	name of file to which to write problem in COPT binary format Range: string	
writelp	name of file to which to write problem in LP file format Range: string	

Option	Description	Default
writemps	name of file to which to write problem in MPS file format Range: string	
writemst	name of file to which to write MIP starting point Range: string	

5.11 CPLEX

5.11.1 Introduction

GAMS/Cplex is a GAMS solver that allows users to combine the high level modeling capabilities of GAMS with the power of Cplex optimizers. Cplex optimizers are designed to solve large, difficult problems quickly and with minimal user intervention. Access is provided (subject to proper licensing) to Cplex solution algorithms for linear, quadratically constrained and mixed integer programming problems. While numerous solving options are available, GAMS/Cplex automatically calculates and sets most options at the best values for specific problems.

All Cplex options available through GAMS/Cplex are summarized at the end of this document.

5.11.2 How to Run a Model with Cplex

The following statement can be used inside your GAMS program to specify using Cplex

```
Option LP = Cplex;     { or QCP, MIP, MIQCP, RMIP or RMIQCP }
```

The above statement should appear before the Solve statement. If Cplex was specified as the default solver during GAMS installation, the above statement is not necessary.

Attention

The free bare-bone link mode (previously GAMS/OSICPLEX) that allowed to solve LP and MIP when the user had a separate CPLEX license has been removed. If you relied on using this bare-bone link option, then do not hesitate to contact sales@gams.com to arrange for a GAMS/CPLEX Solver Link license.

5.11.3 Overview of Cplex

5.11.3.1 Linear Programming

Cplex solves LP problems using several alternative algorithms. The majority of LP problems solve best using Cplex's state of the art dual simplex algorithm. Certain types of problems benefit from using the primal simplex algorithm, the network optimizer, the barrier algorithm, or the sifting algorithm. The concurrent option will allow solving with different algorithms in parallel. The solution is returned by the first to finish.

Solving linear programming problems is memory intensive. Even though Cplex manages memory very efficiently, insufficient physical memory is one of the most common problems when running large LPs. When memory is limited, Cplex will automatically make adjustments which may negatively impact performance. If you are working with large models, study the section entitled Physical Memory Limitations carefully.

Cplex is designed to solve the majority of LP problems using default option settings. These settings usually provide the best overall problem optimization speed and reliability. However, there are occasionally reasons for changing option settings to improve performance, avoid numerical difficulties, control optimization run duration, or control output options.

Some problems solve faster with the primal simplex algorithm rather than the default dual simplex algorithm. Very few problems exhibit poor numerical performance in both the primal and the dual. Therefore, consider trying primal simplex if numerical problems occur while using dual simplex.

Cplex has a very efficient algorithm for network models. Network constraints have the following property:

- each non-zero coefficient is either a +1 or a -1
- each column appearing in these constraints has > exactly 2 nonzero entries, one with a +1 > coefficient and one with a -1 coefficient

Cplex can also automatically extract networks that do not adhere to the above conventions as long as they can be transformed to have those properties.

The barrier algorithm is an alternative to the simplex method for solving linear programs. It employs a primal-dual logarithmic barrier algorithm which generates a sequence of strictly positive primal and dual solutions. Specifying the barrier algorithm may be advantageous for large, sparse problems.

Cplex provides a sifting algorithm which can be effective on problems with many more variables than equations. Sifting solves a sequence of LP subproblems where the results from one subproblem are used to select columns from the original model for inclusion in the next subproblem.

GAMS/Cplex also provides access to the Cplex Conflict Refiner previously known as IIS (Irreducibly Inconsistent Set). The conflict refinder takes an infeasible program and produces an set of conflicting constraints. Such a set consists of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Cplex reports the conflict in terms of GAMS equation and variable names and includes the conflict report as part of the normal solution listing.

5.11.3.2 Quadratically Constrained Programming

Cplex can solve models with quadratic constraints. These are formulated in GAMS as models of type QCP. QCP models are solved with the Cplex Barrier method.

QP models are a special case that can be reformulated to have a quadratic objective function and only linear constraints. Those are automatically reformulated from GAMS QCP models. When such problems are convex, Cplex normally solves them efficiently in polynomial time. Nonconvex QPs, however, are known to be quite hard. Cplex applies various approaches to those problems, such approaches as barrier algorithms or branch and bound algorithms. Notably, in the branch and bound approach, there is no theoretical guarantee about the complexity of such a problem. Consequently, solution of such a problem (that is, a nonconvex QP) can take many orders of magnitude longer than the solution of a convex QP of comparable dimensions. Therefore, the default is to reject such model. The parameter OptimalityTarget allows to change this behavior.

For QCP models Cplex returns a primal and in most cases also dual values.

5.11.3.3 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with integer variables, Cplex uses a branch and cut algorithm which solves a series of LP, subproblems. Because a single mixed integer problem generates many subproblems,even small mixed integer problems can be very compute intensive and require significant amounts of physical memory.

GAMS and GAMS/Cplex support Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

Cplex can also solve problems of GAMS model type MIQCP. As in the continuous case, if the base model is a QP the Simplex methods can be used and duals will be available at the solution. If the base model is a QCP, only the Barrier method can be used for the nodes and only primal values will be available at the solution.

5.11.3.4 Feasible Relaxation

The Conflict Refiner identifies the causes of infeasibility by means of inconsistent set of constraints. However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Cplex is known as FeasOpt (for Feasible Optimization) and turned on by parameter FeasOpt in a CPLEX option file. More details can be found in the section entitled Using the Feasibility Relaxation.

5.11.3.5 Solution Pool: Generating and Keeping Multiple Solutions

This section introduces the *solution pool* for storing multiple solutions to a mixed integer programming problem (MIP and MIQCP). The chapter also explains techniques for generating and managing those solutions.

The solution pool stores multiple solutions to a mixed integer programming (MIP and MIQCP) model. With this feature, you can direct the algorithm to generate multiple solutions in addition to the optimal solution. For example, some constraints may be difficult to formulate efficiently as linear expressions, or the objective may be difficult to quantify exactly. In such cases, obtaining multiple solutions will help you choose one which best fits all your criteria,including the criteria that could not be expressed easily in a conventional MIP or MIQCP model. For example,

- You can collect solutions within a given percentage of the optimal solution. To do so, apply the solution pool gap parameters SolnPoolAGap and SolnPoolGap.
- You can collect a set of diverse solutions. To do so, use the solution pool replacement parameter SolnPoolReplace to set the solution pool replacement strategy to 2. In order to control the diversity of solutions even more finely, apply a *diversity filter*.
- In an advanced application of this feature, you can collect solutions with specific properties. To do so, see the use of the *incumbent filter*.
- You can collect all solutions or all optimal solutions to model. To do so, set the solution pool intensity parameter SolnPoolIntensity to its highest value.

Please note, that the value for *best possible* can exceed the optimal solution value if CPLEX has already solved the model to optimality but continues to search for additional solutions.

Filling the Solution Pool

There are two ways to fill the solution pool associated with a model: You can *accumulate* successive incumbents or generate alternative solutions by *populating* the solution pool. The method is selected with the parameter SolnPoolPop:

- The regular optimization procedure automatically adds incumbents to the solution pool as they are discovered (SolnPoolPop = 1).
- Cplex also provides a procedure specifically to generate multiple solutions. You can invoke this procedure by setting option SolnPoolPop = 2. You can also invoke this procedure many times in a row in order to explore the solution space differently. In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory. This is done by specifying a GAMS program (option SolnPoolPopRepeat) that inspects the solutions. In case this GAMS program terminates normally, i.e. no execution or compilation error, the exploration for alternative solutions proceeds.

The option SolnPoolReplace designates the strategy for replacing a solution in the solution pool when the solution pool has reached its capacity. The value 0 replaces solutions according to a first-in, first-out policy. The value 1 keeps the solutions with the best objective values. The value 2 replaces solutions in order to build a set of diverse solutions.

If the solutions you obtain are too similar to each other, try setting SolnPoolReplace to 2.

The replacement strategy applies only to the subset of solutions created in the current call of populate. Solutions already in the pool are not affected by the replacement strategy. They will not be replaced, even if they satisfy the criterion of the replacement strategy. So with every repeated call of the populate procedure the solution pool will be extended by the newly found solution. After the GAMS program specified in SolnPoolPopRepeat determined to continue the search for alternative solutions, the file specified by option SolnPoolPopDel option is read in. The solution numbers present in this file will be delete from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

Enumerating All Solutions

With the solution pool, you can collect all solutions to a model. To do so, set the solution pool intensity parameter SolnPoolIntensity to its highest value, 4 and set SolnPoolPop = 2.

You can also enumerate all solutions that are valid for a specific criterion. For example, if you want to enumerate all alternative optimal solutions, do the following:

- Set the pool absolute gap parameter SolnPoolAGap = 0.0 (for models with potential numerical issues, better use a small non-zero like 1e-6).
- Set the pool intensity parameter SolnPoolIntensity = 4.
- Set the populate limit parameter PopulateLim to a value sufficiently large for your model; for example, 2100000000.
- Set the pool population parameter SolnPoolPop = 2.

Beware, however, that, even for small models, the number of possible solutions is likely to be huge. Consequently, enumerating all of them will take time and consume a large quantity of memory.

There may be an infinite number of possible values for a continuous variable, and it is not practical to enumerate all of them on a finite-precision computer. Therefore, populate gives only one solution for each set of binary and integer variables, even though there may exist several solutions that have the same values for all binary and integer variables but different values for continuous variables.

Likewise, for the same reason, the populate procedure does not generate all possible solutions for unbounded models. As soon as the proof of unboundedness is obtained, the populate procedure stops.

Cplex uses numerical methods of finite-precision arithmetic. Consequently, the feasibility of a solution depends on the value given to tolerances. Two parameters define the tolerances that assess the feasibility of a solution:

- the integrality tolerance EpInt
- the feasibility tolerance EpRHS

A solution may be considered feasible for one pair of values for these two parameters, and infeasible for a different pair. This phenomenon is especially noticeable in models with numeric difficulties, for example, in models with BigM coefficients.

Since the definition of a feasible solution is subject to tolerances, the total number of solutions to a model may vary, depending on the approach used to enumerate solutions, and on precisely which tolerances are used. In most models, this tolerance issue is not problematic. But, in the presence of numeric difficulties, Cplex may create solutions that are slightly infeasible or integer infeasible, and therefore create more solutions than expected.

Filtering the Solution Pool

Filtering allows you to control properties of the solutions generated and stored in the solution pool. Cplex provides two predefined ways to filter solutions.

If you want to filter solutions based on their difference as compared to a reference solution, use a *diversity filter*. This filter is practical for most purposes. However, if you require finer control of which solutions to keep and which to eliminate, use the *incumbent filter*.

Diversity Filter

A diversity filter allows you to generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables using dot option DivFlt and lower and upper bounds DivFltLo and DivFltUp. In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution. If you need more than one diversity filter, for example, to generate solutions that share the characteristics of several different solutions, additional filters can be specified through a Cplex Filter File using parameter ReadFLT. Details can be found in the example model `solnpool` in the GAMS model library.

Incumbent Filter

If you need to enforce more complex constraints on solutions (e.g. if you need to enforce nonlinear constraints), you can use the incumbent filtering. The incumbent checking routine is part of the GAMS BCH Facility. It will accept or reject incumbents independent of a solution pool. During the populate or regular optimize procedure, the incumbent checking routine specified by the parameter UserIncbCall is called each time a new solution is found, even if the new solution does not improve the objective value of the incumbent. The incumbent filter allows your application to accept or reject the new solution based on your own criteria. If the GAMS program specified by UserIncbCall terminates normally, the solution is rejected. If this program returns with a compilation or execution error, the incumbent is accepted.

Accessing the Solution Pool

The solutions are stored in GAMS Data eXchange (GDX) file and can be loaded by your GAMS program. Details can be found in the model `solnpool` in the GAMS model library and in . If you instruct Cplex to generate thousands of solution this becomes inefficient. The option SolnPoolMerge triggers the creation of a single GDX file containing all solutions.

The GAMS/Cplex link produces, if properly instructed, a GAMS Data eXchange(GDX) file with name specified in SolnPool that contains a set `Index` with elements `file1`, `file2`, ... The associated text of these elements contain the file names of the individual GDX solution file. The name is constructed using the prefix `soln` (which can be specified differently by option SolnPoolPrefix), the name of the model and a sequence number. For example `soln_loc_p1.gdx`. GAMS/Cplex will overwrite existing GDX files without warning. The set `Index` allows us to conveniently walk through the different solutions in the solution pool. A complete model can be found in the model `solnpool` in the GAMS model library.

```
...
solve mymodel min z using mip;

set soln           possible solutions in the solution pool /file1*file1000/
    solnpool(soln) actual solutions;
file fsol;

execute_load 'solnpool.gdx', solnpool=Index;
loop(solnpool(soln),
  put_utility fsol 'gdxin' / solnpool.te(soln):0:0;
  execute_loadpoint;
  display z.l;
);
```

If you instruct Cplex to generate thousands of solution this method becomes inefficient. The option SolnPoolMerge triggers the creation of a single GDX file containing all solutions. Details on usage of this option can be found in the model `solmpool` in the GAMS model library.

5.11.3.6 Benders Algorithm

CPLEX implements Benders algorithm.

Given a formulation of a problem, CPLEX can decompose the model into a single master and (possibly multiple) subproblems. To do so, CPLEX makes use of annotations that you supply for your model. This approach can be applied to mixed-integer linear programs (MILP). For certain types of problems, this approach can offer significant performance improvements.

The parameter, BendersStrategy, specifies to CPLEX how you want to apply Benders algorithm as a strategy to solve your model. By default, if you did not annotate your model to specify a decomposition, CPLEX executes conventional branch and bound. If you annotated your model, CPLEX attempts to refine your decomposition and applies Benders algorithm. With this parameter, you can direct CPLEX to decompose your model and to apply its implementation of Benders algorithm in one of these alternative ways:

- 1: CPLEX attempts to decompose your model strictly according to your annotations.
- 2: CPLEX decomposes your model by using your annotations as hints and refining the decomposition where it can. CPLEX initially decomposes your model according to your annotation and then attempts to refine that decomposition by further decomposing the specified subproblems. This approach can be useful if you annotate certain variables to go into master, and all others to go into a single subproblem, which CPLEX can then decompose further for you.
- 3: CPLEX automatically decomposes your model, ignoring any annotations you may have supplied. CPLEX puts all integer variables into the master, puts all continuous variables into a subproblem and decomposes that subproblem, if possible.

If you want to specify a decomposition to CPLEX, you need to annotate your model and specify a Benders partition of your variable space. These Benders partition can be conveniently specified with the dot option BendersPartition or through the .stage variable suffix.

These Benders partition values specify to CPLEX whether certain variables belong to the master or to one of the subproblems assigned to workers (where the subproblems are numbered from 1 (one) to N, the number of subproblems). If you annotate a given variable with the value 0 (zero), CPLEX assigns that variable to the master. If you annotate a given variable with the value i, where i is greater than or equal to 1 (one), CPLEX assigns that variable to subproblem i. If a variable is not specified, the default will be to go into the master problem. Note that with `variable.BendersPartition 1` you can assign all variables to the subproblem and then selectively assign the master variables with `varname.BendersPartition 0`.

If you want to communicate the Benders partition values via the .stage variable suffix, the partition numbers are off by one compared to the partition number via the dot option. So the master variables have stage 1, and the subproblems start with stage 2. If you want to leave a variable unassigned you can either make the stage 0 or fractional (e.g. 1.5). Discrete variables are automatically put into the master problem and don't need to be set. In addition to specifying the Benders partition values via the .stage variable suffix the link option BendersPartitionInStage needs to be set to 1.

CPLEX produces an error if the annotated decomposition does not yield disjoint subproblems. For example, if your annotations specify that two (or more) variables belong in different subproblems, yet your model specifies that these variables participate in the same constraint, then these variables are linked. Consequently, the subproblems where these variables appear according to your annotations are not disjoint with respect to the partition.

5.11.3.7 Multiple Objectives

While typical optimization models have a single objective function, real-world optimization problems often have multiple, competing objectives. For example, in a production planning model, you may want to both maximize profits and minimize late orders, or in a workforce scheduling application, you may want to both minimize the number of shifts that are short-staffed while also respecting worker's shift preferences.

The main challenge you face when working with multiple, competing objectives is deciding how to manage the tradeoffs between them. Cplex provides tools that simplify the task: Cplex allows you to blend multiple objectives, to treat them *hierarchically*, or to combine the two approaches. In a blended approach, you optimize a weighted combination of the individual objectives. In a hierarchical or *lexicographic* approach, you set a priority for each objective, and optimize in priority order. When optimizing for one objective,

you only consider solutions that would not degrade the objective values of higher-priority objectives. Cplex allows you to enter and manage your objectives, to provide weights for a blended approach, or to set priorities for a hierarchical approach. Cplex will only solve multi-objective models with strictly linear objectives. Moreover, for continous models, Cplex will report a primal only solution (not dual information).

Following the workforce application the specifications of the objectives would be done as follows:

```
equations defObj, defNumShifts, defSumPreferences;
variables obj, numShifts, sumPreferences;

defobj..            obj =e= numShifts - 1/100*sumPreferences;
defNumShifts..      numShifts =e= ...;
defSumPreferences.. sumPreferences =e= ...;

model workforce /all/;
solve workforce minimizing obj using mip;
```

With the default setting Cplex will solve the blended objective. Using the parameter MultObj Cplex will use a hierarchical approach. A hierarchical or lexicographic approach assigns a priority to each objective, and optimizes for the objectives in decreasing priority order. At each step, it finds the best solution for the current objective, but only from among those that would not degrade the solution quality for higher-priority objectives. The priority is specified by the absolute value of the objective coefficient in the blended objective function (`defObj`). In the example, the `numShifts` objective with coefficient 1 has higher priority than the `sumPreferences` objective with absolute objective coefficient 1/100. The sign of the objective coefficient determines the direction of the particular objective function. So here `numShifts` will be minimized (same direction as on the `solve` statement) while `sumPreferences` will be maximized. GAMS needs to identify the various objective functions, therefore the objective variables can only appear in the blended objective functions and in the particular objective defining equation.

By default, the hierarchical approach won't allow later objectives to degrade earlier objectives. This behavior can be relaxed through a pair of attributes: ObjNRelTol and ObjNAbsTol. Their meaning differs for continuous and discrete models:

- Discrete models: By setting one of the two tolerances for a particular objective, you can indicate that later objectives are allowed to degrade this objective by the specified relative or absolute amount, respectively. In our earlier example, if the optimal value for `numShifts` is 100, and if we set ObjNAbsTol for this objective to 20, then the second optimization step maximizing `sumPreferences` would find the best solution for the second objective from among all solutions with objective 120 or better for `numShifts`. Note that if you modify both tolerances, later optimizations would use the looser of the two values (i.e., the one that allows the larger degradation).
- Continuous Models: ObjNAbsTol does not include a hard constraint for previous objective functions in susequent solves. Instead, ObjNAbsTol defines a threshold for the reduced costs above which nonbasic variables in the associated LP solve will be fixed at the bound at which they reside. The option ObjNRelTol has no effect here. In order to apply the above discrete strategy for continuous models, set MultObjTolMip to `1`.

5.11.4 GAMS Options

The following GAMS options are used by GAMS/Cplex:

- **Option BRatio = x;**

Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Cplex not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed.

- **Option IterLim = n;**

  Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS.

  Cplex handles the iteration limit for MIP problems differently than some other GAMS solvers. The iteration limit is applied per node instead of as a total over all nodes. For MIP problems, controlling the length of the solution run by limiting the execution time (ResLim) is preferable.

  Simlarly, when using the sifting algorithm, the iteration limit is applied per sifting iteration (ie per LP). The number of sifting iterations (LPs) can be limited by setting Cplex parameter SiftItLim. It is the number of sifting iterations that is reported back to GAMS as iterations used.

- **Option ResLim = x;**

  Sets the time limit in seconds. In case resLim assumes its default value (1e+10) Cplex will use its own default (1e+75). The algorithm will terminate and pass on the current solution to GAMS.

- **Option SysOut = On;**

  Will echo Cplex messages to the GAMS listing file. This option may be useful in case of a solver failure.

- ***ModelName*.Cheat = x;**

  Cheat value: each new integer solution must be at least x better than the previous one. Can speed up the search, but you may miss the optimal solution. The cheat parameter is specified in absolute terms (like the OptCA option). The Cplex option ObjDif overrides the GAMS cheat parameter.

- ***ModelName*.Cutoff = x;**

  Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm.

- ***ModelName*.NodLim = x;**

  Maximum number of nodes to process for a MIP problem.

- **Option OptCA = x;**

  Absolute optimality criterion for a MIP problem. The OptCA option asks Cplex to stop when

  $$|BP - BF| < \text{OptCA}$$

  where BF is the objective function value of the current best integer solution while BP is the best possible integer solution.

  Note: This option also influences the SubMIPs (e.g., used for the RINS heuristic) and can thus influence the solution path.

- ***ModelName*.OptCR = x;**

  Relative optimality criterion for a MIP problem. Notice that Cplex uses a different definition than GAMS normally uses. The OptCR option asks Cplex to stop when

  $$(|BP - BF|)/(1.0e - 10 + |BF|) < \text{OptCR}$$

  where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. The GAMS definition is:

$$(|BP - BF|)/(|BP|) < \text{OptCR}$$

Note: This option also influences the SubMIPs (e.g., used for the RINS heuristic) and can thus influence the solution path.

- ***ModelName*.OptFile= 1;**

  Instructs Cplex to read the option file. The name of the option file is `cplex.opt`.

- ***ModelName*.PriorOpt= 1;**

  Instructs Cplex to use priority branching information passed by GAMS through the `variable.prior` parameters.

- ***ModelName*.TryInt= x;**

  Causes GAMS/Cplex to make use of current variable values when solving a MIP problem. If a variable value is within x of a bound, it will be moved to the bound and the preferred branching direction for that variable will be set toward the bound. The preferred branching direction will only be effective when priorities are used. Priorities and tryint are sometimes not very effective and often outperformed by GAMS/CPLEX default settings. Supporting GAMS/CPLEX with knowledge about a known solution can be passed on by different means, please read more about this in section entitled Starting from a MIP Solution.

GAMS/Cplex also sets many model attributes that can by used in your GAMS program by accessing *ModelName.suffix*. A list of model attributes available after the solve can be found here Model Attributes Mainly Used After Solve.

Cplex has the concept of deterministic time i.e. a measure of time that respects the same solution path to arrive at the same values in the solution while it yields the same level of performance for repeated solving of the same model with the same parameter settings, on the same computing platform. The length of a deterministic time tick may vary by platform. Nevertheless, ticks are normally consistent measures for a given platform (combination of hardware and software) carrying the same load. In other words, the correspondence of ticks to clock time depends on the hardware, software, and the current load of the machine. For the same platform and same load, the ratio of ticks per second stays roughly constant, independent of the model solved. However, for very short optimization runs, the variation of this ratio is typically high. GAMS/Cplex reports the deterministic time ticks spend inside Cplex in the model attribute `ETAlg`. Normally `ETAlg` is reporting the seconds.

5.11.5 Summary of CPLEX Options

The various Cplex options are listed here by category, with a few words about each to indicate its function. The options are listed again, in alphabetical order and with detailed descriptions, in the last section of this document.

5.11.5.1 Preprocessing and General Options

Option	Description	Default
advind	advanced basis use	`determined by GAMS Bratio`
aggfill	aggregator fill parameter	`10`
aggind	aggregator on/off	`-1`
bndrng	do lower / upper bound ranging	
calcqcpduals	calculate the dual values of a quadratically constrained problem	`1`

Option	Description	Default
clocktype	clock type for computation time	2
coeredind	coefficient reduction on/off	-1
cpumask	switch and mask to bind threads to processors (Linux only)	auto
datacheck	controls data consistency checking and modeling assistance	0
depind	dependency checker on/off	-1
dettilim	deterministic time limit	1.0e+75
feasopt	computes a minimum-cost relaxation to make an infeasible model feasible	0
feasoptmode	mode of FeasOpt	0
.feaspref	feasibility preference	1
fixoptfile	name of option file which is read just before solving the fixed problem	
folding	LP folding will be attempted during the preprocessing phase	-1
freegamsmodel	preserves memory by dumping the GAMS model instance representation temporarily to disk	0
iafile	secondary option file to be read in interactive mode triggered by iatriggerfile	
iatriggerfile	file that triggers the reading of a secondary option file in interactive mode	
iatriggertime	time interval in seconds the link looks for the trigger file in interactive mode	60
indicoptstrict	abort in case of an error in indicator constraint in solver option file	1
interactive	allow interactive option setting after a Control-C	0
lpmethod	algorithm to be used for LP problems	0
memoryemphasis	reduces use of memory	0
multobj	controls the hierarchical optimization of multiple objectives	0
multobjdisplay	level of display during multiobjective optimization	1
multobjmethod	method used for multi-objective solves	0
multobjoptfiles	List of option files used for individual solves within multi-objective optimization	
multobjtolmip	enables hard constraints for hierarchical optimization objectives based on degradation tolerances	1
names	load GAMS names into Cplex	1
numericalemphasis	emphasizes precision in numerically unstable or difficult problems	0
objnabstol	allowable absolute degradation for objective	

Option	Description	Default
objnreltol	allowable relative degradation for objective	
objrng	do objective ranging	no objective ranging is done
optimalitytarget	type of optimality that Cplex targets	0
parallelmode	parallel optimization mode	1
predual	give dual problem to the optimizer	0
preind	turn presolver on/off	1
prepass	number of presolve applications to perform	-1
prereform	set presolve reformulations	3
printoptions	list values of all options to GAMS listing file	0
qextractalg	quadratic extraction algorithm in GAMS interface	0
qpmethod	algorithm to be used for QP problems	0
qtolin	linearization of the quadratic terms in the objective function of a QP or MIQP model	-1
randomseed	sets the random seed differently for diversity of solutions	changes with each Cplex release
readparams	read Cplex parameter file	
reduce	primal and dual reduction type	3
relaxpreind	presolve for initial relaxation on/off	-1
rerun	rerun problem if presolve infeasible or unbounded	nono
rhsrng	do right-hand-side ranging	no right-hand-side ranging is done
rngrestart	write GAMS readable ranging information file	ranging information is printed to the listing file
scaind	matrix scaling on/off	0
solutiontype	type of solution (basic or non basic) for an LP or QP	0
threads	global default thread count	GAMS Threads
tilim	overrides the GAMS ResLim option	GAMS ResLim
tuning	invokes parameter tuning tool	
tuningdettilim	tuning deterministic time limit per model or suite	1.0e+75
tuningdisplay	level of information reported by the tuning tool	1
tuningmeasure	measure for evaluating progress for a suite of models	1
tuningrepeat	number of times tuning is to be repeated on perturbed versions	1
tuningtilim	tuning time limit per model or suite	0.2*GAMS ResLim

Option	Description	Default
warninglimit	determines how many times warnings of a specific type (datacheck=2) will be displayed	10
workdir	directory for working files	current or project directory
workmem	memory available for working storage	2048.0

5.11.5.2 Simplex Algorithmic Options

Option	Description	Default
conflictalg	algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model	0
conflictdisplay	decides how much information CPLEX reports when the conflict refiner is working	1
craind	crash strategy (used to obtain starting basis)	1
dpriind	dual simplex pricing	0
dynamicrows	switch for dynamic management of rows	-1
epper	perturbation constant	1.0e-06
iis	run the conflict refiner also known as IIS finder if the problem is infeasible	0
netfind	attempt network extraction	2
netppriind	network simplex pricing	0
perind	force initial perturbation	0
perlim	number of stalled iterations before perturbation	0
ppriind	primal simplex pricing	0
pricelim	pricing candidate list	0, in which case it is determined automatically
reinv	refactorization frequency	0, in which case it is determined automatically
sifting	switch for sifting from simplex optimization	1

5.11.5.3 Simplex Limit Options

Option	Description	Default
itlim	iteration limit	GAMS IterLim
netitlim	iteration limit for network simplex	large
objllim	objective function lower limit	-1.0e+75
objulim	objective function upper limit	1.0e+75
singlim	limit on singularity repairs	10

5.11.5.4 Simplex Tolerance Options

Option	Description	Default
epmrk	Markowitz pivot tolerance	0.01
epopt	optimality tolerance	1.0e-06
eprhs	feasibility tolerance	1.0e-06
ltol	basis identification primal tolerance	0
mtol	basis identification dual tolerance	0
netepopt	optimality tolerance for the network simplex method	1.0e-06
neteprhs	feasibility tolerance for the network simplex method	1.0e-06

5.11.5.5 Barrier Specific Options

Option	Description	Default
baralg	algorithm selection	0
barcolnz	dense column handling	0
barcrossalg	barrier crossover method	0
barepcomp	convergence tolerance	1.0e-08
bargrowth	unbounded face detection	1.0e+12
baritlim	iteration limit	large
barmaxcor	maximum correction limit	-1
barobjrng	maximum objective function	1.0e+20
barorder	row ordering algorithm selection	0
barqcpepcomp	convergence tolerance for the barrier optimizer for QCPs	1.0e-07
barstartalg	barrier starting point algorithm	1

5.11.5.6 Sifting Specific Options

Option	Description	Default
siftalg	sifting subproblem algorithm	0
siftitlim	limit on sifting iterations	large

5.11.5.7 MIP Algorithmic Options

Option	Description	Default
bbinterval	best bound interval	7

Option	Description	Default
.benderspartition	Benders partition	0
benderspartitioninstage	Benders partition through stage variable suffix	0
bendersstrategy	Benders decomposition algorithm as a strategy	0
bndstrenind	bound strengthening	-1
bqpcuts	boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality	0
brdir	set branching direction	0
bttol	backtracking limit	1.0
cardls	decides how often to apply the cardinality local search heuristic (CLSH)	-1
cliques	clique cut generation	0
covers	cover cut generation	0
cutlo	lower cutoff for tree search	-1.0e+75
cuts	default cut generation	0
cutsfactor	cut limit	-1.0
cutup	upper cutoff for tree search	1.0e+75
disjcuts	disjunctive cuts generation	0
divetype	MIP dive strategy	0
eachcutlim	sets a limit for each type of cut	2100000000
flowcovers	flow cover cut generation	0
flowpaths	flow path cut generation	0
fpheur	feasibility pump heuristic	0
fraccuts	Gomory fractional cut generation	0
gubcovers	GUB cover cut generation	0
heurfreq	heuristic frequency	0
heuristiceffort	the effort that CPLEX spends on heuristics during a MIP solve	1.0
implbd	implied bound cut generation	0
.lazy	Lazy constraints activation	0
lazyconstraints	Indicator to use lazy constraints	0
lbheur	local branching heuristic	0
liftprojcuts	lift-and-project cuts	0
localimplied	generation of locally valid implied bound cuts	0
lowerobjstop	in a minimization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value lower than or equal to the specified value	-1e75
mcfcuts	multi-commodity flow cut generation	0

Option	Description	Default
mipemphasis	MIP solution tactics	0
mipkappastats	MIP kappa computation	0
mipordind	priority list on/off	GAMS PriorOpt
mipordtype	priority order generation	0
mipsearch	search strategy for mixed integer programs	0
mipstart	use mip starting values	0
mipstopexpr	stopping expression for branch and bound	
miqcpstrat	MIQCP relaxation choice	0
mircuts	mixed integer rounding cut generation	0
multimipstart	use multiple mipstarts provided via gdx files	
nodecuts	decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree	0
nodefileind	node storage file indicator	1
nodesel	node selection strategy	1
preslvnd	node presolve selector	0
probe	perform probing before solving a MIP	0
qpmakepsdind	adjust MIQP formulation to make the quadratic matrix positive-semi-definite	1
relaxfixedinfeas	accept small infeasibilties in the solve of the fixed problem	0
repeatpresolve	reapply presolve at root after preprocessing	-1
rinsheur	relaxation induced neighborhood search frequency	0
rltcuts	Reformulation Linearization Technique (RLT) cuts	0
solvefinal	switch to solve the problem with fixed discrete variables	1
sos1reform	automatic logarithmic reformulation of special ordered sets of type 1 (SOS1)	0
sos2reform	automatic logarithmic reformulation of special ordered sets of type 2 (SOS2)	0
startalg	MIP starting algorithm	0
strongcandlim	size of the candidates list for strong branching	10
strongitlim	limit on iterations per branch for strong branching	0
subalg	algorithm for subproblems	0
submipnodelim	limit on number of nodes in an RINS subMIP	500
submipscale	scale the problem matrix when CPLEX solves a subMIP during MIP optimization	0
submipstartalg	starting algorithm for a subMIP of a MIP	0
submipsubalg	algorithm for subproblems of a subMIP of a MIP	0

Option	Description	Default
symmetry	symmetry breaking cuts	-1
upperobjstop	in a maximization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value greater than or equal to the specified value	1e75
.usercut	User cut activation	0
usercutpool	Indicator to use user cuts	0
varsel	variable selection strategy at each node	0
workeralgorithm	set method for optimizing benders subproblems	0
zerohalfcuts	zero-half cuts	0

5.11.5.8 MIP Limit Options

Option	Description	Default
aggcutlim	aggregation limit for cut generation	3
auxrootthreads	number of threads for auxiliary tasks at the root node	0
cutpass	maximum number of cutting plane passes	0
fraccand	candidate limit for generating Gomory fractional cuts	200
fracpass	maximum number of passes for generating Gomory fractional cuts	0
intsollim	maximum number of integer solutions	large
nodelim	maximum number of nodes to solve	GAMS NodLim
polishafterdettime	deterministic time before starting to polish a feasible solution	1.0e+75
polishafterepagap	absolute MIP gap before starting to polish a feasible solution	0.0
polishafterepgap	relative MIP gap before starting to polish a solution	0.0
polishafterintsol	MIP integer solutions to find before starting to polish a feasible solution	2147483647
polishafternode	nodes to process before starting to polish a feasible solution	2147483647
polishaftertime	time before starting to polish a feasible solution	1.0e+75
probedettime	deterministic time spent probing	1.0e+75
probetime	time spent probing	1.0e+75
repairtries	try to repair infeasible MIP start	0
trelim	maximum space in memory for tree	1.0e+75

5.11.5.9 MIP Solution Pool Options

Option	Description	Default
.divflt	solution pool range filter coefficients	0

Option	Description	Default
divfltlo	lower bound on diversity	mindouble
divfltup	upper bound on diversity	maxdouble
populatelim	limit of solutions generated for the solution pool by populate method	20
readflt	reads Cplex solution pool filter file	
solnpool	solution pool file name	
solnpoolagap	absolute tolerance for the solutions in the solution pool	1.0e+75
solnpoolcapacity	limits of solutions kept in the solution pool	2100000000
solnpoolgap	relative tolerance for the solutions in the solution pool	1.0e+75
solnpoolintensity	solution pool intensity for ability to produce multiple solutions	0
solnpoolmerge	solution pool file name for merged solutions	
solnpoolnumsym	maximum number of variable symbols when writing merged solutions	10
solnpoolpop	methods to populate the solution pool	1
solnpoolpopdel	file with solution numbers to delete from the solution pool	
solnpoolpoprepeat	method to decide if populating the solution should be repeated	
solnpoolprefix	file name prefix for GDX solution files	soln
solnpoolreplace	strategy for replacing a solution in the solution pool	0

5.11.5.10 MIP Tolerance Options

Option	Description	Default
bendersfeascuttol	Tolerance for whether a feasibility cut has been violated in Benders decomposition	1.0e-06
bendersoptcuttol	Tolerance for optimality cuts in Benders decomposition	1.0e-06
epagap	absolute stopping tolerance	GAMS OptCA
epgap	relative stopping tolerance	GAMS OptCR
epint	integrality tolerance	1.0e-05
eplin	degree of tolerance used in linearization	0.001
objdif	overrides GAMS Cheat parameter	0.0
relobjdif	relative cheat parameter	0.0

5.11.5.11 Output Options

Option	Description	Default
bardisplay	progress display level	1

Option	Description	Default
clonelog	enable clone logs	0
exactkappa	report exact condition number in quality report	0
mipdisplay	progress display level	4
mipinterval	progress display interval	0
miptrace	filename of MIP trace file	
mpslongnum	MPS file format precision of numeric output	1
netdisplay	network display level	2
quality	write solution quality statistics	0
siftdisplay	sifting display level	1
simdisplay	simplex display level	1
writeannotation	produce a Cplex annotation file	
writebas	produce a Cplex basis file	
writeflt	produce a Cplex solution pool filter file	
writelp	produce a Cplex LP file	
writemps	produce a Cplex MPS file	
writemst	produce a Cplex mst file	
writeord	produce a Cplex ord file	
writeparam	produce a Cplex parameter file with all active options	
writepre	produce a Cplex LP/MPS/SAV file of the presolved problem	
writeprob	produce a Cplex problem file and inferrs the type from the extension	
writesav	produce a Cplex binary problem file	

5.11.5.12 BCH Facility Options

Option	Description	Default
usercallparmfile	Command-line parameter include file used in GAMS command-line calls triggered by BCH	
usercutcall	the GAMS command line to call the cut generator	
usercutfirst	calls the cut generator for the first n nodes	10
usercutfreq	determines the frequency of the cut generator model calls	10
usercutinterval	determines the interval when to apply the multiplier for the frequency of the cut generator model calls	100
usercutmult	determines the multiplier for the frequency of the cut generator model calls	2
usercutnewint	calls the cut generator if the solver found a new integer feasible solution	1
usergdxin	the name of the GDX file read back into Cplex	bchin.gdx

Option	Description	Default
usergdxname	the name of the GDX file exported from the solver with the solution at the node	bchout.gdx
usergdxnameinc	the name of the GDX file exported from the solver with the incumbent solution	bchout_i.gdx
usergdxprefix	prefixes usergdxin, usergdxname, and usergdxnameinc	
usergdxsol	the name of the GDX file exported by Cplex to store the solution of extra columns	bchsol.gdx
userheurcall	the GAMS command line to call the heuristic	
userheurfirst	calls the heuristic for the first n nodes	10
userheurfreq	determines the frequency of the heuristic model calls	10
userheurinterval	determines the interval when to apply the multiplier for the frequency of the heuristic model calls	100
userheurmult	determines the multiplier for the frequency of the heuristic model calls	2
userheurnewint	calls the heuristic if the solver found a new integer feasible solution	1
userheurobjfirst	Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement	0
userincbcall	the GAMS command line to call the incumbent checking program	
userincbicall	the GAMS command line to call the incumbent reporting program	
userjobid	postfixes lf, o on call adds –userjobid to the call. Postfixes gdxname, gdxnameinc and gdxin	
userkeep	calls gamskeep instead of gams	0
userlazyconcall	the GAMS command line to call the lazy constraint generator	

5.11.5.13 The GAMS/Cplex Options File

The GAMS/Cplex options file consists of one option or comment per line. An asterisk (∗) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file `cplex.opt`.

```
scaind 1
simdisplay 2
```

It will cause Cplex to use a more aggressive scaling method than the default. The iteration log will have an entry for each iteration instead of an entry for each refactorization.

5.11.6 Special Notes

5.11.6.1 Physical Memory Limitations

For the sake of computational speed, Cplex should use only available physical memory rather than virtual or paged memory. When Cplex recognizes that a limited amount of memory is available it automatically makes algorithmic adjustments to compensate. These adjustments almost always reduce optimization

speed. Learning to recognize when these automatic adjustments occur can help to determine when additional memory should be added to the computer.

On virtual memory systems, if memory paging to disk is observed, a considerable performance penalty is incurred. Increasing available memory will speed the solution process dramatically. Also consider option MemoryEmphasis to conserve memory where possible.

When solving LPs, the Simplex algorithm (lpmethod 1 or 2) usually consumes less memory than the Barrier algorithm (lpmethod 2). During the Simplex algorithm, Cplex performs an operation called refactorization at a frequency determined by the ReInv option setting. The longer Cplex works between refactorizations, the greater the amount of memory required to complete each iteration. Therefore, one means for conserving memory is to increase the refactorization frequency. Since refactorization is an expensive operation, increasing the refactorization frequency by reducing the ReInv option setting generally will slow performance. Cplex will automatically increase the refactorization frequency if it encounters low memory availability. This can be seen by watching the iteration log. The default log reports problem status at every refactorization. If the number of iterations between iteration log entries is decreasing, Cplex is increasing the refactorization frequency. Since Cplex might increase the frequency to once per iteration, the impact on performance can be dramatic. Providing additional memory should be beneficial.

GAMS/Cplex also provides the option FreeGAMSModel to free some memory allocated by the GAMS link and making it available to Cplex. This only works when the GAMS parameter `solveLink` is set 0 which should be always done when memory is tight because GAMS completely vacates memory.

The Threads options also has a significant impact on memory consumption. The concurrent solvers keep multiple copies of the problem in memory which double or even triples the amount of memory consumed. If memory tight, set the Threads parameter to 1.

Not loading GAMS names into CPLEX can also help in saving some memory. This can be done by setting the model attribute dictFile or by using CPLEX option names.

5.11.6.2 Using Special Ordered Sets

For some models a special structure can be exploited. GAMS allows you to declare SOS1 and SOS2 variables (Special Ordered Sets of type 1 and 2).

In Cplex the definition for SOS1 variables is:

- A set of variables for which at most one variable may be non-zero.

The definition for SOS2 variables is:

- A set of variables for which at most two variables may be non-zero. If two variables are non-zero, they must be adjacent in the set.

5.11.6.3 Using Semi-Continuous and Semi-Integer Variables

GAMS allows the declaration of semi-continuous and semi-integer variables. These variable types are directly supported by GAMS/Cplex. For example:

```
SemiCont Variable x;
x.lo = 3.2;
x.up = 8.7;

SemiInt Variable y;
y.lo = 5;
y.up = 10;
```

Variable x will be allowed to take on a value of 0.0 or any value between 3.2 and 8.7. Variable y will be allowed to take on a value of 0 or any integral value between 5 and 10.

5.11.6.4 Running Out of Memory for MIP Problems

The most common difficulty when solving MIP problems is running out of memory. This problem arises when the branch and bound tree becomes so large that insufficient memory is available to solve an LP subproblem. As memory gets tight, you may observe frequent warning messages while Cplex attempts to navigate through various operations within limited memory. If a solution is not found shortly the solution process will be terminated with an unrecoverable integer failure message.

The tree information saved in memory can be substantial. Cplex saves a basis for every unexplored node. When utilizing the best bound method of node selection, the list of such nodes can become very long for large or difficult problems. How large the unexplored node list can become is entirely dependent on the actual amount of physical memory available and the actual size of the problem. Certainly increasing the amount of memory available extends the problem solving capability. Unfortunately, once a problem has failed because of insufficient memory, you can neither project how much further the process needed to go nor how much memory would be required to ultimately solve it.

Memory requirements can be limited by using the WorkMem, option with the NodeFileInd option. Setting NodeFileInd to 2 or 3 will cause Cplex to store portions of the branch and bound tree on disk whenever it grows to larger than the size specified by option WorkMem. That size should be set to something less than the amount of physical memory available.

Another approach is to modify the solution process to utilize less memory.

- Set option NodeSel to use a best estimate strategy or, more drastically a depth-first-search. Depth first search rarely generates a large unexplored node list since Cplex will be diving deep into the branch and bound tree rather than jumping around within it.
- Set option VarSel to use strong branching. Strong branching spends extra computation time at each node to choose a better branching variable. As a result it generates a smaller tree. It is often faster overall, as well.
- On some problems, a large number of cuts will be generated without a correspondingly large benefit in solution speed. Cut generation can be turned off using option Cuts.
- Use the Simplex algorithm instead of the Barrier algorithm using options StartAlg and SubAlg.
- Use option solvefinal to avoid solving the final LP.

See also Physical Memory Limitations.

5.11.6.5 Failing to Prove Integer Optimality

One frustrating aspect of the branch and bound technique for solving MIP problems is that the solution process can continue long after the best solution has been found. Remember that the branch and bound tree may be as large as 2^n nodes, where n equals the number of binary variables. A problem containing only 30 binary variables could produce a tree having over one billion nodes! If no other stopping criteria have been set, the process might continue ad infinitum until the search is complete or your computer's memory is exhausted.

In general you should set at least one limit on the optimization process before beginning an optimization. Setting limits ensures that an exhaustive tree search will terminate in reasonable time. Once terminated, you can rerun the problem using some different option settings.

5.11.6.6 Starting from a MIP Solution

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution. When you provide such a starting solution, you may invoke relaxation induced neighborhood search (RINS heuristic) or solution polishing to improve the given solution. This first integer solution may include continuous and discrete variables of various types, such as semi-continuous variables or special ordered sets.

If you specify values for all discrete variables, GAMS/CPLEX will check the validity of the values as an integer-feasible solution; if you specify values for only a portion of the discrete variables, GAMS/CPLEX will attempt to fill in the missing values in a way that leads to an integer-feasible solution. If the specified values do not lead directly to an integer-feasible solution, GAMS/CPLEX will apply a quick heuristic to try to repair the MIP Start. The number of times that GAMS/CPLEX applies the heuristic is controlled by the repair tries parameter (RepairTries). If this process succeeds, the solution will be treated as an integer solution of the current problem.

A MIP start will only be used by GAMS/CPLEX if the MipStart parameter is set to 1. The starting values must be set via the `.L` variable attribute in the GAMS model before the solve statement.

5.11.6.7 Using the Feasibility Relaxation

The feasibility relaxation is enabled by the FeasOpt parameter in a CPLEX solver option file.

With the FeasOpt option CPLEX accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candidates for relaxation and weighted equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the dot option FeasPref. A negative or zero preference means that the associated bound or constraint is not to be modified. The weighted penalty function is constructed from these preferences. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. However, it is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a CPLEX solver option file. The syntax is:

(variable or equation) `.feaspref` *(value)*

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the `cplex.opt` file can be specified by:

```
feasopt 1
v.feaspref            1
v.feaspref('i1',*)    2
v.feaspref('i1','j2') 0

e.feaspref(*,'j1')    0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxation on. Furthermore, we specify that all variables `v(i,j)` have preference of 1, except variables over set element `i1`, which have a preference of 2. The variable over set element `i1` and `j2` has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords `variables` or `equations` instead of the individual variable and equations names (e.g. `variables.feaspref 1`).

The parameter FeasOptMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasOptMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations). Please check description of parameters FeasOpt and FeasOptMode for details. Also check example models `feasopt*` in the GAMS Model library.

5.11.6.8 Sensitivity Analysis

Sensitivity analysis (post-optimality analysis) in linear programming allows one to find out more about an optimal solution for a problem. In particular, objective ranging and right-hand-side ranging give information about how much an objective coefficient or a right-hand-side value can change without changing the optimal basis. In other words, they give information about how sensitive the optimal basis is to a change in the objective function or a right-hand-side.

Although not so much used for practical large scale problems and not available for mixed-integer or nonlinear models, ranging information can still be of use in some circumstances. This section describes how to produce ranging information with GAMS/CPLEX.

To obtain objective ranging information for a particular variable, the name of the GAMS variable should be specified with the ObjRng option. For example, to obtain ranging information for a variable `prod`, add the line

```
objrng prod
```

to a CPLEX options file. The ObjRng option can be repeated to specify ranging for more than one variable. To specify ranging for all variables, use the keyword `all`, i.e.,

```
objrng all
```

Similarly, to obtain right-hand-side ranging information for a particular equation, the name of the equation should be specified with the RhsRng option. Also this option can be repeated to obtain right-hand-side ranging information for several equations. To specify ranging for all equations use the keyword `all`.

As an example, consider solving the model [**TRNSPORT**] from the GAMS model library with CPLEX and options file

```
objrng all
rhsrng all
```

This gives the following table in the listing file:

```
EQUATION NAME                              LOWER      CURRENT        UPPER
-------------                              -----      -------        -----
cost                                        -INF            0         +INF
supply(seattle)                              300          350          625
supply(san-diego)                            550          600         +INF
demand(new-york)                              50          325          375
demand(chicago)                               25          300          350
demand(topeka)                                 0          275          325


VARIABLE NAME                              LOWER      CURRENT        UPPER
-------------                              -----      -------        -----
x(seattle, new-york)                       0.216        0.225        0.225
x(seattle, chicago)                            0        0.153        0.162
x(seattle, topeka)                         0.126        0.162         +INF
x(san-diego, new-york)                     0.225        0.225        0.234
x(san-diego, chicago)                      0.153        0.162         +INF
x(san-diego, topeka)                           0        0.126        0.162
z                                           -INF            1         +INF
```

If obtaining ranging information in a listing file is not sufficient, option RngRestart can be used to specify a file to which to write ranging information in GAMS syntax. For example, using an options file containing

```
rhsrng supply
rhsrng demand
rngrestart ranges.inc
```

will result in a file named `ranges.inc` being written with the following content:

```
* Include file with ranging information
* The set RNGLIM /LO,UP/ is assumed to be
* declared.

PARAMETER  supplyRNG(i,RNGLIM) /
seattle.LO                  300
seattle.UP                  625
san-diego.LO                550
san-diego.UP               +INF
/;
PARAMETER  demandRNG(j,RNGLIM) /
new-york.LO                  50
new-york.UP                 375
chicago.LO                   25
chicago.UP                  350
topeka.LO                     0
topeka.UP                   325
/;
```

For each equation specified, the ranging information is stored in a newly declared corresponding GAMS parameter. The name of the parameter is based on the name of the equation, but with `RNG` appended. The user is responsible for ensuring that the new name does not exceed the maximum symbol name length of GAMS identifiers. Further, the domain list of the new parameter is the same as the domain list for the corresponding equation with an additional dimension added to the end. The user is responsible for ensuring that the new parameter does not exceed the maximum number of index positions.

5.11.7 GAMS/Cplex Log File

Cplex reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the primal simplex algorithm, the iteration log starts with the iteration number followed by the scaled infeasibility value. Once feasibility has been attained, the objective function value is listed instead. At the default value for option simdisplay there is a log line for each refactorization. The screen log has the following appearance:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.
Using conservative initial basis.

Iteration log . . .
Iteration:     1    Scaled infeas =        193998.067174
Iteration:    29    Objective     =         -3484.286415
Switched to devex.
Iteration:    98    Objective     =         -1852.931117
Iteration:   166    Objective     =          -349.706562

Optimal solution found.

Objective :         901.161538
```

The iteration log for the dual simplex algorithm is similar, but the dual infeasibility and dual objective are reported instead of the corresponding primal values:

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.

Iteration log . . .
Iteration:     1   Scaled dual infeas =             3.890823
Iteration:    53   Dual objective     =          4844.392441
Iteration:   114   Dual objective     =          1794.360714
Iteration:   176   Dual objective     =          1120.183325
Iteration:   238   Dual objective     =           915.143030
Removing shift (1).

Optimal solution found.

Objective :         901.161538
```

The log for the network algorithm adds statistics about the extracted network and a log of the network iterations. The optimization is finished by one of the simplex algorithms and an iteration log for that is produced as well.

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.01 sec.
Extracted network with 25 nodes and 116 arcs.
Extraction time =   -0.00 sec.
Iteration log . . .
Iteration:     0   Infeasibility     =          1232.378800 (-1.32326e+12)

Network - Optimal:  Objective =    1.5716820779e+03
Network time =    0.01 sec.  Iterations = 26 (24)

Iteration log . . .
Iteration:     1    Scaled infeas =        212696.154729
Iteration:    62    Scaled infeas =         10020.401232
Iteration:   142    Scaled infeas =          4985.200129
Switched to devex.
Iteration:   217    Objective     =         -3883.782587
Iteration:   291    Objective     =         -1423.126582

Optimal solution found.

Objective :         901.161538
```

The log for the barrier algorithm adds various algorithm specific statistics about the problem before starting the iteration log. The iteration log includes columns for primal and dual objective values and infeasibility values. A special log follows for the crossover to a basic solution.

```
Tried aggregator 1 time.
LP Presolve eliminated 2 rows and 39 columns.
Aggregator did 30 substitutions.
Reduced LP has 243 rows, 335 columns, and 3912 nonzeros.
Presolve time =    0.02 sec.
Number of nonzeros in lower triangle of A*A' = 6545
Using Approximate Minimum Degree ordering
Total time for automatic ordering = 0.01 sec.
Summary statistics for Cholesky factor:

  Rows in Factor            = 243
  Integer space required    = 578
  Total non-zeros in factor = 8491
  Total FP ops to factor    = 410889

 Itn      Primal Obj        Dual Obj  Prim Inf Upper Inf  Dual Inf
   0  -1.2826603e+06   7.4700787e+08  2.25e+10  6.13e+06  4.00e+05
   1  -2.6426195e+05   6.3552653e+08  4.58e+09  1.25e+06  1.35e+05
   2  -9.9117854e+04   4.1669756e+08  1.66e+09  4.52e+05  3.93e+04
   3  -2.6624468e+04   2.1507018e+08  3.80e+08  1.04e+05  1.20e+04
   4  -1.2104334e+04   7.8532364e+07  9.69e+07  2.65e+04  2.52e+03
   5  -9.5217661e+03   4.2663811e+07  2.81e+07  7.67e+03  9.92e+02
   6  -8.6929410e+03   1.4134077e+07  4.94e+06  1.35e+03  2.16e+02
   7  -8.3726267e+03   3.1619431e+06  3.13e-07  6.84e-12  3.72e+01
   8  -8.2962559e+03   3.3985844e+03  1.43e-08  5.60e-12  3.98e-02
   9  -3.8181279e+03   2.6166059e+03  1.58e-08  9.37e-12  2.50e-02
  10  -5.1366439e+03   2.8102021e+03  3.90e-06  7.34e-12  1.78e-02
  11  -1.9771576e+03   1.5960442e+03  3.43e-06  7.02e-12  3.81e-03
```

```
  12  -4.3346261e+02   8.3443795e+02  4.99e-07  1.22e-11  7.93e-04
  13   1.2882968e+02   5.2138155e+02  2.22e-07  1.45e-11  8.72e-04
  14   5.0418542e+02   5.3676806e+02  1.45e-07  1.26e-11  7.93e-04
  15   2.4951043e+02   6.5911879e+02  1.73e-07  1.43e-11  5.33e-04
  16   2.4666057e+02   7.6179064e+02  7.83e-06  2.17e-11  3.15e-04
  17   4.6820025e+02   8.1319322e+02  4.75e-06  1.78e-11  2.57e-04
  18   5.6081604e+02   7.9608915e+02  3.09e-06  1.98e-11  2.89e-04
  19   6.4517294e+02   7.7729659e+02  1.61e-06  1.27e-11  3.29e-04
  20   7.9603053e+02   7.8584631e+02  5.91e-07  1.91e-11  3.00e-04
  21   8.5871436e+02   8.0198336e+02  1.32e-07  1.46e-11  2.57e-04
  22   8.8146686e+02   8.1244367e+02  1.46e-07  1.84e-11  2.29e-04
  23   8.8327998e+02   8.3544569e+02  1.44e-07  1.96e-11  1.71e-04
  24   8.8595062e+02   8.4926550e+02  1.30e-07  2.85e-11  1.35e-04
  25   8.9780584e+02   8.6318712e+02  1.60e-07  1.08e-11  9.89e-05
  26   8.9940069e+02   8.9108502e+02  1.78e-07  1.07e-11  2.62e-05
  27   8.9979049e+02   8.9138752e+02  5.14e-07  1.88e-11  2.54e-05
  28   8.9979401e+02   8.9139850e+02  5.13e-07  2.18e-11  2.54e-05
  29   9.0067378e+02   8.9385969e+02  2.45e-07  1.46e-11  1.90e-05
  30   9.0112149e+02   8.9746581e+02  2.12e-07  1.71e-11  9.61e-06
  31   9.0113610e+02   8.9837069e+02  2.11e-07  1.31e-11  7.40e-06
  32   9.0113661e+02   8.9982723e+02  1.90e-07  2.12e-11  3.53e-06
  33   9.0115644e+02   9.0088083e+02  2.92e-07  1.27e-11  7.35e-07
  34   9.0116131e+02   9.0116262e+02  3.07e-07  1.81e-11  3.13e-09
  35   9.0116154e+02   9.0116154e+02  4.85e-07  1.69e-11  9.72e-13
Barrier time =    0.39 sec.

Primal crossover.
  Primal:  Fixing 13 variables.
       12 PMoves:  Infeasibility  1.97677059e-06  Objective  9.01161542e+02
        0 PMoves:  Infeasibility  0.00000000e+00  Objective  9.01161540e+02
  Primal:  Pushed 1, exchanged 12.
  Dual:  Fixing 3 variables.
        2 DMoves:  Infeasibility  1.28422758e-36  Objective  9.01161540e+02
        0 DMoves:  Infeasibility  1.28422758e-36  Objective  9.01161540e+02
  Dual:  Pushed 3, exchanged 0.
Using devex.
Total crossover time =    0.02 sec.

Optimal solution found.

Objective :         901.161540
```

For MIP problems, during the branch and bound search, Cplex reports the node number, the number of nodes left, the value of the Objective function, the number of integer variables that have fractional values, the current best integer solution, the best relaxed solution at a node and an iteration count. The last column show the current optimality gap as a percentage. CPLEX logs an asterisk (∗) in the left-most column for any node where it finds an integer-feasible solution or new incumbent. The + denotes an incumbent generated by the heuristic.

```
Tried aggregator 1 time.
MIP Presolve eliminated 1 rows and 1 columns.
Reduced MIP has 99 rows, 76 columns, and 419 nonzeros.
Presolve time =    0.00 sec.

Iteration log . . .
Iteration:     1   Dual objective     =             0.000000
Root relaxation solution time =    0.01 sec.
```

```
        Nodes                                         Cuts/
   Node  Left     Objective  IInf  Best Integer    Best Node    ItCnt     Gap

      0     0        0.0000    24                    0.0000       40
*     0+    0        6.0000     0        6.0000       0.0000       40  100.00%
*    50+   50        4.0000     0        4.0000       0.0000      691  100.00%
    100    99        2.0000    15        4.0000       0.4000     1448   90.00%
Fixing integer variables, and solving final LP..
Tried aggregator 1 time.
LP Presolve eliminated 100 rows and 77 columns.
All rows and columns eliminated.
Presolve time =    0.00 sec.

Solution satisfies tolerances.

MIP Solution    :            4.000000    (2650 iterations, 185 nodes)
Final LP        :            4.000000    (0 iterations)

Best integer solution possible :          1.000000
Absolute gap                   :                 3
Relative gap                   :               1.5
```

5.11.8 Detailed Descriptions of CPLEX Options

These options should be entered in the options file after setting the GAMS ModelName.OptFile parameter to 1. The name of the options file is `cplex.opt`. The options file is case insensitive and the keywords should be given in full.

advind *(integer)*: advanced basis use ↩

Use an Advanced Basis. GAMS/Cplex will automatically use an advanced basis from a previous solve statement. The GAMS `BRatio` option can be used to specify when not to use an advanced basis. The Cplex option `AdvInd` can be used to ignore a basis passed on by GAMS (it overrides *BRatio*).

Default: `determined by GAMS Bratio`

value	meaning
0	Do not use advanced basis
1	Use advanced basis if available
2	Crash an advanced basis if available (use basis with presolve)

aggcutlim *(integer)*: aggregation limit for cut generation ↩

Limits the number of constraints that can be aggregated for generating flow cover and mixed integer rounding cuts. For most purposes, the default will be satisfactory.

Default: `3`

aggfill *(integer)*: aggregator fill parameter ↩

Aggregator fill limit. If the net result of a single substitution is more non-zeros than the setting of the `AggFill` parameter, the substitution will not be made.

Default: `10`

aggind *(integer)*: aggregator on/off ↩

This option, when set to a nonzero value, will cause the Cplex aggregator to use substitution where possible to reduce the number of rows and columns in the problem. If set to a positive value, the aggregator will be applied the specified number of times, or until no more reductions are possible. At the default value of -1, the aggregator is applied once for linear programs and an unlimited number of times for mixed integer problems.

Default: `-1`

value	meaning
`-1`	Once for LP, unlimited for MIP
`0`	Do not use
`>0`	Aggregator will be applied the specified number of times

auxrootthreads *(integer)*: number of threads for auxiliary tasks at the root node ↩

Partitions the number of threads for CPLEX to use for auxiliary tasks while it solves the root node of a problem. On a system that offers N processors or N global threads, if you set this parameter to n, where $N>n>0$ then CPLEX uses at most n threads for auxiliary tasks and at most N-n threads to solve the root node. See also the parameter Threads.

You cannot set n, the value of this parameter, to a value greater than or equal to N, the number of processors or global threads offered on your system. In other words, when you set this parameter to a value other than its default, that value must be strictly less than the number of processors or global threads on your system. Independent of the auxiliary root threads parameter, CPLEX will never use more threads than those defined by the global default thread count parameter. CPLEX also makes sure that there is at least one thread available for the main root tasks. For example, if you set the global threads parameter to 3 and the auxiliary root threads parameter to 4, CPLEX still uses only two threads for auxiliary root tasks in order to keep one thread available for the main root tasks. At its default value, 0 (zero), CPLEX automatically chooses the number of threads to use for the primary root tasks and for auxiliary tasks. The number of threads that CPLEX uses to solve the root node depends on several factors: 1) the number of processors available on your system; 2) the number of threads available to your application on your system (for example, as a result of limited resources or competition with other applications); 3) the value of the global default thread count parameter Threads.

Default: `0`

value	meaning
`-1`	Off: do not use additional threads for auxiliary tasks
`0`	Automatic: let CPLEX choose the number of threads to use
`N>n>0`	Use n threads for auxiliary root tasks

baralg *(integer)*: algorithm selection ↩

Selects which barrier algorithm to use. The default setting of 0 uses the infeasibility-estimate start algorithm for LPs and MIP subproblems and the standard barrier algorithm, option 3, for other cases. The standard barrier algorithm is almost always fastest. The alternative algorithms, options 1 and 2, may eliminate numerical difficulties related to infeasibility, but will generally be slower.

Default: `0`

value	meaning
0	Same as 1 for LPs and MIP subproblems, 3 otherwise
1	Infeasibility-estimate start
2	Infeasibility-constant start
3	standard barrier algorithm

barcolnz *(integer)*: dense column handling ↩

Determines whether or not columns are considered dense for special barrier algorithm handling. At the default setting of 0, this parameter is determined dynamically. Values above 0 specify the number of entries in columns to be considered as dense.

Default: 0

barcrossalg *(integer)*: barrier crossover method ↩

Selects which crossover method is used at the end of a barrier optimization. To turn off crossover set SolutionType to 2.

Default: 0

value	meaning
0	Automatic
1	Primal crossover
2	Dual crossover

bardisplay *(integer)*: progress display level ↩

Determines the level of progress information to be displayed while the barrier method is running.

Default: 1

value	meaning
0	No progress information
1	Display normal information
2	Display diagnostic information

barepcomp *(real)*: convergence tolerance ↩

Determines the tolerance on complementarity for convergence of the barrier algorithm. The algorithm will terminate with an optimal solution if the relative complementarity is smaller than this value.

Default: 1.0e-08

bargrowth *(real)*: unbounded face detection ↩

Used by the barrier algorithm to detect unbounded optimal faces. At higher values, the barrier algorithm will be less likely to conclude that the problem has an unbounded optimal face, but more likely to have numerical difficulties if the problem does have an unbounded face.

Default: 1.0e+12

baritlim *(integer)*: iteration limit ↩

Determines the maximum number of iterations for the barrier algorithm. When set to 0, no Barrier iterations occur, but problem *setup* occurs and information about the setup is displayed (such as Cholesky factorization information). When left at the default value, there is no explicit limit on the number of iterations.

Default: `large`

barmaxcor *(integer)*: maximum correction limit ↩

Specifies the maximum number of centering corrections that should be done on each iteration. Larger values may improve the numerical performance of the barrier algorithm at the expense of computation time. The default of -1 means the number is automatically determined.

Default: `-1`

barobjrng *(real)*: maximum objective function ↩

Determines the maximum absolute value of the objective function. The barrier algorithm looks at this limit to detect unbounded problems.

Default: `1.0e+20`

barorder *(integer)*: row ordering algorithm selection ↩

Determines the ordering algorithm to be used by the barrier method. By default, Cplex attempts to choose the most effective of the available alternatives. Higher numbers tend to favor better orderings at the expense of longer ordering run times. The automatic option includes additional processing and may yield results that differ from the explicit choice.

Default: `0`

value	meaning
0	Automatic
1	Approximate Minimum Degree (AMD)
2	Approximate Minimum Fill (AMF)
3	Nested Dissection (ND)

barqcpepcomp *(real)*: convergence tolerance for the barrier optimizer for QCPs ↩

Range: [`1.0e-12`, `1.0e+75`]

Default: `1.0e-07`

barstartalg *(integer)*: barrier starting point algorithm ↩

This option sets the algorithm to be used to compute the initial starting point for the barrier solver. The default starting point is satisfactory for most problems. Since the default starting point is tuned for primal problems, using the other starting points may be worthwhile in conjunction with the PreDual parameter.

Default: `1`

value	meaning
1	default primal, dual is 0
2	default primal, estimate dual
3	primal average, dual is 0
4	primal average, estimate dual

bbinterval *(integer)*: best bound interval ↩

Set interval for selecting a best bound node when doing a best estimate search. Active only when NodeSel is 2 (best estimate). Decreasing this interval may be useful when best estimate is finding good solutions but making little progress in moving the bound. Increasing this interval may help when the best estimate node selection is not finding any good integer solutions. Setting the interval to 1 is equivalent to setting NodeSel to 1.

Default: `7`

bendersfeascuttol *(real)*: Tolerance for whether a feasibility cut has been violated in Benders decomposition ↩

Default: `1.0e-06`

bendersoptcuttol *(real)*: Tolerance for optimality cuts in Benders decomposition ↩

Default: `1.0e-06`

.benderspartition *(integer)*: Benders partition ↩

Default: `0`

benderspartitioninstage *(boolean)*: Benders partition through stage variable suffix ↩

Default: `0`

bendersstrategy *(integer)*: Benders decomposition algorithm as a strategy ↩

Given a formulation of a problem, CPLEX can decompose the model into a single master and (possibly multiple) subproblems. To do so, CPLEX can make use of annotations that you supply for your model. The strategy can be applied to mixed-integer linear programs (MILP). For certain types of problems, this approach offers significant performance improvements as subproblems can be solved in parallel.

For mixed integer programs (MIP), under certain conditions, CPLEX can apply Benders algorithm to improve the search to find more feasible solutions more quickly.

Default: `0`

value	meaning
-1	Off Execute conventional branch and bound; ignore any Benders annotations. That is, do not use Benders algorithm even if a Benders partition of the current model is present

value	meaning
0	Automatic If annotations specifying a Benders partition of the current model are available, CPLEX attempts to decompose the model. CPLEX uses the master as given by the annotations, and attempts to partition the subproblems further, if possible, before applying Benders algorithm to solve the model. If the user supplied annotations, but the annotations supplied do not lead to a complete decomposition into master and disjoint subproblems (that is, if the annotations are wrong in that sense), CPLEX produces an error.
1	Apply user annotations CPLEX applies Benders algorithm to a decomposition based on annotations supplied by the user. If no annotations to decompose the model are available, this setting produces an error. If the user supplies annotations, but the supplied annotations do not lead to a complete partition of the original model into disjoint master and subproblems, then this setting produces an error.
2	Apply user annotations with automatic support for subproblems CPLEX accepts the master as given and attempts to decompose the remaining elements into disjoint subproblems to assign to workers. It then solves the Benders decomposition of the model. If no annotations to decompose the model are available, this setting produces an error. If the user supplies annotations, but the supplied annotations do not lead to a complete partition of the original model into disjoint master and subproblems, then this setting produces an error.
3	Apply automatic decomposition CPLEX ignores any annotation supplied with the model; CPLEX applies presolve; CPLEX then automatically generates a Benders partition, putting integer variables in master and continuous linear variables into disjoint subproblems. CPLEX then solves the Benders decomposition of the model. If the problem is a strictly linear program (LP), that is, there are no integer-constrained variables to put into master, then CPLEX reports an error. If the problem is a mixed integer linear program (MILP) where all variables are integer-constrained, (that is, there are no continuous linear variables to decompose into disjoint subproblems) then CPLEX reports an error.

bndrng *(string)*: do lower / upper bound ranging ↩

Calculate sensitivity ranges for the specified GAMS lower and upper bounds. Unlike most options, `BNDRng` can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS lower and upper bound named. Specifying `all` will cause range information to be produced for all lower and upper bounds. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option RngRestart is specified.

bndstrenind *(integer)*: bound strengthening ↩

Use bound strengthening when solving mixed integer problems. Bound strengthening tightens the bounds on variables, perhaps to the point where the variable can be fixed and thus removed from consideration during the branch and bound algorithm. This reduction is usually beneficial, but occasionally, due to its iterative nature, takes a long time.

Default: `-1`

value	meaning
-1	Determine automatically
0	Don't use bound strengthening
1	Use bound strengthening

bqpcuts *(integer)*: boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality ↩

Default: `0`

value	meaning
`-1`	Do not generate BQP cuts
`0`	Determined automatically
`1`	Generate BQP cuts moderately
`2`	Generate BQP cuts aggressively
`3`	Generate BQP cuts very aggressively

brdir *(integer)*: set branching direction ↩

Used to decide which branch (up or down) should be taken first at each node.

Default: `0`

value	meaning
`-1`	Down branch selected first
`0`	Algorithm decides
`1`	Up branch selected first

bttol *(real)*: backtracking limit ↩

This option controls how often backtracking is done during the branching process. At each node, Cplex compares the objective function value or estimated integer objective value to these values at parent nodes; the value of the `bttol` parameter dictates how much relative degradation is tolerated before backtracking. Lower values tend to increase the amount of backtracking, making the search more of a pure best-bound search. Higher values tend to decrease the amount of backtracking, making the search more of a depth-first search. This parameter is used only once a first integer solution is found or when a cutoff has been specified.

Range: [`0.0`, `1.0`]

Default: `1.0`

calcqcpduals *(integer)*: calculate the dual values of a quadratically constrained problem ↩

Default: `1`

value	meaning
`0`	Do not calculate dual values
`1`	Calculate dual values as long as it does not interfere with presolve reductions
`2`	Calculate dual values and disable any presolve reductions that would interfere

cardls *(integer)*: decides how often to apply the cardinality local search heuristic (CLSH) ↩

Default: `-1`

value	meaning
`-1`	Do not apply CLSH

value	meaning
0	Automatic
1	Apply the CLSH only at the root node
2	Apply the CLSH at the nodes of the branch and bound tree

cliques *(integer)*: clique cut generation ↩

Determines whether or not clique cuts should be generated during optimization.

Default: 0

value	meaning
-1	Do not generate clique cuts
0	Determined automatically
1	Generate clique cuts moderately
2	Generate clique cuts aggressively
3	Generate clique cuts very aggressively

clocktype *(integer)*: clock type for computation time ↩

Decides how computation times are measured for both reporting performance and terminating optimization when a time limit has been set. Small variations in measured time on identical runs may be expected on any computer system with any setting of this parameter.

Default: 2

value	meaning
0	Automatic
1	CPU time
2	Wall clock time

clonelog *(integer)*: enable clone logs ↩

The clone logs contain information normally recorded in the ordinary log file but inconvenient to send through the normal log channel in case of parallel execution. The information likely to be of most interest to you are special messages, such as error messages, that result from calls to the LP optimizers called for the subproblems. The clone log files are named cloneK.log, where K is the index of the clone, ranging from 0 (zero) to the number of threads minus one. Since the clones are created at each call to a parallel optimizer and discarded when it exits, the clone logs are opened at each call and closed at each exit. The clone log files are not removed when the clones themselves are discarded.

Default: 0

value	meaning
-1	Clone log files off
0	Automatic
1	Clone log files on

coeredind *(integer)*: coefficient reduction on/off ↩

Coefficient reduction is a technique used when presolving mixed integer programs. The benefit is to improve the objective value of the initial (and subsequent) linear programming relaxations by reducing the number of non-integral vertexes. However, the linear programs generated at each node may become more difficult to solve.

Default: `-1`

value	meaning
`-1`	Automatic
`0`	Do not use coefficient reduction
`1`	Reduce only to integral coefficients
`2`	Reduce all potential coefficients
`3`	Reduce aggressively with tilting

conflictalg *(integer)*: algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model ↩

Default: `0`

conflictdisplay *(integer)*: decides how much information CPLEX reports when the conflict refiner is working ↩

Default: `1`

value	meaning
`0`	No output
`1`	Summary display
`2`	Detailed display

covers *(integer)*: cover cut generation ↩

Determines whether or not cover cuts should be generated during optimization.

Default: `0`

value	meaning
`-1`	Do not generate cover cuts
`0`	Determined automatically
`1`	Generate cover cuts moderately
`2`	Generate cover cuts aggressively
`3`	Generate cover cuts very aggressively

cpumask *(string)*: switch and mask to bind threads to processors (Linux only) ↩

The value of this parameter serves as a switch to turn on (or to turn off) CPLEX binding of multiple threads to multiple processors on platforms where this feature is available. Hexadecimal values of this parameter serve as a mask to specify to CPLEX which processors (or cores) to use in binding multiple threads to multiple processors. CPU binding is also sometimes known as *processor affinity*. CPU binding reduces the variability of CPLEX runs. On some occasions, running the same CPLEX on the same (non trivial) models would produce a big variation in runtime, e.g. 1000 seconds versus 900 seconds on a 12 core machine. These differences happen

while CPLEX still gets exactly the same results and executes the exact same path, thanks to its completely deterministic algorithms. Running the same tests with CPU binding enabled reduced this variability in running time significantly.

If not set to `off` or `auto` CPLEX treats the value of this parameter as a string that resembles a hexadecimal number without the usual 0x prefix. A valid string consists of these elements: a) any digit from 0 (zero) through 9 (inclusive), b) any lower case character in the range a through f (inclusive), and c) any upper case character in the range A through F (inclusive). CPLEX rejects a string containing any other digits or characters than those.

When the value of this parameter is a valid string, each bit of this string corresponds to a central processing unit (CPU), that is, to a processor or core. The lowest order bit of the string corresponds to the first logical CPU, and the highest order corresponds to the last logical CPU. For example, `00000001` designates processor #0, `00000003` designates processors #0 and #1, `FFFFFFFF` designates all processors #0 through #31. CPLEX uses the ith CPU if and only if the ith bit of this string is set to 1 (one). Tip: For GNU/Linux users, this parameter behaves like the `taskset` command (except that this parameter lacks the prefix 0x).

If this CPU mask parameter is set to a valid string that designates a hexadecimal number, but global Threads count is set to 0 (zero), then CPLEX still starts as many threads as the number of cores on the machine, but only the cores enabled in the mask will be used.

For example, if a user sets this CPU mask parameter to the hexadecimal value "f" on a 16-core machine, and the user sets the global Threads count to 0 (zero), the result is 16 threads. These 16 threads will be bound to the first four cores in a round-robin way: treads 1,5,9,13 to core 1, threads 2,6,10,14 to core 2 and so on. This situation is probably not what the user intended. Therefore, if you set this CPU mask parameter, then you should also set global threads count; indeed, you should set the threads parameter to the number of active cores designated by the mask.

For example, on a 16 core machine, consider the difference between the value "off" and the value *ffff*. If the value of this parameter is "off" CPLEX does no binding. If the value of this parameter is *ffff*, CPLEX binds threads to cores.

Default: `auto`

value	meaning
`auto`	CPLEX decides whether to bind threads to cores (or processors)
`off`	CPLEX performs no binding
`hex`	CPLEX binds the threads in round-robin fashion to the cores specified by the mask

craind *(integer)*: crash strategy (used to obtain starting basis) ↩

The crash option biases the way Cplex orders variables relative to the objective function when selecting an initial basis.

Default: `1`

value	meaning
`-1`	Primal: alternate ways of using objective coefficients. Dual: aggressive starting basis
`0`	Primal: ignore objective coefficients during crash. Dual: aggressive starting basis
`1`	Primal: alternate ways of using objective coefficients. Dual: default starting basis

cutlo *(real)*: lower cutoff for tree search ↩

Sets the lower cutoff tolerance. When the problem is a maximization problem, CPLEX cuts off or discards solutions that are less than the specified cutoff value. If the model has no solution with an objective value greater than or equal to the cutoff value, then CPLEX declares the model infeasible. In other words, setting the lower cutoff value c for a maximization problem is similar to adding this constraint to the objective function of the model: `obj>=c`.

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

Default: `-1.0e+75`

cutpass *(integer)*: maximum number of cutting plane passes ↩

Sets the upper limit on the number of passes that will be performed when generating cutting planes on a mixed integer model.

Default: `0`

value	meaning
`-1`	None
`0`	Automatically determined
`>0`	Maximum passes to perform

cuts *(string)*: default cut generation ↩

Allows generation setting of all optional cuts at once. This is done by changing the meaning of the default value (0: automatic) for the various Cplex cut generation options. The options affected are `Cliques`, `Covers`, `DisjCuts`, `FlowCovers`, `FlowPaths`, `FracCuts`, `GUBCovers`, `ImplBd`, `LiftProjCuts`, `MCFCuts`, `MIRCuts`, and `Symmetry`.

Default: `0`

value	meaning
`-1`	Do not generate cuts
`0`	Determined automatically
`1`	Generate cuts moderately
`2`	Generate cuts aggressively
`3`	Generate cuts very aggressively
`4`	Generate cuts highly aggressively
`5`	Generate cuts extremely aggressively

cutsfactor *(real)*: cut limit ↩

This option limits the number of cuts that can be added. For values between zero and one inclusive (that is, in the range [0.0, 1.0], CPLEX generates no cuts.

For values strictly greater than 1.0 (one), CPLEX limits the number of rows in the model with cuts added.

The limit on this total is the product of *CutsFactor* times the original number of rows. If CPLEX has presolved the model, the *original number of rows* is the number of rows in the presolved model. (This behavior with respect to a presolved model is unchanged.)

CPLEX regards negative values of this parameter as equivalent to the default value -1.0. That is, a negative value specifies no particular limit on the number of cuts. CPLEX computes and dynamically adjusts such a limit automatically

Default: `-1.0`

cutup *(real)*: upper cutoff for tree search ↩

Sets the upper cutoff tolerance. When the problem is a minimization problem, CPLEX cuts off or discards any solutions that are greater than the specified upper cutoff value. If the model has no solution with an objective value less than or equal to the cutoff value, CPLEX declares the model infeasible. In other words, setting an upper cutoff value c for a minimization problem is similar to adding this constraint to the objective function of the model: `obj<=c`.

This option overrides the GAMS Cutoff setting.

This parameter is not effective with FeasOpt. FeasOpt cannot analyze an infeasibility introduced by this parameter. If you want to analyze such a condition, add an explicit objective constraint to your model instead.

Default: `1.0e+75`

datacheck *(integer)*: controls data consistency checking and modeling assistance ↩

When the value of this parameter is set to level 2, CPLEX turns on both data consistency checking and modeling assistance. At this level, CPLEX issues warnings at the start of the optimization about disproportionate values (too large, too small) in coefficients, bounds, and righthand sides (RHS).

Default: `0`

value	meaning
0	Data checking off
1	Data checking on
2	Data checking and model assistance on

depind *(integer)*: dependency checker on/off ↩

This option determines if and when the dependency checker will be used.

Default: `-1`

value	meaning
-1	Automatic
0	Turn off dependency checking
1	Turn on only at the beginning of preprocessing
2	Turn on only at the end of preprocessing
3	Turn on at the beginning and at the end of preprocessing

dettilim *(real)*: deterministic time limit ↩

Sets a time limit expressed in ticks, a unit to measure work done deterministically.

The length of a deterministic tick may vary by platform. Nevertheless, ticks are normally consistent measures for a given platform (combination of hardware and software) carrying the same load. In other words, the correspondence of ticks to clock time depends on the hardware, software, and the current load of the machine. For the same platform and same load, the ratio of ticks per second stays roughly constant, independent of the model solved. However, for very short optimization runs, the variation of this ratio is typically high.

Default: `1.0e+75`

disjcuts *(integer)*: disjunctive cuts generation ↩

Determines whether or not to generate disjunctive cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: `0`

value	meaning
`-1`	Do not generate disjunctive cuts
`0`	Determined automatically
`1`	Generate disjunctive cuts moderately
`2`	Generate disjunctive cuts aggressively
`3`	Generate disjunctive cuts very aggressively

divetype *(integer)*: MIP dive strategy ↩

The MIP traversal strategy occasionally performs probing dives, where it looks ahead at both children nodes before deciding which node to choose. The default (automatic) setting chooses when to perform a probing dive, and the other two settings direct Cplex when to perform probing dives: never or always.

Default: `0`

value	meaning
`0`	Automatic
`1`	Traditional dive
`2`	Probing dive
`3`	Guided dive

.divflt *(real)*: solution pool range filter coefficients ↩

A diversity filter for a solution pool (see option SolnPool) allows you generate solutions that are similar to (or different from) a set of reference values that you specify for a set of binary variables. In particular, you can use a diversity filter to generate more solutions that are similar to an existing solution or to an existing partial solution.

A diversity filter drives the search for multiple solutions toward new solutions that satisfy a measure of diversity specified in the filter. This diversity measure applies only to binary variables. Potential new solutions are compared to a reference set. This reference set is specified with this dot option. If no reference set is specified, the difference measure will be computed relative to the other solutions in the pool. The diversity measure is computed by summing the pair-wise absolute differences from solution and the reference values.

Default: `0`

divfltlo *(real)*: lower bound on diversity ↩

Please check option DivFlt for general information on a diversity filter.

If you specify a lower bound on the diversity using `DivFltLo`, Cplex will look for solutions that are different from the reference values. In other words, you can say, Give me solutions that differ by at least this amount in this set of variables.

Default: `mindouble`

divfltup *(real)*: upper bound on diversity ↩

Please check option DivFlt for general information on a diversity filter.

If you specify an upper bound on diversity `DivFltUp`, Cplex will look for solutions similar to the reference values. In other words, you can say, Give me solutions that are close to this one, within this set of variables.

Default: `maxdouble`

dpriind *(integer)*: dual simplex pricing ↩

Pricing strategy for dual simplex method. Consider using dual steepest-edge pricing. Dual steepest-edge is particularly efficient and does not carry as much computational burden as the primal steepest-edge pricing.

Default: `0`

value	meaning
0	Determined automatically
1	Standard dual pricing
2	Steepest-edge pricing
3	Steepest-edge pricing in slack space
4	Steepest-edge pricing, unit initial norms
5	Devex pricing

dynamicrows *(integer)*: switch for dynamic management of rows ↩

This parameter specifies how CPLEX should manage rows in the current model during dual simplex optimization. More specifically, this parameter controls the use of the kernel simplex method (KSM) for the dual simplex algorithm. That is, CPLEX dynamically adjusts the dimensions of the basis matrix during execution of the dual simplex algorithm, according to the settings of this parameter.

When the value of this parameter is -1, its default value, this parameter specifies that the user wants CPLEX to manage rows dynamically, adjusting the dimensions of the basis matrix during dual simplex optimization. When it is set to 0, this parameter specifies that CPLEX must keep all rows. When it is set to 1, this parameter specifies that CPLEX can keep or discard rows according to its internal calculations.

Default: `-1`

value	meaning
-1	Automatic
0	Keep all rows
1	Manage rows

eachcutlim *(integer)*: sets a limit for each type of cut ↩

This parameter allows you to set a uniform limit on the number of cuts of each type that Cplex generates. By default, the limit is a large integer; that is, there is no effective limit by default.

Tighter limits on the number of cuts of each type may benefit certain models. For example, a limit on each type of cut will prevent any one type of cut from being created in such large number that the limit on the total number of all types of cuts is reached before other types of cuts have an opportunity to be created. A setting of 0 means no cuts.

This parameter does not influence the number of Gomory cuts. For means to control the number of Gomory cuts, see also the fractional cut parameters: FracCand, FracCuts, and FracPass.

Default: `2100000000`

epagap *(real)*: absolute stopping tolerance ↩

Synonym: optca

Absolute tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the `epagap` setting, the optimization is stopped. This option overrides GAMS OptCA which provides its initial value.

Note: This option also influences the SubMIPs (e.g., used for the RINS heuristic) and can thus influence the solution path.

Default: `GAMS OptCA`

epgap *(real)*: relative stopping tolerance ↩

Synonym: optcr

Relative tolerance on the gap between the best integer objective and the objective of the best node remaining. When the value falls below the value of the `epgap` setting, the mixed integer optimization is stopped. Note the difference in the Cplex definition of the relative tolerance with the GAMS definition. This option overrides GAMS OptCR which provides its initial value.

Note: This option also influences the SubMIPs (e.g., used for the RINS heuristic) and can thus influence the solution path.

Range: [`0.0`, `1.0`]

Default: `GAMS OptCR`

epint *(real)*: integrality tolerance ↩

Integrality Tolerance. This specifies the amount by which an integer variable can be different than an integer and still be considered feasible.

Range: [`0.0`, `0.5`]

Default: `1.0e-05`

eplin *(real)*: degree of tolerance used in linearization ↩

Default: `0.001`

epmrk *(real)*: Markowitz pivot tolerance ↩

The Markowitz tolerance influences pivot selection during basis factorization. Increasing the Markowitz threshold may improve the numerical properties of the solution.

Range: [`1.0e-04`, `1.0`]

Default: `0.01`

epopt *(real)*: optimality tolerance ↩

The optimality tolerance influences the reduced-cost tolerance for optimality. This option setting governs how closely Cplex must approach the theoretically optimal solution.

Range: [`1.0e-09`, `0.1`]

Default: `1.0e-06`

epper *(real)*: perturbation constant ↩

Perturbation setting. Highly degenerate problems tend to stall optimization progress. Cplex automatically perturbs the variable bounds when this occurs. Perturbation expands the bounds on every variable by a small amount thereby creating a different but closely related problem. Generally, the solution to the less constrained problem is easier to solve. Once the solution to the perturbed problem has advanced as far as it can go, Cplex removes the perturbation by resetting the bounds to their original values.

If the problem is perturbed more than once, the perturbation constant is probably too large. Reduce the `epper` option to a level where only one perturbation is required. Any value greater than or equal to 1.0e-8 is valid.

Default: `1.0e-06`

eprhs *(real)*: feasibility tolerance ↩

Feasibility tolerance. This specifies the degree to which a problem's basic variables may violate their bounds. This tolerance influences the selection of an optimal basis and can be reset to a higher value when a problem is having difficulty maintaining feasibility during optimization. You may also wish to lower this tolerance after finding an optimal solution if there is any doubt that the solution is truly optimal. If the feasibility tolerance is set too low, Cplex may falsely conclude that a problem is infeasible.

Range: [`1.0e-09`, `0.1`]

Default: `1.0e-06`

exactkappa *(boolean)*: report exact condition number in quality report ↩

Default: `0`

feasopt *(boolean)*: computes a minimum-cost relaxation to make an infeasible model feasible ↩

With `Feasopt` turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option FeasOptMode.

Feasible relaxations are available for all problem types with the exception of quadratically constraint problems.

Default: `0`

value	meaning
0	Turns Feasible Relaxation off
1	Turns Feasible Relaxation on

feasoptmode *(integer)*: mode of FeasOpt ↩

The parameter `FeasOptMode` allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter `FeasOptMode` indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations).

Default: `0`

value	meaning
0	Minimize sum of relaxations Minimize the sum of all required relaxations in first phase only
1	Minimize sum of relaxations and optimize Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations
2	Minimize number of relaxations Minimize the number of constraints and bounds requiring relaxation in first phase only
3	Minimize number of relaxations and optimize Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations
4	Minimize sum of squares of relaxations Minimize the sum of squares of required relaxations in first phase only
5	Minimize sum of squares of relaxations and optimize Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

.feaspref *(real)*: feasibility preference ↩

You can express the costs associated with relaxing a bound or right hand side value during a FeasOpt run through the `.feaspref` option. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

Default: `1`

fixoptfile *(string)*: name of option file which is read just before solving the fixed problem ↩

flowcovers *(integer)*: flow cover cut generation ↩

Determines whether or not flow cover cuts should be generated during optimization.

Default: `0`

value	meaning
-1	Do not generate flow cover cuts
0	Determined automatically
1	Generate flow cover cuts moderately
2	Generate flow cover cuts aggressively

flowpaths *(integer)*: flow path cut generation ↩

Determines whether or not flow path cuts should be generated during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: 0

value	meaning
-1	Do not generate flow path cuts
0	Determined automatically
1	Generate flow path cuts moderately
2	Generate flow path cuts aggressively

folding *(integer)*: LP folding will be attempted during the preprocessing phase ↩

Default: -1

value	meaning
-1	Automatic
0	Turn off folder
1	Moderate level of folding
2	Aggressive level of folding
3	Very aggressive level of folding
4	Highly aggressive level of folding
5	Extremely aggressive level of folding

fpheur *(integer)*: feasibility pump heuristic ↩

Controls the use of the feasibility pump heuristic for mixed integer programming (MIP) models.

Default: 0

value	meaning
-1	Turns Feasible Pump heuristic off
0	Automatic
1	Apply the feasibility pump heuristic with an emphasis on finding a feasible solution
2	Apply the feasibility pump heuristic with an emphasis on finding a feasible solution with a good objective value

fraccand *(integer)*: candidate limit for generating Gomory fractional cuts ↩

Limits the number of candidate variables for generating Gomory fractional cuts.

Default: 200

fraccuts *(integer)*: Gomory fractional cut generation ↩

Determines whether or not Gomory fractional cuts should be generated during optimization.

Default: `0`

value	meaning
`-1`	Do not generate Gomory fractional cuts
`0`	Determined automatically
`1`	Generate Gomory fractional cuts moderately
`2`	Generate Gomory fractional cuts aggressively

fracpass *(integer)*: maximum number of passes for generating Gomory fractional cuts ↩

Sets the upper limit on the number of passes that will be performed when generating Gomory fractional cuts on a mixed integer model. Ignored if parameter FracCuts is set to a nonzero value.

Default: `0`

value	meaning
`0`	0 Automatically determined
`>0`	Maximum passes to perform

freegamsmodel *(boolean)*: preserves memory by dumping the GAMS model instance representation temporarily to disk ↩

In order to provide the maximum amount of memory to the solver this option dumps the internal representation of the model instance temporarily to disk and frees memory. This option only works with `SolveLink=0` and only for models without quadratic constraints.

Default: `0`

gubcovers *(integer)*: GUB cover cut generation ↩

Determines whether or not GUB (Generalized Upper Bound) cover cuts should be generated during optimization. The default of 0 indicates that the attempt to generate GUB cuts should continue only if it seems to be helping.

Default: `0`

value	meaning
`-1`	Do not generate GUB cover cuts
`0`	Determined automatically
`1`	Generate GUB cover cuts moderately
`2`	Generate GUB cover cuts aggressively

heurfreq *(integer)*: heuristic frequency ↩

This option specifies how often to apply the node heuristic. Setting to a positive number applies the heuristic at the requested node interval. A value of 100, for example, means that heuristics are invoked every hundredth node in the tree.

Default: `0`

value	meaning
-1	Do not use the node heuristic
0	Determined automatically
>0	Call heuristic at the requested node interval

heuristiceffort *(real)*: the effort that CPLEX spends on heuristics during a MIP solve ↩

The value is used to increase (if *>1*) or decrease (if *<1*) the effort that CPLEX spends on heuristics during a MIP solve. If set to 0, no heuristic will run.

Default: 1.0

iafile *(string)*: secondary option file to be read in interactive mode triggered by iatriggerfile ↩

If in interactive mode and this option is set, options will be read from the file specified by this option instead of direct user input (as described in interactive). This option file read can be triggered by interrupting Cplex with a Control-C or using the option iatriggerfile. If defined, GAMS/CPLEX looks for this file (content irrelevant) all iatriggertime seconds and if found, reads the option file iafile. The iatriggerfile is removed afterwards so it does not trigger twice.

iatriggerfile *(string)*: file that triggers the reading of a secondary option file in interactive mode ↩

See iafile.

iatriggertime *(real)*: time interval in seconds the link looks for the trigger file in interactive mode ↩

See iafile.

Default: 60

iis *(integer)*: run the conflict refiner also known as IIS finder if the problem is infeasible ↩

Find an set of conflicting constraints or IIS (Irreducably Inconsistent Set) and write an conflict report to the GAMS solution listing if the model is found to be infeasible.

Default: 0

value	meaning
0	No conflict analysis
1	Conflict analysis after solve if infeasible
2	Conflict analysis without previous solve

implbd *(integer)*: implied bound cut generation ↩

Determines whether or not implied bound cuts should be generated during optimization.

Default: 0

value	meaning
-1	Do not generate implied bound cuts
0	Determined automatically
1	Generate implied bound cuts moderately
2	Generate implied bound cuts aggressively

indicoptstrict *(boolean)*: abort in case of an error in indicator constraint in solver option file ↩

If enabled and a variable or equation specified in an indicator constraint is not present in the model, model generation will abort with an error message. Otherwise, if this option is disabled, erroneous indicator constraints are ignored and a warning is printed.

Default: `1`

interactive *(boolean)*: allow interactive option setting after a Control-C ↩

When set to yes, options can be set interactively after interrupting Cplex with a Control-C. Options are entered just as if they were being entered in the `cplex.opt` file. Control is returned to Cplex by entering `continue`. The optimization can be aborted by entering `abort`. This option can only be used when running from the command line. Moreover, the GAMS option `InteractiveSolver` needs to be set to 1.

Default: `0`

intsollim *(integer)*: maximum number of integer solutions ↩

This option limits the MIP optimization to finding only this number of mixed integer solutions before stopping.

Default: `large`

itlim *(integer)*: iteration limit ↩

Synonym: iterlim

The iteration limit option sets the maximum number of iterations before the algorithm terminates, without reaching optimality. This Cplex option overrides the GAMS IterLim option. Any non-negative integer value is valid.

Default: `GAMS IterLim`

.lazy *(boolean)*: Lazy constraints activation ↩

Determines whether a linear constraint is treated as a lazy constraint. At the beginning of the MIP solution process, any constraint whose Lazy attribute is set to 1 (the default value is 0) is removed from the model and placed in the lazy constraint pool. Lazy constraints remain inactive until a feasible solution is found, at which point the solution is checked against the lazy constraint pool. If the solution violates any lazy constraint, the solution is discarded and one or more of the violated lazy constraints are pulled into the active model.

Lazy constraints are only active if option LazyConstraints is enabled and are specified through the option `.lazy`. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: `0`

lazyconstraints *(boolean)*: Indicator to use lazy constraints ↩

Default: `0`

lbheur *(boolean)*: local branching heuristic ↩

This parameter lets you control whether Cplex applies a local branching heuristic to try to improve new incumbents found during a MIP search. By default, this parameter is off. If you turn it on, Cplex will invoke a local branching heuristic only when it finds a new incumbent. If Cplex finds multiple incumbents at a single node, the local branching heuristic will be applied only to the last one found.

Default: `0`

value	meaning
0	Off
1	Apply local branching heuristic to new incumbent

liftprojcuts *(integer)*: lift-and-project cuts ↩

Default: 0

value	meaning
-1	Do not generate lift-and-project cuts
0	Determined automatically
1	Generate lift-and-project cuts moderately
2	Generate lift-and-project cuts aggressively
3	Generate lift-and-project cuts very aggressively

localimplied *(integer)*: generation of locally valid implied bound cuts ↩

Default: 0

value	meaning
-1	Do not generate locally valid implied bound cuts
0	Determined automatically
1	Generate locally valid implied bound cuts moderately
2	Generate locally valid implied bound cuts aggressively
3	Generate locally valid implied bound cuts very aggressively

lowerobjstop *(real)*: in a minimization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value lower than or equal to the specified value ↩

Default: -1e75

lpmethod *(integer)*: algorithm to be used for LP problems ↩

Specifies which LP algorithm to use. If left at the default value (0 for automatic), and a primal-feasible basis is available, primal simplex will be used. If no primal-feasible basis is available, and Threads is equal to 1, dual simplex will be used. If Threads is greater than 1 and no primal-feasible basis is available, the concurrent option will be used.

Sifting may be useful for problems with many more variables than equations.

The concurrent option runs multiple methods in parallel. The first thread uses dual simplex. The second thread uses barrier. The next thread uses primal simplex. Remaining threads are used by the barrier run. If the aspect ratio (number of columns versus number of rows) is large, and if more than 10 threads are available, then concurrent optimization also invokes sifting on the LP. The solution is returned by first method to finish.

Default: 0

value	meaning
0	Automatic
1	Primal Simplex

value	meaning
2	Dual Simplex
3	Network Simplex
4	Barrier
5	Sifting
6	Concurrent

ltol *(real)*: basis identification primal tolerance ↩

Default: 0

mcfcuts *(integer)*: multi-commodity flow cut generation ↩

Specifies whether Cplex should generate multi-commodity flow (MCF) cuts in a problem where Cplex detects the characteristics of a multi-commodity flow network with arc capacities. By default, Cplex decides whether or not to generate such cuts. To turn off generation of such cuts, set this parameter to -1. Cplex is able to recognize the structure of a network as represented in many real-world models. When it recognizes such a network structure, Cplex is able to generate cutting planes that usually help solve such problems. In this case, the cuts that Cplex generates state that the capacities installed on arcs pointing into a component of the network must be at least as large as the total flow demand of the component that cannot be satisfied by flow sources within the component.

Default: 0

value	meaning
-1	Do not generate MCF cuts
0	Determined automatically
1	Generate MCF cuts moderately
2	Generate MCF cuts aggressively

memoryemphasis *(boolean)*: reduces use of memory ↩

This parameter lets you indicate to Cplex that it should conserve memory where possible. When you set this parameter to its non default value, Cplex will choose tactics, such as data compression or disk storage, for some of the data computed by the barrier and MIP optimizers. Of course, conserving memory may impact performance in some models. Also, while solution information will be available after optimization, certain computations that require a basis that has been factored (for example, for the computation of the condition number Kappa) may be unavailable.

Default: 0

value	meaning
0	Do not conserve memory
1	Conserve memory where possible

mipdisplay *(integer)*: progress display level ↩

The amount of information displayed during MIP solution increases with increasing values of this option.

Default: 4

value	meaning
0	No display
1	Display integer feasible solutions
2	Displays nodes under mipinterval control
3	Same as 2 but adds information on cuts
4	Same as 3 but adds LP display for the root node
5	Same as 3 but adds LP display for all nodes

mipemphasis *(integer)*: MIP solution tactics ↩

This option controls the tactics for solving a mixed integer programming problem.

Default: 0

value	meaning
0	Balance optimality and feasibility
1	Emphasize feasibility over optimality
2	Emphasize optimality over feasibility
3	Emphasize moving the best bound
4	Emphasize hidden feasible solutions
5	Find high quality feasible solutions as early as possible

mipinterval *(integer)*: progress display interval ↩

Controls the frequency of node logging when the parameter MIPDisplay is set higher than 1 (one). Frequency must be an integer; it may be 0 (zero), positive, or negative. By default, CPLEX displays new information in the node log during a MIP solve at relatively high frequency during the early stages of solving a MIP model, and adds lines to the log at progressively longer intervals as solving continues. In other words, CPLEX logs information frequently in the beginning and progressively less often as it works. When the value is a positive integer n, CPLEX displays new incumbents, plus it displays a new line in the log every n nodes. When the value is a negative integer n, CPLEX displays new incumbents, and the negative value determines how much processing CPLEX does before it displays a new line in the node log. A negative value close to zero means that CPLEX displays new lines in the log frequently. A negative value far from zero means that CPLEX displays new lines in the log less frequently. In other words, a negative value of this parameter contracts or dilates the interval at which CPLEX displays information in the node log.

Default: 0

mipkappastats *(integer)*: MIP kappa computation ↩

MIP kappa summarizes the distribution of the condition number of the optimal bases CPLEX encountered during the solution of a MIP model. That summary may let you know more about the numerical difficulties of your MIP model. Because MIP kappa (as a statistical distribution) requires CPLEX to compute the condition number of the optimal bases of the subproblems during branch-and-cut search, you can compute the MIP kappa only when CPLEX solves the subproblem with its simplex optimizer. In other words, in order to obtain results with this parameter, you can not use the sifting optimizer nor the barrier without crossover to solve the subproblems. See the parameters StartAlg and SubAlg.

Computing the kappa of a subproblem has a cost. In fact, computing MIP kappa for the basis matrices can be computationally expensive and thus generally slows down the solution of a

problem. Therefore, the setting 0 (automatic) tells CPLEX generally not to compute MIP kappa, but in cases where the parameter NumericalEmphasis is turned on, CPLEX computes MIP kappa for a sample of subproblems. The value 1 (sample) leads to a negligible performance degradation on average, but can slow down the branch-and-cut exploration by as much as 10% on certain models. The value 2 (full) leads to a 2% performance degradation on average, but can significantly slow the branch-and-cut exploration on certain models. In practice, the value 1 (sample) is a good trade-off between performance and accuracy of statistics. If you need very accurate statistics, then use value 2 (full).

In case CPLEX is instructed to compute a MIP kappa distribution, the parameter Quality is automatically turned on.

Default: `0`

value	meaning
`-1`	No MIP kappa statistics; default
`0`	Automatic: let CPLEX decide
`1`	Compute MIP kappa for a sample of subproblems
`2`	Compute MIP kappa for all subproblems

mipordind *(boolean)*: priority list on/off ↩

Synonym: prioropt

Use priorities. Priorities should be assigned based on your knowledge of the problem. Variables with higher priorities will be branched upon before variables of lower priorities. This direction of the tree search can often dramatically reduce the number of nodes searched. For example, consider a problem with a binary variable representing a yes/no decision to build a factory, and other binary variables representing equipment selections within that factory. You would naturally want to explore whether or not the factory should be built before considering what specific equipment to purchased within the factory. By assigning a higher priority to the build/no build decision variable, you can force this logic into the tree search and eliminate wasted computation time exploring uninteresting portions of the tree. When set at 0 (default), the `MIPOrdInd` option instructs Cplex not to use priorities for branching. When set to 1, priority orders are utilized.

Note: Priorities are assigned to discrete variables using the .prior suffix in the GAMS model. Lower .prior values mean higher priority. The `.prioropt` model suffix has to be used to signal GAMS to export the priorities to the solver.

Default: `GAMS PriorOpt`

value	meaning
`0`	Do not use priorities for branching
`1`	Priority orders are utilized

mipordtype *(integer)*: priority order generation ↩

This option is used to select the type of generic priority order to generate when no priority order is present.

Default: `0`

value	meaning
0	None
1	decreasing cost magnitude
2	increasing bound range
3	increasing cost per coefficient count

mipsearch *(integer)*: search strategy for mixed integer programs ↩

Sets the search strategy for a mixed integer program. By default, Cplex chooses whether to apply dynamic search or conventional branch and cut based on characteristics of the model.

Default: 0

value	meaning
0	Automatic
1	Apply traditional branch and cut strategy
2	Apply dynamic search

mipstart *(integer)*: use mip starting values ↩

This option controls the use of advanced starting values for mixed integer programs. A setting of 2 indicates that the values should be checked to see if they provide an integer feasible solution before starting optimization. For `mipstart` equals 1, 2, 3 or 4 fractional values are rounded to the nearest integer value if the integrality violation is larger than CPLEX's integer tolerance and smaller or equal to tryint. A partial MIP start is applied for `mipstart` equals 1, 3 or 4. Here, for discrete variables only integer values (after possible rounding) are added to the advanced starting values.

Default: 0

value	meaning
0	do not use the values
1	set discrete variable values and use auto mipstart level
2	set all variable values and use check feasibility mipstart level
3	set discrete variable values and use solve fixed mipstart level
4	set discrete variable values and use solve sub-MIP mipstart level
5	set discrete variable values and use solve repair-MIP mipstart level
6	set discrete variable values and use no checks at all. Warning: CPLEX may accept infeasible points as solutions!

mipstopexpr *(string)*: stopping expression for branch and bound ↩

If the provided logical expression is true, the branch-and-bound is aborted. Supported values are: etalg, resusd, nodusd, objest, objval. Supported opertators are: +, -, ∗, /, ^, %, !=, ==, <, <=, >, >=, !, &&, ||, (,), abs, ceil, exp, floor, log, log10, pow, sqrt. Example:

```
nodusd >= 1000 && abs(objest - objval) / abs(objval) < 0.1
```

If multiple stop expressions are given in an option file, the algorithm stops if any of them is true (|| concatenation).

miptrace *(string)*: filename of MIP trace file ↩

For a description of this feature, see chapter Solve trace.

Note: In contrast to other solvers, GAMS/CPLEX doesn't append the MIP trace file after a certain time or node count, but when CPLEX reports global progress. In order to indicate this, the MIP trace file will show `X` instead of `N` or `T`.

miqcpstrat *(integer)*: MIQCP relaxation choice ↩

This option controls how MIQCPs are solved. For some models, the setting 2 may be more effective than 1. You may need to experiment with this parameter to determine the best setting for your model.

Default: `0`

value	meaning
`0`	Automatic
`1`	QCP relaxation Cplex will solve a QCP relaxation of the model at each node.
`2`	LP relaxation Cplex will solve a LP relaxation of the model at each node.

mircuts *(integer)*: mixed integer rounding cut generation ↩

Determines whether or not to generate mixed integer rounding (MIR) cuts during optimization. At the default of 0, generation is continued only if it seems to be helping.

Default: `0`

value	meaning
`-1`	Do not generate MIR cuts
`0`	Determined automatically
`1`	Generate MIR cuts moderately
`2`	Generate MIR cuts aggressively

mpslongnum *(boolean)*: MPS file format precision of numeric output ↩

Determines the precision of numeric output in the MPS file formats. When this parameter is set to its default value 1 (one), numbers are written to MPS files in full-precision; that is, up to 15 significant digits may be written. The setting 0 (zero) writes files that correspond to the standard MPS format, where at most 12 characters can be used to represent a value. This limit may result in loss of precision.

Default: `1`

value	meaning
`0`	Use limited MPS precision
`1`	Use full-precision

mtol *(real)*: basis identification dual tolerance ↩

Default: `0`

multimipstart *(string)*: use multiple mipstarts provided via gdx files ↩

Specifies (multiple) GDX files with values for the variables. Each file is treated as one intial guess for the MIP start. These MIP starts are added in addition to the initial guess provided by the level attribute. A MIP start GDX file can be created, for example, by using the command line option savepoint.

multobj *(boolean)*: controls the hierarchical optimization of multiple objectives ↩

Default: `0`

multobjdisplay *(integer)*: level of display during multiobjective optimization ↩

Default: `1`

value	meaning
0	No display
1	Summary display after each subproblem
2	Summary display after each subproblem, as well as subproblem logs

multobjmethod *(integer)*: method used for multi-objective solves ↩

When solving a continuous multi-objective model using a hierarchical approach, the model is solved once for each objective. The algorithm used to solve for the highest priority objective is controlled by the LPMethod parameter. This parameter determines the algorithm used to solve for subsequent objectives.

Default: `0`

multobjoptfiles *(string)*: List of option files used for individual solves within multi-objective optimization ↩

The options given by the option files in multobjoptfiles are applied on top of the default GAMS/CPLEX options. This includes options set by the user via the standard option file.

If the list of option files in multobjoptfiles is less than the number of objective functions, the default GAMS/CPLEX options (incl. user options as before) are used to solve the remaining instances. Additional option files (i.e. more than objective functions) are ignored.

Applied options can be verified by setting multobjdisplay to 2.

multobjtolmip *(boolean)*: enables hard constraints for hierarchical optimization objectives based on degradation tolerances ↩

CPLEX supports two different strategies to handle the degradation tolerances objnabstol and objnreltol depending on the problem type (continuous or discrete), see objnabstol. This setting enables the discrete strategy for continous models. Note that objnreltol has no effect for discrete models. Enabling this option can lead to higher solution times.

Default: `1`

names *(boolean)*: load GAMS names into Cplex ↩

This option causes GAMS names for the variables and equations to be loaded into Cplex. These names will then be used for error messages, log entries, and so forth. Setting names to no may help if memory is very tight.

Default: `1`

netdisplay *(integer)*: network display level ↩

This option controls the log for network iterations.

Default: `2`

value	meaning
0	No network log.
1	Displays true objective values
2	Displays penalized objective values

netepopt *(real)*: optimality tolerance for the network simplex method ↩

This optimality tolerance influences the reduced-cost tolerance for optimality when using the network simplex method. This option setting governs how closely Cplex must approach the theoretically optimal solution.

Range: [1.0e-11, 0.1]

Default: 1.0e-06

neteprhs *(real)*: feasibility tolerance for the network simplex method ↩

This feasibility tolerance determines the degree to which the network simplex algorithm will allow a flow value to violate its bounds.

Range: [1.0e-11, 0.1]

Default: 1.0e-06

netfind *(integer)*: attempt network extraction ↩

Specifies the level of network extraction to be done.

Default: 2

value	meaning
1	Extract pure network only
2	Try reflection scaling
3	Try general scaling

netitlim *(integer)*: iteration limit for network simplex ↩

Iteration limit for the network simplex method.

Default: large

netppriind *(integer)*: network simplex pricing ↩

Network simplex pricing algorithm. The default of 0 (currently equivalent to 3) shows best performance for most problems.

Default: 0

value	meaning
0	Automatic
1	Partial pricing
2	Multiple partial pricing
3	Multiple partial pricing with sorting

nodecuts *(integer)*: decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree ↩

Default: `0`

nodefileind *(integer)*: node storage file indicator ↩

Specifies how node files are handled during MIP processing. Used when parameter WorkMem has been exceeded by the size of the branch and cut tree. If set to 0 when the tree memory limit is reached, optimization is terminated. Otherwise a group of nodes is removed from the in-memory set as needed. By default, Cplex transfers nodes to node files when the in-memory set is larger than 128 MBytes, and it keeps the resulting node *files* in compressed form in memory. At settings 2 and 3, the node files are transferred to disk. They are stored under a directory specified by parameter WorkDir and Cplex actively manages which nodes remain in memory for processing.

Default: `1`

value	meaning
`0`	No node files
`1`	Node files in memory and compressed
`2`	Node files on disk
`3`	Node files on disk and compressed

nodelim *(integer)*: maximum number of nodes to solve ↩

Synonym: nodlim

The maximum number of nodes solved before the algorithm terminates, without reaching optimality. This option overrides the GAMS NodLim model suffix. When this parameter is set to 0 (this is only possible through an option file), Cplex completes processing at the root; that is, it creates cuts and applies heuristics at the root. When this parameter is set to 1 (one), it allows branching from the root; that is, nodes are created but not solved.

Default: `GAMS NodLim`

nodesel *(integer)*: node selection strategy ↩

This option is used to set the rule for selecting the next node to process when backtracking.

Default: `1`

value	meaning
`0`	Depth-first search This chooses the most recently created node.
`1`	Best-bound search This chooses the unprocessed node with the best objective function for the associated LP relaxation.
`2`	Best-estimate search This chooses the node with the best estimate of the integer objective value that would be obtained once all integer infeasibilities are removed.
`3`	Alternate best-estimate search

numericalemphasis *(boolean)*: emphasizes precision in numerically unstable or difficult problems ↩

This parameter lets you indicate to Cplex that it should emphasize precision in numerically difficult or unstable problems, with consequent performance trade-offs in time and memory.

Default: `0`

value	meaning
`0`	Off
`1`	Exercise extreme caution in computation

objdif *(real)*: overrides GAMS Cheat parameter ↩

Synonym: cheat

A means for automatically updating the cutoff to more restrictive values. Normally the most recently found integer feasible solution objective value is used as the cutoff for subsequent nodes. When this option is set to a positive value, the value will be subtracted from (added to) the newly found integer objective value when minimizing (maximizing). This forces the MIP optimization to ignore integer solutions that are not at least this amount better than the one found so far. The option can be adjusted to improve problem solving efficiency by limiting the number of nodes; however, setting this option at a value other than zero (the default) can cause some integer solutions, including the true integer optimum, to be missed. Negative values for this option will result in some integer solutions that are worse than or the same as those previously generated, but will not necessarily result in the generation of all possible integer solutions. This option overrides the GAMS Cheat parameter.

Default: `0.0`

objllim *(real)*: objective function lower limit ↩

Setting a lower objective function limit will cause Cplex to halt the optimization process once the minimum objective function value limit has been exceeded.

Default: `-1.0e+75`

objnabstol *(string)*: allowable absolute degradation for objective ↩

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization (MultObj). The syntax for this parameter is `ObjNAbsTol ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. For MIPs (or if MultObjTolMip is enabled), if it achieves objective value z when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most `ObjNAbsTol`. For LPs, `ObjNAbsTol` defines a threshold for reduced costs above which nonbasic variables in the associated LP solve will be fixed at the bound at which they reside.

objnreltol *(string)*: allowable relative degradation for objective ↩

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization (MultObj). The syntax for this parameter is `ObjNRelTol ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. For MIPs (or if MultObjTolMip is enabled), if it achieves objective value z when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most `ObjNRelTol`$*|z|$. This option has no effect for continuous models.

objrng *(string)*: do objective ranging ↩

Calculate sensitivity ranges for the specified GAMS variables. Unlike most options, `ObjRng` can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS variable named. Specifying `all` will cause range information to be produced for all variables. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option RngRestart is specified.

Default: `no objective ranging is done`

objulim *(real)*: objective function upper limit ↩

Setting an upper objective function limit will cause Cplex to halt the optimization process once the maximum objective function value limit has been exceeded.

Default: `1.0e+75`

optimalitytarget *(integer)*: type of optimality that Cplex targets ↩

This parameter specifies the type of solution that CPLEX attempts to compute with respect to the optimality of that solution when CPLEX solves a continuous (QP) or mixed integer (MIQP) quadratic model. In other words, the variables of the model can be continuous or mixed integer and continuous; the objective function includes a quadratic term, and perhaps the objective function is not positive semi-definite (non PSD). This parameter does not apply to quadratically constrained mixed integer problems (MIQCP); that is, this parameter does not apply to mixed integer problems that include a quadratic term among the constraints.

Default: `0`

value	meaning
`0`	Automatic CPLEX first attempts to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX will return with an error (Q is not PSD).
`1`	Search for a globally optimal solution to a convex model CPLEX searches for a globally optimal solution to a convex model. In problems of type QP or MIQP, this setting interacts with linearization switch QToLin for QP, MIQP
`2`	Search for a solution that satisfies first-order optimality conditions no optimality guarantee CPLEX first attempt to compute a provably optimal solution. If CPLEX cannot compute a provably optimal solution because the objective function is not convex, CPLEX searches for a solution that satisfies first-order optimality conditions but is not necessarily globally optimal.
`3`	Search for a globally optimal solution regardless of convexity If the problem type is QP, CPLEX first changes the problem type to MIQP. CPLEX then solves the problem (whether originally QP or MIQP) to global optimality. In problems of type QP or MIQP, this setting interacts with with linearization switch QToLin for QP, MIQP. With this setting information about dual values is not available for the solution.

parallelmode *(integer)*: parallel optimization mode ↩

Sets the parallel optimization mode. Possible modes are automatic, deterministic, and opportunistic.

In this context, deterministic means that multiple runs with the same model at the same parameter settings on the same platform will reproduce the same solution path and results. In contrast, opportunistic implies that even slight differences in timing among threads or in the order in which tasks are executed in different threads may produce a different solution path and consequently different timings or different solution vectors during optimization executed in parallel threads. When running with multiple threads, the opportunistic setting entails less synchronization between threads and consequently may provide better performance.

In deterministic mode, Cplex applies as much parallelism as possible while still achieving deterministic results. That is, when you run the same model twice on the same platform with the same parameter settings, you will see the same solution and optimization run.

More opportunities to exploit parallelism are available if you do not require determinism. In other words, Cplex can find more opportunities for parallelism if you do not require an invariant, repeatable solution path and precisely the same solution vector. To use all available parallelism, you need to select the opportunistic parallel mode. In this mode, Cplex will utilize all opportunities for parallelism in order to achieve best performance.

However, in opportunistic mode, the actual optimization may differ from run to run, including the solution time itself and the path traveled in the search.

Parallel MIP optimization can be opportunistic or deterministic.

Parallel barrier optimization is only deterministic.

A GAMS/Cplex run will use deterministic mode unless explicitly specified.

If `ParallelMode` is explicitly set to 0 (automatic) the settings of this parallel mode parameter interact with settings of the Threads parameter. Let the result number of threads available to Cplex be n (note that negative values for the threads parameter are possible to exclude work on some cores).

Here is is list of possible value:

Default: `1`

value	meaning
`-1`	Enable opportunistic parallel search mode
`0`	Automatic
`1`	Enable deterministic parallel search mode

perind *(boolean)*: force initial perturbation ↩

Perturbation Indicator. If a problem automatically perturbs early in the solution process, consider starting the solution process with a perturbation by setting `PerInd` to 1. Manually perturbing the problem will save the time of first allowing the optimization to stall before activating the perturbation mechanism, but is useful only rarely, for extremely degenerate problems.

Default: `0`

value	meaning
`0`	not automatically perturbed
`1`	automatically perturbed

perlim *(integer)*: number of stalled iterations before perturbation ↩

Perturbation limit. The number of stalled iterations before perturbation is invoked. The default value of 0 means the number is determined automatically.

Default: `0`

polishafterdettime *(real)*: deterministic time before starting to polish a feasible solution ↩

Default: `1.0e+75`

polishafterepagap *(real)*: absolute MIP gap before starting to polish a feasible solution ↩

Solution polishing can yield better solutions in situations where good solutions are otherwise hard to find. More time-intensive than other heuristics, solution polishing is actually a variety of branch-and-cut that works after an initial solution is available. In fact, it requires a solution to be available for polishing, either a solution produced by branch-and-cut, or a MIP start supplied by a user. Because of the high cost entailed by solution polishing, it is not called throughout branch-and-cut like other heuristics. Instead, solution polishing works in a second phase after a first phase of conventional branch-and-cut. As an additional step after branch-and-cut, solution polishing can improve the best known solution. As a kind of branch-and-cut algorithm itself, solution polishing focuses solely on finding better solutions. Consequently, it may not prove optimality, even if the optimal solution has indeed been found. Like the RINS heuristic, solution polishing explores neighborhoods of previously found solutions by solving subMIPs.

Sets an absolute MIP gap (that is, the difference between the best integer objective and the objective of the best node remaining) after which CPLEX stops branch-and-cut and begins polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: `0.0`

polishafterepgap *(real)*: relative MIP gap before starting to polish a solution ↩

Sets a relative MIP gap after which CPLEX will stop branch-and-cut and begin polishing a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: `0.0`

polishafterintsol *(integer)*: MIP integer solutions to find before starting to polish a feasible solution ↩

Sets the number of integer solutions to find before CPLEX stops branch-and-cut and begins to polish a feasible solution. The default value is such that CPLEX does not invoke solution polishing by default.

Default: `2147483647`

polishafternode *(integer)*: nodes to process before starting to polish a feasible solution ↩

Sets the number of nodes processed in branch-and-cut before CPLEX starts solution polishing, if a feasible solution is available.

Default: `2147483647`

polishaftertime *(real)*: time before starting to polish a feasible solution ↩

Tells CPLEX how much time in seconds to spend during mixed integer optimization before CPLEX starts polishing a feasible solution. The default value is such that CPLEX does not start solution polishing by default.

Default: `1.0e+75`

populatelim *(integer)*: limit of solutions generated for the solution pool by populate method ↩

Limits the number of solutions generated for the solution pool during each call to the populate procedure. Populate stops when it has generated *PopulateLim* solutions. A solution is counted if it is valid for all filters (see DivFlt and consistent with the relative and absolute pool gap parameters (see SolnPoolGap and SolnPoolAGap), and has not been rejected by the incumbent checking routine (see UserIncbCall), whether or not it improves the objective of the model. This parameter does not apply to MIP optimization generally; it applies only to the populate procedure.

If you are looking for a parameter to control the number of solutions stored in the solution pool, consider the parameter SolnPoolCapacity instead.

Populate will stop before it reaches the limit set by this parameter if it reaches another limit, such as a time or node limit set by the user.

Default: `20`

ppriind *(integer)*: primal simplex pricing ↩

Pricing algorithm. Likely to show the biggest impact on performance. Look at overall solution time and the number of Phase I and total iterations as a guide in selecting alternate pricing algorithms. If you are using the dual Simplex method use DPriInd to select a pricing algorithm. If the number of iterations required to solve your problem is approximately the same as the number of rows in your problem, then you are doing well. Iteration counts more than three times greater than the number of rows suggest that improvements might be possible.

Default: `0`

value	meaning
`-1`	Reduced-cost pricing This is less compute intensive and may be preferred if the problem is small or easy. This option may also be advantageous for dense problems (say 20 to 30 nonzeros per column).
`0`	Hybrid reduced-cost and Devex pricing
`1`	Devex pricing This may be useful for more difficult problems which take many iterations to complete Phase I. Each iteration may consume more time, but the reduced number of total iterations may lead to an overall reduction in time. Tenfold iteration count reductions leading to threefold speed improvements have been observed. Do not use devex pricing if the problem has many columns and relatively few rows. The number of calculations required per iteration will usually be disadvantageous.
`2`	Steepest edge pricing If devex pricing helps, this option may be beneficial. Steepest-edge pricing is computationally expensive, but may produce the best results on exceptionally difficult problems.
`3`	Steepest edge pricing with slack initial norms This reduces the computationally intensive nature of steepest edge pricing.
`4`	Full pricing

predual *(integer)*: give dual problem to the optimizer ↩

Solve the dual. Some linear programs with many more rows than columns may be solved faster by explicitly solving the dual. The `PreDual` option will cause Cplex to solve the dual while returning the solution in the context of the original problem. This option is ignored if presolve is turned off.

Default: `0`

value	meaning
`-1`	do not give dual to optimizer
`0`	automatic
`1`	give dual to optimizer

preind *(boolean)*: turn presolver on/off ↩

Perform Presolve. This helps most problems by simplifying, reducing and eliminating redundancies. However, if there are no redundancies or opportunities for simplification in the model, if may be faster to turn presolve off to avoid this step. On rare occasions, the presolved model, although smaller, may be more difficult than the original problem. In this case turning the presolve off leads to better performance. Specifying 0 turns the aggregator off as well.

Default: `1`

prepass *(integer)*: number of presolve applications to perform ↩

Number of MIP presolve applications to perform. By default, Cplex determines this automatically. Specifying 0 turns off the presolve but not the aggregator. Set PreInd to 0 to turn both off.

Default: `-1`

value	meaning
`-1`	Determined automatically
`0`	No presolve
`>0`	Number of MIP presolve applications to perform

prereform *(integer)*: set presolve reformulations ↩

Default: `3`

preslvnd *(integer)*: node presolve selector ↩

Indicates whether node presolve should be performed at the nodes of a mixed integer programming solution. Node presolve can significantly reduce solution time for some models. The default setting is generally effective.

Default: `0`

value	meaning
`-1`	No node presolve
`0`	Automatic
`1`	Force node presolve
`2`	Perform probing on integer-infeasible variables
`3`	Perform aggressive node probing

pricelim *(integer)*: pricing candidate list ↩

Size for the pricing candidate list. Cplex dynamically determines a good value based on problem dimensions. Only very rarely will setting this option manually improve performance. Any non-negative integer values are valid.

Default: `0, in which case it is determined automatically`

printoptions *(boolean)*: list values of all options to GAMS listing file ↩

Write the values of all options to the GAMS listing file. Valid values are no or yes.

Default: `0`

probe *(integer)*: perform probing before solving a MIP ↩

Determines the amount of probing performed on a MIP. Probing can be both very powerful and very time consuming. Setting the value to 1 can result in dramatic reductions or dramatic increases in solution time depending on the particular model.

Default: `0`

value	meaning
`-1`	No probing
`0`	Automatic
`1`	Limited probing
`2`	More probing
`3`	Full probing

probedettime *(real)*: deterministic time spent probing ↩

Default: `1.0e+75`

probetime *(real)*: time spent probing ↩

Limits the amount of time in seconds spent probing.

Default: `1.0e+75`

qextractalg *(integer)*: quadratic extraction algorithm in GAMS interface ↩

Default: `0`

value	meaning
`0`	Automatic
`1`	ThreePass: Uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage.
`2`	DoubleForward: Uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.
`3`	Concurrent: Uses ThreePass and DoubleForward in parallel. As soon as one finishes, the other one stops.

qpmakepsdind *(boolean)*: adjust MIQP formulation to make the quadratic matrix positive-semi-definite

↩

Determines whether Cplex will attempt to adjust a MIQP formulation, in which all the variables appearing in the quadratic term are binary. When this feature is active, adjustments will be made to the elements of a quadratic matrix that is not nominally positive semi-definite (*PSD*, as required by Cplex for all QP formulations), to make it PSD, and will also attempt to tighten an already PSD matrix for better numerical behavior. The default setting of 1 means `yes` but you can turn it off if necessary; most models should benefit from the default setting.

Default: `1`

value	meaning
`0`	Off
`1`	On

qpmethod *(integer)*: algorithm to be used for QP problems ↩

Specifies which QP algorithm to use.

At the default of 0 (automatic), barrier is used for QP problems and dual simplex for the root relaxation of MIQP problems.

Default: `0`

value	meaning
`0`	Automatic
`1`	Primal Simplex
`2`	Dual Simplex
`3`	Network Simplex
`4`	Barrier
`5`	Sifting
`6`	Concurrent dual, barrier, and primal

qtolin *(integer)*: linearization of the quadratic terms in the objective function of a QP or MIQP model ↩

This parameter switches on or off linearization of the quadratic terms in the objective function of a quadratic program or of a mixed integer quadratic program.

In a convex mixed integer quadratic program, this parameter controls whether Cplex linearizes the product of binary variables in the objective function during presolve. In a nonconvex quadratic program or mixed integer quadratic program solved to global optimality according to OptimalityTarget, this parameter controls how Cplex linearizes the product of bounded variables in the objective function during presolve.

This parameter interacts with the existing parameter OptimalityTarget: When the solution target type is set to 1 (that is, Cplex searches for a globally optimal solution to a convex model), then in a convex MIQP, this parameter tells Cplex to replace the product of a binary variable and a bounded linear variable by a linearly constrained variable. When the solution target type is set to 3, then in a nonconvex QP or nonconvex MIQP, this parameter controls the initial relaxation.

Default: `-1`

value	meaning
-1	Automatic
0	Off, Cplex does not linearize quadratic terms in the objective
1	On, Cplex linearizes quadratic terms in the objective

quality *(boolean)*: write solution quality statistics ↩

Write solution quality statistics to the listing and log file. If set to yes, the statistics appear after the Solve Summary and before the Solution Listing and contain information about infeasibility levels, solution value magnitued, and the condition number (kappa):

```
Solution Quality Statistics:
                                  unscaled                scaled
                                max         sum         max         sum
primal infeasibility        0.000e+00   0.000e+00   0.000e+00   0.000e+00
dual infeasibility          0.000e+00   0.000e+00   0.000e+00   0.000e+00
primal residual             0.000e+00   0.000e+00   0.000e+00   0.000e+00
dual residual               0.000e+00   0.000e+00   0.000e+00   0.000e+00
primal solution vector      3.000e+02   9.000e+02   3.000e+02   9.000e+02
dual solution vector        1.000e+00   1.504e+00   1.000e+00   1.504e+00
slacks                      5.000e+01   5.000e+01   5.000e+01   5.000e+01
reduced costs               3.600e-02   4.500e-02   3.600e-02   4.500e-02

Condition number of the scaled basis matrix =    9.000e+00
```

Default: 0

randomseed *(integer)*: sets the random seed differently for diversity of solutions ↩

Default: changes with each Cplex release

readflt *(string)*: reads Cplex solution pool filter file ↩

The GAMS/Cplex solution pool options cover the basic use of diversity and range filters for producing multiple solutions. If you need multiple filters, weights on diversity filters or other advanced uses of solution pool filters, you could produce a Cplex filter file with your favorite editor or the GAMS Put Facility and read this into GAMS/Cplex using this option.

readparams *(string)*: read Cplex parameter file ↩

reduce *(integer)*: primal and dual reduction type ↩

Determines whether primal reductions, dual reductions, or both, are performed during preprocessing. It is occasionally advisable to do only one or the other when diagnosing infeasible or unbounded models.

Default: 3

value	meaning
0	No primal or dual reductions
1	Only primal reductions
2	Only dual reductions
3	Both primal and dual reductions

reinv *(integer)*: refactorization frequency ↩

Refactorization Frequency. This option determines the number of iterations between refactorizations of the basis matrix. The default should be optimal for most problems. Cplex's performance is relatively insensitive to changes in refactorization frequency. Only for extremely large, difficult problems should reducing the number of iterations between refactorizations be considered. Any non-negative integer value is valid.

Default: `0, in which case it is determined automatically`

relaxfixedinfeas *(boolean)*: accept small infeasibilties in the solve of the fixed problem ↩

Sometimes the solution of the fixed problem of a MIP does not solve to optimality due to small (dual) infeasibilities. The default behavior of the GAMS/Cplex link is to return the primal solution values only. If the option is set to 1, the small infeasibilities are ignored and a full solution including the dual values are reported back to GAMS.

Default: `0`

value	meaning
0	Off
1	On

relaxpreind *(integer)*: presolve for initial relaxation on/off ↩

This option will cause the Cplex presolve to be invoked for the initial relaxation of a mixed integer program (according to the other presolve option settings). Sometimes, additional reductions can be made beyond any MIP presolve reductions that may already have been done.

Default: `-1`

value	meaning
-1	Automatic
0	do not presolve initial relaxation
1	use presolve on initial relaxation

relobjdif *(real)*: relative cheat parameter ↩

The relative version of the ObjDif option. Ignored if ObjDif is non-zero.

Default: `0.0`

repairtries *(integer)*: try to repair infeasible MIP start ↩

This parameter lets you indicate to Cplex whether and how many times it should try to repair an infeasible MIP start that you supplied. The parameter has no effect if the MIP start you supplied is feasible. It has no effect if no MIP start was supplied.

Default: `0`

value	meaning
-1	None: do not try to repair
0	Automatic
>0	Maximum tries to perform

repeatpresolve *(integer)*: reapply presolve at root after preprocessing ↩

This integer parameter tells Cplex whether to re-apply presolve, with or without cuts, to a MIP model after processing at the root is otherwise complete.

Default: `-1`

value	meaning
`-1`	Automatic
`0`	Turn off represolve
`1`	Represolve without cuts
`2`	Represolve with cuts
`3`	Represolve with cuts and allow new root cuts

rerun *(string)*: rerun problem if presolve infeasible or unbounded ↩

The Cplex presolve can sometimes diagnose a problem as being infeasible or unbounded. When this happens, GAMS/Cplex can, in order to get better diagnostic information, rerun the problem with presolve turned off. The GAMS solution listing will then mark variables and equations as infeasible or unbounded according to the final solution returned by the simplex algorithm. The IIS option can be used to get even more diagnostic information. The rerun option controls this behavior. Valid values are `auto`, `yes`, `no` and `nono`. The value of auto is equivalent to no if names are successfully loaded into Cplex and option IIS is set to no. In that case the Cplex messages from presolve help identify the cause of infeasibility or unboundedness in terms of GAMS variable and equation names. If names are not successfully loaded, rerun defaults to yes. Loading of GAMS names into Cplex is controlled by option Names. The value of `nono` only affects MIP models for which Cplex finds a feasible solution in the branch-and-bound tree but the fixed problem turns out to be infeasible. In this case the value `nono` also disables the rerun without presolve, while the value of no still tries this run. Feasible integer solution but an infeasible fixed problem happens in few cases and mostly with badly scaled models. If you experience this try more aggressive scaling (ScaInd) or tightening the integer feasibility tolerance EPInt. If the fixed model is infeasible only the primal solution is returned to GAMS. You can recognize this inside GAMS by checking the marginal of the objective defining constraint which is always nonzero.

Default: `nono`

value	meaning
`auto`	Automatic
`yes`	Rerun infeasible models with presolve turned off
`no`	Do not rerun infeasible models
`nono`	Do not rerun infeasible fixed MIP models

rhsrng *(string)*: do right-hand-side ranging ↩

Calculate sensitivity ranges for the specified GAMS equations. Unlike most options, `RHSRng` can be repeated multiple times in the options file. Sensitivity range information will be produced for each GAMS equation named. Specifying `all` will cause range information to be produced for all equations. Range information will be printed to the beginning of the solution listing in the GAMS listing file unless option RngRestart is specified.

Default: `no right-hand-side ranging is done`

rinsheur *(integer)*: relaxation induced neighborhood search frequency ↩

Cplex implements a heuristic known a Relaxation Induced Neighborhood Search (RINS) for MIP and MIQCP problems. RINS explores a neighborhood of the current incumbent to try to find a new, improved incumbent. It formulates the neighborhood exploration as a MIP, a subproblem known as the subMIP, and truncates the subMIP solution by limiting the number of nodes explored in the search tree.

Parameter `RINSHeur` controls how often RINS is invoked. A value of 100, for example, means that RINS is invoked every hundredth node in the tree.

Default: `0`

value	meaning
`-1`	Disable RINS
`0`	Automatic
`>0`	Call RINS at the requested node interval

rltcuts *(integer)*: Reformulation Linearization Technique (RLT) cuts ↩

This parameter controls the addition of cuts based on the Reformulation Linearization Technique (RLT) for nonconvex quadratic programs (QP) or mixed integer quadratic programs (MIQP) solved to global optimality. That is, the OptimalityTarget parameter must be set to 3. The `RLTCuts` option is not controlled by the option Cuts.

Default: `0`

value	meaning
`-1`	Do not generate RLT cuts
`0`	Determined automatically
`1`	Generate RLT cuts moderately
`2`	Generate RLT cuts aggressively
`3`	Generate RLT cuts very aggressively

rngrestart *(string)*: write GAMS readable ranging information file ↩

Write ranging information, in GAMS readable format, to the file named. If the file extension is GDX, the ranging information is exported as GDX file. Options ObjRng and RHSRng are used to specify which GAMS variables or equations are included.

Default: `ranging information is printed to the listing file`

scaind *(integer)*: matrix scaling on/off ↩

This option influences the scaling of the problem matrix.

Default: `0`

value	meaning
`-1`	No scaling
`0`	Standard scaling An equilibration scaling method is implemented which is generally very effective.
`1`	Modified, more aggressive scaling method This method can produce improvements on some problems. This scaling should be used if the problem is observed to have difficulty staying feasible during the solution process.

siftalg *(integer)*: sifting subproblem algorithm ↩

Sets the algorithm to be used for solving sifting subproblems.

Default: `0`

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex
3	Network simplex
4	Barrier

siftdisplay *(integer)*: sifting display level ↩

Determines the amount of sifting progress information to be displayed.

Default: `1`

value	meaning
0	No display
1	Display major iterations
2	Display LP subproblem information

sifting *(boolean)*: switch for sifting from simplex optimization ↩

Default: `1`

siftitlim *(integer)*: limit on sifting iterations ↩

Sets the maximum number of sifting iterations that may be performed if convergence to optimality has not been reached.

Default: `large`

simdisplay *(integer)*: simplex display level ↩

This option controls what Cplex reports (normally to the screen) during optimization. The amount of information displayed increases as the setting value increases.

Default: `1`

value	meaning
0	No iteration messages are issued until the optimal solution is reported
1	An iteration log message will be issued after each refactorization Each entry will contain the iteration count and scaled infeasibility or objective value.
2	An iteration log message will be issued after each iteration The variables, slacks and artificials entering and leaving the basis will also be reported.

singlim *(integer)*: limit on singularity repairs ↩

The singularity limit setting restricts the number of times Cplex will attempt to repair the basis when singularities are encountered. Once the limit is exceeded, Cplex replaces the current basis with the best factorizable basis that has been found. Any non-negative integer value is valid.

Default: `10`

solnpool *(string)*: solution pool file name ↩

The solution pool enables you to generate and store multiple solutions to a MIP problem. The option expects a GDX filename. This GDX file name contains the information about the different solutions generated by Cplex. Inside your GAMS program you can process the GDX file and read the different solution point files. Please check the GAMS/Cplex solver guide document and the example model `solnpool.gms` from the GAMS model library.

solnpoolagap *(real)*: absolute tolerance for the solutions in the solution pool ↩

Sets an absolute tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the objective of the incumbent solution according to this measure are not kept in the solution pool.

Values of the solution pool absolute gap and the solution pool relative gap SolnPoolGap may differ: For example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and also within 1% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if it is valid for both the relative and the absolute gaps.

The solution pool absolute gap parameter can also be used as a stopping criterion for the populate procedure: if populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is reached.

Default: `1.0e+75`

solnpoolcapacity *(integer)*: limits of solutions kept in the solution pool ↩

Limits the number of solutions kept in the solution pool. At most, `SolnPoolCapacity` solutions will be stored in the pool. Superfluous solutions are managed according to the replacement strategy set by the solution pool replacement parameter SolnPoolReplace.

The optimization (whether by MIP optimization or the populate procedure) will not stop if more than `SolnPoolCapacity` are generated. Instead, stopping criteria are regular node and time limits and PopulateLim, SolnPoolGap and SolnPoolAGap.

Default: `2100000000`

solnpoolgap *(real)*: relative tolerance for the solutions in the solution pool ↩

Sets a relative tolerance on the objective bound for the solutions in the solution pool. Solutions that are worse (either greater in the case of a minimization, or less in the case of a maximization) than the incumbent solution by this measure are not kept in the solution pool.

Values of the solution pool absolute gap SolnPoolAGap and the solution pool relative gap may differ: For example, you may specify that solutions must be within 15 units by means of the solution pool absolute gap and within 1% of the incumbent by means of the solution pool relative gap. A solution is accepted in the pool only if it is valid for both the relative and the absolute gaps.

The solution pool relative gap parameter can also be used as a stopping criterion for the populate procedure: if populate cannot enumerate any more solutions that satisfy this objective quality, then it will stop. In the presence of both an absolute and a relative solution pool gap parameter, populate will stop when the smaller of the two is reached.

Default: `1.0e+75`

solnpoolintensity *(integer)*: solution pool intensity for ability to produce multiple solutions ↩

Controls the trade-off between the number of solutions generated for the solution pool and the amount of time or memory consumed. This parameter applies both to MIP optimization and to the populate procedure.

Values from 1 to 4 invoke increasing effort to find larger numbers of solutions. Higher values are more expensive in terms of time and memory but are likely to yield more solutions.

Default: `0`

value	meaning
`0`	Automatic Its default value, 0 , lets Cplex choose which intensity to apply.
`1`	Mild: generate few solutions quickly For value 1, the performance of MIP optimization is not affected. There is no slowdown and no additional consumption of memory due to this setting. However, populate will quickly generate only a small number of solutions. Generating more than a few solutions with this setting will be slow. When you are looking for a larger number of solutions, use a higher value of this parameter.
`2`	Moderate: generate a larger number of solutions For value 2, some information is stored in the branch and cut tree so that it is easier to generate a larger number of solutions. This storage has an impact on memory used but does not lead to a slowdown in the performance of MIP optimization. With this value, calling populate is likely to yield a number of solutions large enough for most purposes. This value is a good choice for most models.
`3`	Aggressive: generate many solutions and expect performance penalty For value 3, the algorithm is more aggressive in computing and storing information in order to generate a large number of solutions. Compared to values 1 and 2, this value will generate a larger number of solutions, but it will slow MIP optimization and increase memory consumption. Use this value only if setting this parameter to 2 does not generate enough solutions.
`4`	Very aggressive: enumerate all practical solutions For value 4, the algorithm generates all solutions to your model. Even for small models, the number of possible solutions is likely to be huge; thus enumerating all of them will take time and consume a large quantity of memory.

solnpoolmerge *(string)*: solution pool file name for merged solutions ↩

Similar to `solnpool` this option enables you to generate and store multiple solutions to a MIP problem. The option expects a GDX filename. This GDX file contains all variables with an additional first index (determined through SolnPoolPrefix) as parameters (Cplex only reports the primal solution). Inside your GAMS program you can process the GDX file and read all solutions in one read operation. Please check the GAMS/Cplex solver guide document for further solution pool options and the example model `solmpool.gms` from the GAMS model library.

solnpoolnumsym *(integer)*: maximum number of variable symbols when writing merged solutions ↩

Default: `10`

solnpoolpop *(integer)*: methods to populate the solution pool ↩

Regular MIP optimization automatically adds incumbents to the solution pool as they are discovered. Cplex also provides a procedure known as *populate* specifically to generate multiple solutions. You can invoke this procedure either as an alternative to the usual MIP optimizer or as a successor to the MIP optimizer. You can also invoke this procedure many times in a row in order to explore the solution space differently (see option SolnPoolPopRepeat). In particular, you may invoke this procedure multiple times to find additional solutions, especially if the first solutions found are not satisfactory.

Default: `1`

value	meaning
1	Just collect the incumbents found during regular optimization
2	Calls the populate procedure

solnpoolpopdel *(string)*: file with solution numbers to delete from the solution pool ↩

After the GAMS program specified in SolnPoolPopRepeat determined to continue the search for alternative solutions, the file specified by this option is read in. The solution numbers present in this file will be delete from the solution pool before the populate routine is called again. The file is automatically deleted by the GAMS/Cplex link after processing.

solnpoolpoprepeat *(string)*: method to decide if populating the solution should be repeated ↩

After the termination of the populate procedure (see option SolnPoolPop). The GAMS program specified in this option will be called which can examine the solutions in the solution pool and can decide to run the populate procedure again. If the GAMS program terminates normally (not compilation or execution time error) the search for new alternative solutions will be repeated.

solnpoolprefix *(string)*: file name prefix for GDX solution files ↩

Default: `soln`

solnpoolreplace *(integer)*: strategy for replacing a solution in the solution pool ↩

Default: `0`

value	meaning
0	Replace the first solution (oldest) by the most recent solution; first in, first out
1	Replace the solution which has the worst objective
2	Replace solutions in order to build a set of diverse solutions

solutiontype *(integer)*: type of solution (basic or non basic) for an LP or QP ↩

Specifies the type of solution (basic or non basic) that CPLEX attempts to compute for a linear program (LP) or for a quadratic program (QP). In this context, basic means having to do with the basis, and non basic applies to the variables and constraints not participating in the basis.

By default (that is, when the value of this parameter is 0 (zero) automatic), CPLEX seeks a basic solution (that is, a solution with a basis) for all linear programs (LP) and for all quadratic programs (QP).

When the value of this parameter is 1 (one), CPLEX seeks a basic solution, that is, a solution that includes a basis with a basic status for variables and constraints. In other words, CPLEX behaves the same way for the values 0 (zero) and 1 (one) of this parameter.

When the value of this parameter is 2, CPLEX seeks a pair of primal-dual solution vectors. This setting does not prevent CPLEX from producing status information, but in seeking a pair of primal-dual solution vectors, CPLEX possibly may not produce basic status information; that is, it is possible that CPLEX does not produce status information about which variables and constraints participate in the basis at this setting.

Do not use the deprecated value -1 (minus one) of the parameter barrier `crossover` algorithm to turn off crossover of the barrier algorithm but use this parameter to indicate that a primal-dual pair is sufficient.

Default: `0`

value	meaning
0	Automatic
1	Basic solution
2	primal-dual pair

solvefinal *(boolean)*: switch to solve the problem with fixed discrete variables ↩

Sometimes the solution process after the branch-and-cut that solves the problem with fixed discrete variables takes a long time and the user is interested in the primal values of the solution only. In these cases, `solvefinal` can be used to turn this final solve off. Without the final solve no proper marginal values are available and only NAs are returned to GAMS.

Default: `1`

value	meaning
0	Do not solve the fixed problem
1	Solve the fixed problem and return duals

sos1reform *(integer)*: automatic logarithmic reformulation of special ordered sets of type 1 (SOS1) ↩

Default: `0`

sos2reform *(integer)*: automatic logarithmic reformulation of special ordered sets of type 2 (SOS2) ↩

Default: `0`

startalg *(integer)*: MIP starting algorithm ↩

Selects the algorithm to use for the initial relaxation of a MIP.

Default: `0`

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex
3	Network simplex
4	Barrier
5	Sifting
6	Concurrent

strongcandlim *(integer)*: size of the candidates list for strong branching ↩

Limit on the length of the candidate list for strong branching (VarSel = 3).

Default: `10`

strongitlim *(integer)*: limit on iterations per branch for strong branching ↩

Limit on the number of iterations per branch in strong branching (VarSel = 3). The default value of 0 causes the limit to be chosen automatically which is normally satisfactory. Try reducing this value if the time per node seems excessive. Try increasing this value if the time per node is reasonable but Cplex is making little progress.

Default: `0`

subalg *(integer)*: algorithm for subproblems ↩

Strategy for solving linear sub-problems at each node.

Default: `0`

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex
3	Network optimizer followed by dual simplex
4	Barrier with crossover
5	Sifting

submipnodelim *(integer)*: limit on number of nodes in an RINS subMIP ↩

Controls the number of nodes explored in an RINS subMIP. See option RINSHeur.

Default: `500`

submipscale *(integer)*: scale the problem matrix when CPLEX solves a subMIP during MIP optimization ↩

Default: `0`

value	meaning
-1	No scaling
0	Standard scaling
1	Modified, more aggressive scaling method

submipstartalg *(integer)*: starting algorithm for a subMIP of a MIP ↩

Default: `0`

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex

value	meaning
3	Network simplex
4	Barrier
5	Sifting

submipsubalg *(integer)*: algorithm for subproblems of a subMIP of a MIP ↩

Default: 0

value	meaning
0	Automatic
1	Primal simplex
2	Dual simplex
3	Network optimizer followed by dual simplex
4	Barrier with crossover
5	Sifting

symmetry *(integer)*: symmetry breaking cuts ↩

Determines whether symmetry breaking cuts may be added, during the preprocessing phase, to a MIP model.

Default: -1

value	meaning
-1	Automatic
0	Turn off symmetry breaking
1	Moderate level of symmetry breaking
2	Aggressive level of symmetry breaking
3	Very aggressive level of symmetry breaking
4	Highly aggressive level of symmetry breaking
5	Extremely aggressive level of symmetry breaking

threads *(integer)*: global default thread count ↩

Synonym: gthreads

Default number of parallel threads allowed for any solution method. Negative values are interpreted as the number of cores to leave free so setting threads to -1 leaves one core free for other tasks. Cplex does not understand negative values for the `threads` parameter. GAMS/Cplex will translate this into a positive number by applying the following formula: max(1,number of cores-|threads|). Setting threads to 0 lets Cplex use at most 32 threads or the number of cores of the machine, whichever is smaller.

Default: `GAMS Threads`

tilim *(real)*: overrides the GAMS ResLim option ↩

Synonym: reslim

The time limit setting determines the amount of time in seconds that Cplex will continue to solve a problem. This Cplex option overrides the GAMS ResLim option. Any non-negative value is valid.

Default: `GAMS ResLim`

trelim *(real)*: maximum space in memory for tree ↩

Sets an absolute upper limit on the size (in megabytes) of the branch and cut tree. If this limit is exceeded, Cplex terminates optimization.

Default: `1.0e+75`

tuning *(string)*: invokes parameter tuning tool ↩

Invokes the Cplex parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Cplex option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in LP, MPS or SAV format. Cplex will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after the GAMS/Cplex option file name without considering the problem provided by GAMS. Due to technical reasons a single option input line is limited by 256 characters. If the list of model files exceeds this length you can provide a second, third, ... line starting again with keyword `tuning` and a list of model instance files.

The result of such a tuning run is the updated GAMS/Cplex option file with a tuned set of parameters. The solver and model status returned to GAMS will be `NORMAL COMPLETION` and `NO SOLUTION`. More details on Cplex tuning can be found on `IBM's web page`. Tuning is incompatible with the BCH facility and other advanced features of GAMS/Cplex.

tuningdettilim *(real)*: tuning deterministic time limit per model or suite ↩

Default: `1.0e+75`

tuningdisplay *(integer)*: level of information reported by the tuning tool ↩

Specifies the level of information reported by the tuning tool as it works.

Default: `1`

value	meaning
0	Turn off display
1	Display standard minimal reporting
2	Display standard report plus parameter settings being tried
3	Display exhaustive report and log

tuningmeasure *(integer)*: measure for evaluating progress for a suite of models ↩

Controls the measure for evaluating progress when a suite of models is being tuned. Choices are mean average and minmax of time to compare different parameter sets over a suite of models

Default: `1`

value	meaning
1	mean average
2	minmax

tuningrepeat *(integer)*: number of times tuning is to be repeated on perturbed versions ↩

Specifies the number of times tuning is to be repeated on perturbed versions of a given problem. The problem is perturbed automatically by Cplex permuting its rows and columns. This repetition is helpful when only one problem is being tuned, as repeated perturbation and re-tuning may lead to more robust tuning results. This parameter applies to only one problem in a tuning session.

Default: `1`

tuningtilim *(real)*: tuning time limit per model or suite ↩

Sets a time limit per model and per test set (that is, suite of models).

As an example, suppose that you want to spend an overall amount of time tuning the parameter settings for a given model, say, 2000 seconds. Also suppose that you want Cplex to make multiple attempts within that overall time limit to tune the parameter settings for your model. Suppose further that you want to set a time limit on each of those attempts, say, 200 seconds per attempt. In this case you need to specify an overall time limit of 2000 using GAMS option `reslim` or Cplex option TiLim and `tuningtilim` to 200.

Default: `0.2*GAMS ResLim`

upperobjstop *(real)*: in a maximization MILP or MIQP, the solver will abort the optimization process as soon as it finds a solution of value greater than or equal to the specified value ↩

Default: `1e75`

usercallparmfile *(string)*: Command-line parameter include file used in GAMS command-line calls triggered by BCH ↩

.usercut *(boolean)*: User cut activation ↩

Determines whether a linear constraint is treated as a user cut. At the beginning of the MIP solution process, any constraint whose usercut attribute is set to 1 (the default value is 0) is removed from the model and placed in the user cut pool. User cuts may be used by CPLEX at any time to improve the solution process. There is no guarantee that they are actually used.

The user cut pool is only active if option usercutpool is enabled and are specified through the option `.usercut`. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: `0`

usercutcall *(string)*: the GAMS command line to call the cut generator ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

usercutfirst *(integer)*: calls the cut generator for the first n nodes ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `10`

usercutfreq *(integer)*: determines the frequency of the cut generator model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `10`

usercutinterval *(integer)*: determines the interval when to apply the multiplier for the frequency of the cut generator model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `100`

usercutmult *(integer)*: determines the multiplier for the frequency of the cut generator model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `2`

usercutnewint *(boolean)*: calls the cut generator if the solver found a new integer feasible solution ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `1`

usercutpool *(boolean)*: Indicator to use user cuts ↩

Default: `0`

usergdxin *(string)*: the name of the GDX file read back into Cplex ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `bchin.gdx`

usergdxname *(string)*: the name of the GDX file exported from the solver with the solution at the node ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `bchout.gdx`

usergdxnameinc *(string)*: the name of the GDX file exported from the solver with the incumbent solution ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `bchout_i.gdx`

usergdxprefix *(string)*: prefixes usergdxin, usergdxname, and usergdxnameinc ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

usergdxsol *(string)*: the name of the GDX file exported by Cplex to store the solution of extra columns ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `bchsol.gdx`

userheurcall *(string)*: the GAMS command line to call the heuristic ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

userheurfirst *(integer)*: calls the heuristic for the first n nodes ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 10

userheurfreq *(integer)*: determines the frequency of the heuristic model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 10

userheurinterval *(integer)*: determines the interval when to apply the multiplier for the frequency of the heuristic model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 100

userheurmult *(integer)*: determines the multiplier for the frequency of the heuristic model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 2

userheurnewint *(boolean)*: calls the heuristic if the solver found a new integer feasible solution ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 1

userheurobjfirst *(integer)*: Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 0

userincbcall *(string)*: the GAMS command line to call the incumbent checking program ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

userincbicall *(string)*: the GAMS command line to call the incumbent reporting program ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

userjobid *(string)*: postfixes lf, o on call adds –userjobid to the call. Postfixes gdxname, gdxnameinc and gdxin ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

userkeep *(boolean)*: calls gamskeep instead of gams ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: 0

userlazyconcall *(string)*: the GAMS command line to call the lazy constraint generator ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Note: There is no guarantee that CPLEX will use all of the added violated lazy constraints provided due to technical and/or efficiency reasons. It may thus happen that a later candidate solution violates previously provided lazy constraints. In this case consider passing the constraint again.

varsel *(integer)*: variable selection strategy at each node ↩

This option is used to set the rule for selecting the branching variable at the node which has been selected for branching. The default value of 0 allows Cplex to select the best rule based on the problem and its progress.

Default: 0

value	meaning
-1	Branch on variable with minimum infeasibility This rule may lead more quickly to a first integer feasible solution, but will usually be slower overall to reach the optimal integer solution.
0	Branch variable automatically selected
1	Branch on variable with maximum infeasibility This rule forces larger changes earlier in the tree, which tends to produce faster overall times to reach the optimal integer solution.
2	Branch based on pseudo costs Generally, the pseudo-cost setting is more effective when the problem contains complex trade-offs and the dual values have an economic interpretation.
3	Strong Branching This setting causes variable selection based on partially solving a number of subproblems with tentative branches to see which branch is most promising. This is often effective on large, difficult problems.
4	Branch based on pseudo reduced costs

warninglimit *(integer)*: determines how many times warnings of a specific type (datacheck=2) will be displayed ↩

By default, when modeling assistance is turned on via the data consistency checking parameter, CPLEX will display 10 warnings for a given modeling issue and then omit the rest. This parameter controls this limit and allows the user to display all of the warnings if desired. In order to see all warnings change the value to its negative.

Default: `10`

workdir *(string)*: directory for working files ↩

The name of an existing directory into which Cplex may store temporary working files. Used for MIP node files and by out-of-core Barrier.

Default: `current or project directory`

workeralgorithm *(integer)*: set method for optimizing benders subproblems ↩

Default: `0`

value	meaning
0	Automatic
1	Primal Simplex
2	Dual Simplex
3	Network Simplex
4	Barrier
5	Sifting

workmem *(real)*: memory available for working storage ↩

Upper limit on the amount of memory, in megabytes, that Cplex is permitted to use for working files. See parameter WorkDir.

Default: `2048.0`

writeannotation *(string)*: produce a Cplex annotation file ↩

writebas *(string)*: produce a Cplex basis file ↩

Write a basis file.

writeflt *(string)*: produce a Cplex solution pool filter file ↩

Write the diversity filter to a Cplex FLT file.

writelp *(string)*: produce a Cplex LP file ↩

Write a file in Cplex LP format.

writemps *(string)*: produce a Cplex MPS file ↩

Write an MPS problem file.

writemst *(string)*: produce a Cplex mst file ↩

Write a Cplex MST (containing the MIP start) file.

writeord *(string)*: produce a Cplex ord file ↩

Write a Cplex ORD (containing priority and branch direction information) file.

writeparam *(string)*: produce a Cplex parameter file with all active options ↩

Write a Cplex parameter (containing all modified Cplex options) file.

writepre *(string)*: produce a Cplex LP/MPS/SAV file of the presolved problem ↩

Synonym: writepremps

Write a Cplex LP, MPS, or SAV file of the presolved problem. The file extension determines the problem format. For example, `WritePre presolved.lp` creates a file `presolved.lp` in Cplex LP format.

writeprob *(string)*: produce a Cplex problem file and inferrs the type from the extension ↩

Write a problem file in a format inferred from the extension. Possible formats are

- `SAV`: Binary matrix and basis file
- `MPS`: MPS format
- `LP`: CPLEX LP format with names modified to conform to LP format
- `REW`: MPS format, with all names changed to generic names
- `RLP`: LP format, with all names changed to generic names
- `ALP`: LP format, with generic name of each variable, type of each variable, bound of each variable If the file name ends with `.bz2` or `.gz`, a compressed file is written.

writesav *(string)*: produce a Cplex binary problem file ↩

Write a binary problem file.

zerohalfcuts *(integer)*: zero-half cuts ↩

Decides whether or not to generate zero-half cuts for the problem. The value 0, the default, specifies that the attempt to generate zero-half cuts should continue only if it seems to be helping. If the dual bound of your model does not make sufficient progress, consider setting this parameter to 2 to generate zero-half cuts more aggressively.

Default: `0`

value	meaning
-1	Off
0	Automatic
1	Generate zero-half cuts moderately
2	Generate zero-half cuts aggressively

5.11.9 Setting up a GAMS/Cplex-Link license

The GAMS/Cplex-Link license requires that you have a valid license agreement with IBM for use of the current version of the Cplex library.

To add the Cplex-Link to your GAMS license, please write to sales@gams.com to confirm that you have such a license agreement and that your use of the GAMS solver will comply with the terms of that license agreement. Please reference your GAMS License ID, DCxxxx.

5.12 Deterministic Equivalent (DE)

Author

Martha Loewe

5.12.1 Introduction

DE is a solver for stochastic programs modeled with GAMS Extended Mathematical Programming for Stochastic Programming (EMP SP). DE can solve multi-stage LP, MIP, QCP, NLP and MINLP stochastic programming models. For details about EMP SP and the syntax to modify an existing GAMS model to be an stochastic programming model in GAMS EMP SP see Stochastic Programming. A list of DE solver options is given at the end of this document.

Stochastic programs are mathematical programs that include data that is not known with certainty, but is approximated by probability distributions. The simplest form of a stochastic program is the two-stage stochastic linear program with recourse. In mathematical terms it is defined as follows.

Let $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^m$ be two variables and let the set of all realizations of the unknown data be given by Ω, $\Omega = \{\omega_1, \ldots, \omega_S\} \subseteq \mathbb{R}^r$, where r is the number of the random variables representing the uncertain parameters. Then the stochastic program is given by

$$\begin{array}{lllllllll} & \text{Min}_x & z = & c^T x & + & \mathbb{E}[Q(x,\omega)] & & & \\ & \text{s.t.} & & Ax & & & = & b, & x \geq 0, \\ \text{where} \quad Q(x,\omega) = & \text{Min}_y & & & & q_\omega^T y(\omega) & & & \\ & \text{s.t.} & & T_\omega x & + & W_\omega y(\omega) & = & h_\omega, & y(\omega) \geq 0, \quad \forall \omega \in \Omega. \end{array} \tag{5.1}$$

The first two lines define the first-stage problem and the last two lines define the second-stage problem. In the first stage, x is the decison variable, c^T represents the cost coefficients of the objective function and $\mathbb{E}[Q(x,\omega)]$ denotes the expected value of the optimal solution of the second stage problem. In addition, A denotes the coefficients and b the right-hand side of the first stage constraints. In the second stage, y is the decision variable, T represents the transition matrix, W the recourse matrix (cost of recourse) and h

the right-hand side of the second stage constraints. Note that all parameters and the decision variable of the second stage are dependent on ω. The objective variable z is also a random variable, since it is a function of ω. As a random variable cannot be optimized, DE automatically optimizes the *expected value* of the objective variable z. DE is also able to optimize other risk measures, as will be discussed later.

In the first stage, a decision has to be made before the realization of the uncertain data is clear. The optimal solution of the first stage is fixed and only then the values that the uncertain parameters take will become known. Given the fixed solution of the first stage and the new data, in the second stage recourse action can be taken and the optimal solution determined. Each possible realization of the uncertain data is represented by an $\omega_s \in \Omega$ and is called a *scenario*. The objective is to find a feasible solution x that minimizes the total cost, namely the sum of the first-stage costs and the expected second-stage costs.

One of the most common methods to solve a two-stage stochastic LP is to build and solve the *deterministic equivalent*. Assume that the uncertain parameters follow a (finite) discrete distribution and that each scenario ω_s occurs with probability $P(\omega_s) = p_s$ for all $s = 1, \ldots, S$ and $\sum_s p_s = 1$. So $\mathbb{E}[Q(x, \omega)] = \sum_s p_s q^T y_s$, where y_s denotes the optimal second-stage decision for the scenario ω_s. Then the deterministic equivalent can be expressed as follows:

$$
\begin{array}{llllllllll}
c^T x & + & p_1 q^T y_1 & + & p_2 q^T y_2 & + & \ldots & + & p_S q^T y_S & \\
\text{s.t.} & & & & & & & & & \\
Ax & & & & & & & & = & b \\
T_1 x & + & W_1 y_1 & & & & & & = & h_1 \\
T_2 x & & & + & W_2 y_2 & & & & = & h_2 \\
\vdots & & & + & & & \ddots & & & \vdots \\
T_S x & & & & & & & + \; W_S y_S & = & h_S \\
x \in \mathbb{R}^n & & y_1 \in \mathbb{R}^m & & y_2 \in \mathbb{R}^m & & & & y_S \in \mathbb{R}^m &
\end{array}
\tag{5.2}
$$

Note that for stochastic linear programs the deterministic equivalent is just a very large linear program.

The solver DE automatically generates the deterministic equivalent of a stochastic program and then hands it over for solution to a solver, termed the *subsolver*. The subsolver is the default solver for the type of model to be solved. The user may choose another subsolver with the option subsolver.

If modelers want to use DE, they may either add the option

```
option emp = de;
```

before the `solve` statement in the model or they may choose the solver DE on the command line, e.g.

```
gams mymodel.gms emp=de
```

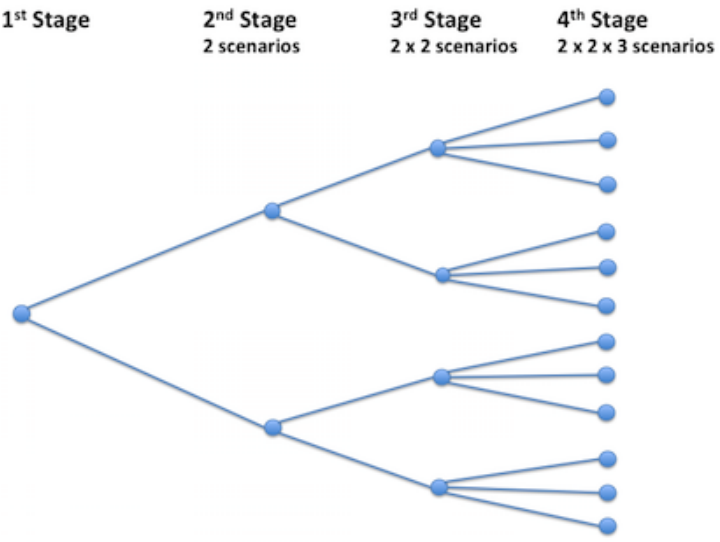
5.12.2 Random Variables

Random variables and their distributions and stages determine the number of scenarios in the model and thus the number of equations that are generated for the deterministic equivalent. If there are two or more random variables in one stage they may be jointly distributed or not.

Assume that there is a two-stage stochastic model $\mathcal{M}_2$ that has two random variables a and b in the second stage. The random variables both follow a discrete distribution, the uncertain data for a is given by Ω_a and the uncertain data for b is given by Ω_b. If the random variables are *not* jointly distributed, then $|\Omega_a| \times |\Omega_b|$ scenarios are generated. If the random variables are jointly distributed, then we need to have $|\Omega_a| = |\Omega_b|$ and the number of scenarios is $|\Omega_a|$.

In addition to two-stage stochastic models, DE can also solve multi-stage stochastic models. Let $\mathcal{M}_4$ be a stochastic model with four stages. In each of the stages after the first stage new data is revealed that was unknown in the previous stages. Assume that in each stage there is just one random variable. Let Ω_{I} denote the uncertain data in the second stage and Ω_{II} and Ω_{III} represent the uncertain data of the third and fourth stage respectively. Let $|\Omega_{\mathrm{I}}| = n_1$, $|\Omega_{\mathrm{II}}| = n_2$ and $|\Omega_{\mathrm{III}}| = n_3$. DE builds the deterministic equivalent of this model by recursively generating a scenario tree. In the second stage the number of scenarios is n_1, in the third stage there are $n_1 \times n_2$ scenarios and in the fourth stage there are $n_1 \times n_2 \times n_3$ scenarios. Note that the number of scenarios in the model equals the number of scenarios in the last stage. However, the number of equations generated for the deterministic equivalent is the sum of the scenarios in each stage, so in this example it is $n_1 + n_1 \times n_2 + n_1 \times n_2 \times n_3 = n_1 \times (1 + n_2 \times (1 + n_3))$.

The following figure illustrates the scenario tree of a stochastic model with four stages.



Note that the structure of the scenario tree is limited: each node in a stage has exactly the same number of arcs.

If there is more than one random variable per stage, then the number of scenarios increases accordingly. Note that there is no absolute limit to the number of stages and random variables, but the model can become very large very quickly as the number of stages and random variables increases. Note further that the random variables are stagewise independent. The realization of the random variables in one stage does not influence the realization of the random variables in a later stage.

5.12.3 Sampling Procedures

DE can solve only stochastic programs with random variables that follow discrete probability distributions. If the random data follows a continuous probability distribution the set Ω has an infinite number of elements. Sampling is a process that generates a finite approximation Ω_N to Ω. The size of Ω_N is N and each element of Ω_N has the same probability $p = \frac{1}{N}$. Thus continuous probability distributions are approximated by discrete distributions. EMP SP provides the keyword `sample` to facilitate sampling. An example follows.

```
randvar d normal 45 10
sample d 100
```

These two lines are a portion of the `emp.info` file. In the first line `d` is defined as a random variable that follows a Normal distribution with mean 45 and standard deviation 10. In the second line it is specified that the continuous distribution is approximated by a sample of size 100. Note that the user determines the sample size N. Note further that the use of the keyword `sample` is mandatory for random variables with continuous distributions. Otherwise the following error message will appear:

```
*** Only random variables with sampled continuous distributions supported.
```

Observe that behind the scenes the sampling is performed by the LINDO system. So the modeler needs a LINDO licence for sample sizes larger than 10 (smaller sample sizes are included in the demo version). The LINDO system offers three variance reduction algorithms: the Antithetic algorithm, the Latin Square algorithm and the Monte Carlo algorithm. They may be enabled using either the option svr_ls_antithetic or svr_ls_latinsquare or svr_ls_montecarlo respectively.

Alternatively, the modeler may chose to generate a sample of the distribution first and then enter the sample as a discrete distribution in the `emp.info` file. There are two stochastic libraries that can be used for this procedure: the GAMS Stochastic Library and the LINDO Sampling Library. An example and details about the procedure using the LINDO library are given in section Sampling in the EMP manual.

5.12.4 The Expected Value Problem

Consider the special case where the sample size is 1 and the "sampled" value equals the expected value of the random variable. This entails that the random variable is replaced by its expected value. The resulting model is deterministic and is called the Expecetd Value Program. DE facilitates solving the Expected Value Problem through the option solveEVProb. If this option is specified in the option file (see example below) the Expected Value Problem is solved after the original stochastic model and the solution is reported. In addition, DE offers the option to choose a separate subsolver for the Expecetd Value Problem, the option evsubsolver.

In the following example the stochastic model is solved with the subsolver Conopt and the Expected Value Problem is solved with the subsolver Minos.

```
$onecho > de.opt
subsolver conopt
solveEVProb
evsubsolver minos
$offecho
mymodel.optfile=1;
solve mymodel min obj using emp scenario dict;
```

Note that the option solveEVProb is not defined for models with chance constraints and models featuring VaR and CVaR. These model types are discussed in the next section.

5.12.5 What types of models can DE handle?

As mentioned earlier, DE can solve two-stage and multi-stage stochastic linear, quadratic and nonlinear models and the mixed-integer versions of all these models. There are two further classes of stochastic models that DE is equipped to solve: stochastic programs with chance constraints and stochastic programs with other risk measures such as Variance at Risk (VaR) and Conditional Variance at Risk (CVaR). CVaR is also called Expected Shortfall.

In stochastic programs with chance constraints the goal is to make an optimal decision prior to the realization of random data while allowing the constraints to be violated with a certain probability. Mathematically, a stochastic linear program with chance constraints can be expressed as follows:

$$\begin{array}{ll} \text{Min}_x & c^T x \\ \text{s.t.} & P(Ax \leq b) \geq p \\ & x \geq 0, \end{array} \tag{5.3}$$

where $x \in \mathbb{R}^n$ is the decision variable and c^T denotes the coefficients of the objective function, $A \in \mathbb{R}^{m \times n}$ is a random matrix and represents the coefficients and $b \in \mathbb{R}^m$ is a random vector and denotes the right-hand side of the constraints. The distinctive feature of stochastic programs with chance constraints is that the constraints (or some of them) may be violated with probability $\epsilon = 1 - p$, where $0 < p \leq 1$.

DE offers three reformulation options to solve stochastic programs with chance constraints: a reformulation using a mixed-integer program with big M notation, a convex hull reformulation and the use of indicator variables and indicator constraints. Note that the default is the MIP reformulation (the default value of M is set to 10000 and can be customized). With the option ccreform the reformulation method can be determined by the user.

For details on single and joint chance constraints and examples how to use the option `ccreform`, see the section Chance Constraints with EMP in the EMP manual. Note that random variables in programs with chance constraints may follow discrete or continuous probability distributions. Note further that there are no stages in stochastic programs with chance constraints.

As mentioned earlier, DE automatically optimizes the expected value of the objective function variable. DE also supports other risk measures, such as VaR and CVaR. In mathematical terms, DE is able to solve the following:

$$\text{Min}_x \quad \mathcal{R}(z), \tag{5.4}$$

where z represents the objective function variable and $\mathcal{R}$ denotes a risk measure such as the Expected Value, VaR and CVaR or a combination of risk measures (e.g. the weighted sum of Expected Value and CVaR). In addition, the modeler can also choose to trade off risk measures. For more details about risk measures see section Risk Measures with EMP in the EMP manual.

5.12.6 Reformulation Techniques

In this section some details are given about the reformulations that DE is performing behind the scenes.

5.12.6.1 Simple two-stage stochastic model

As mentioned previously, DE converts a stochastic model into its deterministic equivalent. Using the model `nbsimple.gms` from the GAMS EMP model library as an example, we show how exactly the deterministic equivalent is built. Note that this model is also discussed in detail in the section A Simple Example: The News Vendor Problem of the EMP manual.

The model is given by the following equations:

$$\text{Max}_x \quad z(x) = -cx + \mathbb{E}[Q(x, \omega)], \quad x \geq 0, \tag{5.5}$$

where

$$\begin{array}{rlrl} Q(x,\omega) = \text{Max}_{s,i,l} & & vs(\omega) - hi(\omega) - pl(\omega) & \\ \text{s.t.} & x - s(\omega) - i(\omega) & & = 0 \\ & s(\omega) & + l(\omega) & = d(\omega) \\ & s(\omega),\, i(\omega),\, l(\omega) & \geq 0 & \forall \omega \in \Omega. \end{array}$$

Here c, v, h and p are some given parameters, the first-stage decision variable is x, the second-stage decision variables are s, i and l and the uncertain data is represented by the random variable $d(\omega)$. We assume that the annotations specify that d follows a Normal distribution and the sample size to approximate this continuous distribution is 4 (where all scenarios are equally likely, so $p(s) = 0.25$).

First, DE draws 4 values from the Normal distribution specified in the annotations. These 4 values are the basis for the 4 scenarios that are generated. For each scenario, a separate equation for the two second-stage constraints is built (that are 8 equations). In the process, for each of the three decison variables of the second stage 4 variables are generated since the second-stage decison variables may take different values for each scenario. Next, an equation for the profit for each scenario is generated (4 equations) and finally, the equation to compute the average profit is built. So there are a total of 13 equations. They are given here:

$$\begin{array}{lcl} x - s_1 - i_1 & = & 0 \\ x - s_2 - i_2 & = & 0 \\ x - s_3 - i_3 & = & 0 \\ x - s_4 - i_4 & = & 0 \\ s_1 + l_1 & = & d_1 \\ s_2 + l_2 & = & d_2 \\ s_3 + l_3 & = & d_3 \\ s_4 + l_4 & = & d_4 \\ z_1 & = & -cx + vs_1 - hi_1 - pl_1 \\ z_2 & = & -cx + vs_2 - hi_2 - pl_2 \\ z_3 & = & -cx + vs_3 - hi_3 - pl_3 \\ z_4 & = & -cx + vs_4 - hi_4 - pl_4 \\ z & = & 0.25z_1 + 0.25z_2 + 0.25z_3 + 0.25z_4 \end{array} \tag{5.6}$$

Note that there is only one realization of the first-stage decision variable x while for the second-stage decision variables there as many realizations as there are scenarios.

5.12.6.2 A multi-stage stochastic model

DE recursively builds a scenario tree to generate the deterministic equivalent of multi-stage models. We take the model `inventory` from the EMP manual to demonstrate how this is done. The 4-stage model is given by the following equations:

$$\begin{array}{ll} \text{Max}_{y_1,i_1} & z = -\alpha y_1 - \gamma i_1 + \mathbb{E}[Q((y_1,i_1),\omega_{\text{I}}) + \mathbb{E}[Q((y_1,i_1),\omega_{\text{II}}) + \mathbb{E}[Q((y_1,i_1),\omega_{\text{III}})]]] \\ \text{s.t.} & i_1 = y_1 \\ & y_1, i_1 \geq 0, \end{array}$$

where

$$\begin{array}{rlll} Q((y_1,i_1),\omega_\zeta) = & \text{Max}_{s_t,y_t,i_t} & \beta s_t(\omega_\zeta) - \alpha y_t(\omega_\zeta) - \delta i_t(\omega_\zeta) & \\ & \text{s.t.} & i_{t-1} + y_t = s_t + i_t & \\ & & s_t \leq i_{t-1} & \\ & & s_t \leq d_t(\omega_\zeta) & \\ & & i_t \leq \kappa & \\ & & s_t, y_t, i_t \geq 0 \quad \text{for} \quad t = 2,3,4. & \end{array}$$

Here y_1 and i_1 are the first-stage decision variables and y_t, i_t and s_t are the decison variables of the later stages; α, β, γ, δ and κ denote fixed parameters. The uncertain data is represented by the random variable $d_t(\omega_\zeta)$, where $\omega_\zeta \in \Omega_\zeta$ and $\zeta = \mathrm{I}, \mathrm{II}, \mathrm{III}$. Note that s_t, y_t and i_t depend on the realization of d_t. Now assume that d_t follows a continuous distribution and the sample size to approximate this distribution is just 3 for stages 2 to 4. Note that in multi-stage models the sample size may vary for each stage. Note further that for all practical purposes this sample size is too small, it was chosen here for reasons of clarity of exposition. As usual, the distribution(s) and sample size(s) are specified in the annotations. Observe that the last constraint ($i_t \leq \kappa$) is implemented as a bound on the variable i_t, not as an equation. So, in addition to the objective equations, there is one equation in the first stage and three equations in each of the later stages.

This model is reformulated in the following way. First, an equation for the first stage constraint is generated. Secondly, three realizations of d_2 are chosen through a sampling procedure, so there are three scenarios at stage 2. For each of these three realizations of the random variable the equations for the second stage are generated (9 equations). Thirdly, three realizations of d_3 are sampled for stage 3. Each of these three realizations may follow one of the three scenarios of the second stage, so there are nine scenarios in stage 3 and hence $9 \times 3 = 27$ equations are generated for stage 3. The procedure is repeated in stage 4 where there are 27 scenarios and $27 \times 3 = 81$ equations. Next, the value of the objective variable for each scenario is computed by generating 27 equations for z_s and finally, an equation to compute z is built, where $z = \sum_s p(s) z_s$.

5.12.6.3 Chance Constraints

Models with chance constraints are reformulated as mixed integer problems by default. This reformulation is discussed in detail in the section Chance Constraints with EMP in the EMP manual. Reformulations using a convex hull or indicator variables and indicator constraints are also possible. The reformulation method may be determined by the user with the option ccreform.

5.12.6.4 Computing VaR

Models involving VaR are reformulated using mixed integer programs with big M notation. By the example model `portfolio.gms` we demonstrate how this is done. This model is discussed in the section Value at Risk (VaR) in the EMP manual. The model is given by the following equations:

$$\begin{array}{lll} \text{Max} & \underline{VaR}_\theta[R] & \\ \text{s.t} & R = \sum_j w_j v_j(\omega) & \forall \omega \in \Omega \\ & \sum_j w_j = 1 & \\ & w_j \geq 0, & \end{array} \tag{5.7}$$

where $\underline{VaR}_\theta$ is the VaR at the lower θth percentile, the variable R is the return (and is a function of the random variable $v_j(\omega)$), w_j the weight associated with each asset j and $v_j(\omega)$ is the (random variable) return of each asset j. The weights can also be interpreted as proportions of the amount to be invested, their sum must be 1. Note that w_j is the decision variable in this problem.

Let $y(s)$ be a binary variable that acts as an indicator variable. For each scenario, it equals 1 if the return $R \geq \underline{VaR}_\theta[R]$ and it is 0 otherwise. Then the model is reformulated to the following MIP:

$$\begin{array}{lll} \text{Max} & \underline{VaR}_\theta[R] & \\ \text{s.t} & r(s) = \sum_j w_j v_j(s) & \forall s = 1, \ldots, S \\ & r(s) \geq \underline{VaR}_\theta[R] - M(1 - y(s)) & \forall s = 1, \ldots, S \\ & cc = 1 - \sum_s p(s) y(s) & \\ & \sum_j w_j = 1 & \\ & 0 \leq cc \leq \theta & \\ & w_j \geq 0. & \end{array} \tag{5.8}$$

Here $r(s)$ denotes the return per scenario and cc is a variable that represents the probability of violations (a violation occurs if in a scenario the return is smaller than $\underline{VaR}_\theta[R]$). As usual, $p(s)$ is the probability that a scenario occurs and M is the big M. The first three constraints together with the bounds on cc ensure that the probability of violations is smaller than or equal to θ.

Note that the default value for big M for models with VaR is 1000 (and it is different from the default value for big M for chance constraints). It may be customized with the option varbigm.

5.12.6.5 Computing CVaR

Models with CVaR are reformulated by converting the annotations into new variables and equations. Using the model `portfolio.gms` as an example we demonstrate how this is done. This model features in the GAMS EMP library and is also discussed in the section Conditional Value at Risk (CVaR) in the EMP manual. The model is given by:

$$\begin{array}{lll} \text{Max} & \underline{CVaR}_\theta[R] & \\ \text{s.t} & R = \sum_j w_j v_j(\omega) & \forall \omega \in \Omega \\ & \sum_j w_j = 1 & \\ & w_j \geq 0, & \end{array} \tag{5.9}$$

where $\underline{CVaR}_\theta$ is the CVaR at the left tail of the distribution at the confidence level θ and all other variables are as defined above. Note that again, w_j is the decision variable in this problem.

DE reformulates this model by introducing three new variables and three associated equations. Let $r(s)$ denote the return per scenario, t the target return to be achieved with probability approximately $1-\theta$ and $d(s)$, $d(s) \geq 0$, the lower deviation from t per scenario. The equations follow.

$$\begin{aligned} r(s) &= \sum_j w_j v_j(s) \\ d(s) &\geq t - r(s) \\ \underline{CVaR}_\theta[R] &= t - \frac{1}{\theta} \sum p(s) d(s) \end{aligned} \tag{5.10}$$

Thus the model is reformulated to an LP and may be solved by an appropriate solver. It is easy to see that t equals $\underline{VaR}_\theta[R]$.

5.12.7 Logfile

The logfile gives much information about the solver progress. The following is the DE log output from running the **nbdiscjoint** model from the GAMS EMP Model Library:

```
--- Starting compilation
--- nbdisjoint.gms(74) 3 Mb
--- Starting execution: elapsed 0:00:00.024
--- nbdisjoint.gms(72) 4 Mb
---
--- collecting and writing gdx file
---
--- Generating EMP model nb
--- nbdisjoint.gms(72) 6 Mb
---   5 rows  9 columns  19 non-zeroes
---   6 nl-code  2 nl-non-zeroes
```

```
--- nbdisjoint.gms(72) 4 Mb
--- Executing DE: elapsed 0:00:00.083
--- Input model type identified and solved as LP
DE               24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X
--- DE has 26 rows 50 columns 116 non-zeroes

IBM ILOG CPLEX   24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X
Cplex 12.6.2.0

Reading data...
Starting Cplex...
Unable to load names.
Tried aggregator 1 time.
LP Presolve eliminated 14 rows and 31 columns.
Reduced LP has 12 rows, 19 columns, and 36 nonzeros.
Presolve time = 0.01 sec. (0.02 ticks)

Iteration log . . .
Iteration:     1   Dual infeasibility =            52.800000
Iteration:    10   Dual objective     =          2660.700000
LP status(1): optimal
Cplex Time: 0.06sec (det. 0.05 ticks)

Optimal solution found.
Objective :        1173.900000

--- Restarting execution
--- nbdisjoint.gms(72) 2 Mb
--- Reading solution for model nb
--- nbdisjoint.gms(72) 3 Mb
---
--- scattering and reading gdx file
---
--- randvar  Id = d maps to s_d
--- nbdisjoint.gms(72) 3 Mb
--- randvar  Id = r maps to s_r
--- level    Id = s maps to s_s
--- level    Id = x maps to s_x
--- nbdisjoint.gms(72) 4 Mb
--- Scatter finished in 2 ms
--- nbdisjoint.gms(74) 4 Mb
*** Status: Normal completion
--- Job nbdisjoint.gms Stop 12/06/15 18:02:47 elapsed 0:00:00.382
```

First, the model in scalar form is generated. It has 5 equations, so the model statistics report 5 rows. Further, it has 6 positive variables, one free variable and 2 random variables, so there are 9 columns. A matrix form representation of the model with the variables as columns and the equations as rows has 19 non-zero entries. Next, DE generates the deterministic equivalent. The statistics for the deterministic equivalent indicate that there are 26 rows, 50 columns and 116 non-zero entries. The stochastic model has 6 scenarios, so for each second-stage equation there are 6 equations in the deterministic equivalent (i.e. a total of 24 equations). In addition, there is one first-stage equation and one equation to compute the expected value of the objective variable, which brings the sum total to 26 equations or rows. Similarly, for each second-stage variable in the stochastic model there are 6 variables in the determininistic equivalent (i.e a total of 48 variables). In addition, there is the first-stage decision variable and a variable for the expected value of the objective adding up to a total of 50 variables or columns.

Note that the model type is identified as an LP and thus the default LP solver is invoked. In this case the subsolver is Cplex and most of the remainder of the logfile is output from the subsolver. At the end the mapping specified in the dictionary is performed.

The logfile of solving stochastic models with VaR or CVaR is similar. It reports that the model type is being determined, the deterministic equivalent built and then handed over to the appropriate subsolver to be solved. However, when solving stochastic programs with chance constraints there is much more happening behind the scenes. DE automatically hands over the model to the subsolver JAMS. By default, JAMS reformulates the model and generates a scalar version of it. The scalar version of the reformulated model is then handed back to the GAMS system to be solved by an appropriate subsolver. The logfile reports the progress of the subsolver until the model is solved. Then the solution from the GAMS subsolver is handed to JAMS and a list of disjunsctions and their activity level is reported. (The equations in the scalar version of the reformulated model that may be left unsatisfied are called *disjunctions*. *Active* disjunctions refer to equations that are satisfied and *not active* disjunctions refer to equations that are not satisfied.)

Parts of the DE logfile from running the **simplechance** model from the GAMS EMP Model Library are given below. These parts of the logfile demonstrate the solution process for stochastic models with chance constraints that was just described.

```
...
--- Generating EMP model sc
--- simplechance.gms(79) 6 Mb
---   3 rows  5 columns  9 non-zeroes
---   9 nl-code  4 nl-non-zeroes
--- simplechance.gms(79) 4 Mb
--- Executing DE: elapsed 0:00:00.024
--- Input model type identified and solved as LP
DE               24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X

--- Reset Solvelink = 2

JAMS 1.0         24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X

JAMS - Solver for Extended Mathematical Programs (EMP)
------------------------------------------------------

--- Using Option File
Reading parameter(s) from "jams.159"
>>  EMPInfoFile .../Models/225a/jamsinfo.dat
>>  SubSolver CPLEX
Finished reading from "jams.159"

--- EMP Summary
    Logical Constraints = 0
    Disjunctions        = 7
    Adjusted Constraint = 0
    Flipped Constraints = 0
    Dual Variable Maps  = 0
    Dual Equation Maps  = 0
    VI Functions        = 0
    Equilibrium Agent   = 0
    Bilevel Followers   = 0

*** Warning 7 of 7 BigM disjunctions use DE's default for bigM (10000)

--- The model .../Models/225a/emp.dat will be solved by GAMS
---
--- Job emp.dat Start 12/06/15 18:43:18 24.5.1 r54187 DEX-DEG x86 64bit/MacOS X
GAMS 24.5.1   Copyright (C) 1987-2015 GAMS Development. All rights reserved
...
--- Generating MIP model m
```

```
--- emp.dat(68) 3 Mb
---   10 rows  19 columns  40 non-zeroes
---   7 discrete-columns
--- Executing CPLEX: elapsed 0:00:00.016

IBM ILOG CPLEX   24.5.1 r54187 Released Sep 23, 2015 DEG x86 64bit/MacOS X
Cplex 12.6.2.0

Reading data...
Starting Cplex...
....
MIP Solution:            4.750000    (11 iterations, 0 nodes)
Final Solve:             4.750000    (2 iterations)
...
--- Reading solution for model m
*** Status: Normal completion
--- Job emp.dat Stop 12/06/15 18:43:18 elapsed 0:00:00.299
--- Returning from GAMS step
---
--- Disjunction Summary
     Disjunction 1 is not active
     Disjunction 2 is active
     Disjunction 3 is active
     Disjunction 4 is active
     Disjunction 5 is not active
     Disjunction 6 is active
     Disjunction 7 is active
---
--- Restarting execution
--- simplechance.gms(79) 2 Mb
--- Reading solution for model sc
--- simplechance.gms(79) 3 Mb
---
--- scattering and reading gdx file
---
--- randvar  Id = om1 maps to s_om1
--- simplechance.gms(79) 3 Mb
--- randvar  Id = om2 maps to s_om2
--- level    Id = x1 maps to x1_l
--- marginal Id = x1 maps to x1_m
--- level    Id = x2 maps to x2_l
--- level    Id = e1 maps to e1_l
--- level    Id = e2 maps to e2_l
--- simplechance.gms(79) 4 Mb
--- Scatter finished in 2 ms
--- simplechance.gms(81) 4 Mb
*** Status: Normal completion
```

5.12.8 Summary of DE Options

5.12.8.1 General Options

Option	Description	Default
correlationtype	Sample correlation type	0

Option	Description	Default
empinfofile	Path and name of file containing additional EMP-SP information as randvar, jrandvar, stage etc.	
subsolver	Subsolver to run	
subsolveropt	Optfile value to pass to the subsolver	1
svr_ls_antithetic	Sample variance reduction map to Lindo Antithetic algorithm	
svr_ls_latinsquare	Sample variance reduction map to Lindo Latin Square algorithm	
svr_ls_montecarlo	Sample variance reduction map to Lindo Montecarlo algorithm	

5.12.8.2 Options for chance constraint models

Option	Description	Default
ccreform	Reformulation option passed to JAMS	bigM
jamsopt	JAMS option file	

5.12.8.3 Options for recourse models

Option	Description	Default
evsubsolver	Subsolver to run on expected value problem	
evsubsolveropt	Optfile value to pass to the subsolver for expected value problem	1
maxnodes	Tree size limit	100000
solveEVProb	Solve and report the expected value solution	0
varbigm	Big M for Value at Risk reformulation	1000.0

5.12.9 Detailed Descriptions of DE Options

ccreform *(string)*: Reformulation option passed to JAMS ↩

This option determines how to formulate the indicator contraints of the chance constraints in the deterministic equivalent. The model is passed to JAMS for the actual reformulation and solution with a subsolver. The possible reformulation options are `bigM [big eps threshold]`, `chull [big eps]`, and `indic`. The `indic` setting only works with solver that under stand the indicator syntax in a GAMS option file.

Default: `bigM`

correlationtype *(integer)*: Sample correlation type ↩

Default: `0`

value	meaning
0	Pearson
1	Kendall
2	Spearman

empinfofile *(string)*: Path and name of file containing additional EMP-SP information as randvar, jrandvar, stage etc. ↩

evsubsolver *(string)*: Subsolver to run on expected value problem ↩

evsubsolveropt *(integer)*: Optfile value to pass to the subsolver for expected value problem ↩

Range: {1, ..., 999}

Default: 1

jamsopt *(string)*: JAMS option file ↩

maxnodes *(integer)*: Tree size limit ↩

Range: {1000, ..., ∞}

Default: 100000

solveEVProb *(no value)*: Solve and report the expected value solution ↩

Default: 0

subsolver *(string)*: Subsolver to run ↩

subsolveropt *(integer)*: Optfile value to pass to the subsolver ↩

Range: {1, ..., 999}

Default: 1

svr_ls_antithetic *(string)*: Sample variance reduction map to Lindo Antithetic algorithm ↩

svr_ls_latinsquare *(string)*: Sample variance reduction map to Lindo Latin Square algorithm ↩

svr_ls_montecarlo *(string)*: Sample variance reduction map to Lindo Montecarlo algorithm ↩

varbigm *(real)*: Big M for Value at Risk reformulation ↩

Default: 1000.0

5.13 DECIS

Author

Gerd Infanger; Vienna University of Technology; Stanford University [1]

5.13.1 DECIS

5.13.1.1 Introduction

DECIS is a system for solving large-scale stochastic programs, i.e. programs that include parameters (coefficients and right-hand sides) that are not known with certainty, but are assumed to be known by their probability distribution. It employs Benders decomposition and advanced Monte Carlo sampling techniques. DECIS includes a variety of solution strategies, such as solving the universe problem, the expected value problem, Monte Carlo sampling within the Benders decomposition algorithm, and Monte Carlo pre-sampling. When using Monte Carlo sampling the user has the option of employing crude Monte Carlo without variance reduction techniques, or using as variance reduction techniques importance sampling or control variates, based on either an additive or a multiplicative approximation function. Pre-sampling is limited to using crude Monte Carlo only.

For solving linear and nonlinear programs (master and subproblems arising from the decomposition) DECIS interfaces with MINOS or CPLEX. MINOS, see Murtagh and Saunders (1983) [142] , is a state-of-the-art solver for large-scale linear and nonlinear programs, and CPLEX is one of the fastest linear programming solvers available.

For details about the DECIS system consult the `DECIS User's Guide` see Infanger (1997) [100] . It includes a comprehensive mathematical description of the methods used by DECIS. In this Guide we concentrate on how to use DECIS directly from GAMS, and especially on how to model stochastic programs using the GAMS/DECIS interface. First, however, in section What DECIS Can Do we give a brief description of what DECIS can do and what solution strategies it uses. This description has been adapted from the DECIS User's Guide. In section GAMS/DECIS we discuss in detail how to set up a stochastic problem using GAMS/DECIS and give a description of the parameter setting and outputs obtained. In Appendix A - GAMS/DECIS Illustrative Examples we show the GAMS/DECIS formulation of two illustrative examples (APL1P and APL1PC) discussed in the DECIS User's Guide. A list of DECIS error messages are represented in Appendix B - Error Messages.

5.13.1.2 What DECIS Can Do

DECIS solves two-stage stochastic linear programs with recourse:

$$\begin{array}{lrrcll} \min & z = cx + Ef^{\omega}y^{\omega} & & & & \\ s/t & Ax & & = & b & \\ & -B^{\omega}x + & D^{\omega}y^{\omega} & = & d^{\omega} & \\ & x, & y^{\omega} & \geq & 0, & \omega \in \Omega. \end{array}$$

where x denotes the first-stage, y^ω the second-stage decision variables, c represents the first-stage and f^ω the second-stage objective coefficients, A, b represent the coefficients and right hand sides of the first-stage constraints, and B^ω, D^ω, d^ω represent the parameters of the second-stage constraints, where the transition matrix B^ω couples the two stages. In the literature D^ω is often referred to as the technology matrix or recourse matrix. The first stage parameters are known with certainty. The second stage parameters are random parameters that assume outcomes labeled ω with probability $p(\omega)$, where Ω denotes the set of all possible outcome labels.

At the time the first-stage decision x has to be made, the second-stage parameters are only known by their probability distribution of possible outcomes. Later, after x has already been determined, an actual outcome of the second-stage parameters will become known, and the second-stage decision y^ω is made based on knowledge of the actual outcome ω. The objective is to find a feasible decision x that minimizes the total expected costs, the sum of first-stage costs and expected second-stage costs.

For discrete distributions of the random parameters, the stochastic linear program can be represented by the corresponding *equivalent deterministic linear program*:

$$\begin{array}{rrrrrrrrrrrrr}
\min\ z & = & cx & + & p^1fy^1 & + & p^2fy^2 & + & \cdots & + & p^Wfy^W & & \\
s/t & & Ax & & & & & & & & & = & b \\
 & & -B^1x & + & Dy^1 & & & & & & & = & d^1 \\
 & & -B^2x & & & + & Dy^2 & & & & & = & d^2 \\
 & & \vdots & & & & & & \ddots & & & & \vdots \\
 & & -B^Wx & & & & & & & + & Dy^W & = & d^W \\
 & & x, & & y^1, & & y^2, & & \ldots, & & y^W & \geq & 0,
\end{array}$$

which contains all possible outcomes $\omega \in \Omega$. Note that for practical problems W is very large, e.g., a typical number could be 10^{20}, and the resulting equivalent deterministic linear problem is too large to be solved directly.

In order to see the two-stage nature of the underlying decision making process the folowing representation is also often used:

$$\begin{array}{rrrr}
\min\ cx & + & E\ z^\omega(x) & \\
Ax & & & = b \\
x & & & \geq 0
\end{array}$$

where

$$\begin{array}{rrrl}
z^\omega(x) = \min\ f^\omega y^\omega & & & \\
D^\omega y^\omega & = & d^\omega + B^\omega x & \\
y^\omega & \geq & 0,\ \omega \in \Omega = \{1, 2, \ldots, W\}. &
\end{array}$$

DECIS employs different strategies to solve two-stage stochastic linear programs. It computes an exact optimal solution to the problem or approximates the true optimal solution very closely and gives a confidence interval within which the true optimal objective lies with, say, 95% confidence.

5.13.1.3 Representing Uncertainty

It is favorable to represent the uncertain second-stage parameters in a structure. Using $V = (V_1, \ldots, V_h)$ an h-dimensional independent random vector parameter that assumes outcomes $v^\omega = (v_1, \ldots, v_h)^\omega$ with probability $p^\omega = p(v^\omega)$, we represent the uncertain second-stage parameters of the problem as functions of the independent random parameter V:

$$f^\omega = f(v^\omega), \quad B^\omega = B(v^\omega), \quad D^\omega = D(v^\omega), \quad d^\omega = d(v^\omega).$$

Each component V_i has outcomes $v_i^{\omega_i}$, $\omega_i \in \Omega_i$, where ω_i labels a possible outcome of component i, and Ω_i represents the set of all possible outcomes of component i. An outcome of the random vector

$$v^\omega = (v_1^{\omega_1}, \ldots, v_h^{\omega_h})$$

consists of h independent component outcomes. The set

$$\Omega = \Omega_1 \times \Omega_2 \times \ldots \times \Omega_h$$

represents the crossing of sets Ω_i. Assuming each set Ω_i contains W_i possible outcomes, $|\Omega_i| = W_i$, the set Ω contains $W = \prod W_i$ elements, where $|\Omega| = W$ represents the number of all possible outcomes of the random vector V. Based on independence, the joint probability is the product

$$p^\omega = p_1^{\omega_1} p_2^{\omega_2} \cdots p_h^{\omega_h}.$$

Let η denote the vector of all second-stage random parameters, e.g., $\eta = \text{vec}(f, B, D, d)$. The outcomes of η may be represented by the following general linear dependency model:

$$\eta^\omega = \text{vec}(f^\omega, B^\omega, d^\omega, d^\omega) = Hv^\omega, \omega \in \Omega$$

where H is a matrix of suitable dimensions. DECIS can solve problems with such general linear dependency models.

5.13.1.4 Solving the Universe Problem

We refer to the universe problem if we consider all possible outcomes $\omega \in \Omega$ and solve the corresponding problem exactly. This is not always possible, because there may be too many possible realizations $\omega \in \Omega$. For solving the problem DECIS employs Benders decomposition, splitting the problem into a master problem, corresponding to the first-stage decision, and into subproblems, one for each $\omega \in \Omega$, corresponding to the second-stage decision. The details of the algorithm and techniques used for solving the universe problem are discussed in The DECIS User's Manual.

Solving the universe problem is referred to as strategy 4. Use this strategy only if the number of universe scenarios is reasonably small. There is a maximum number of universe scenarios DECIS can handle, which depends on your particular resources.

5.13.1.5 Solving the Expected Value Problem

The expected value problem results from replacing the stochastic parameters by their expectation. It is a linear program that can also easily be solved by employing a solver directly. Solving the expected value problem may be useful by itself (for example as a benchmark to compare the solution obtained from solving the stochastic problem), and it also may yield a good starting solution for solving the stochastic problem. DECIS solves the expected value problem using Benders decomposition. The details of generating the expected value problem and the algorithm used for solving it are discussed in the DECIS User's Manual. To solve the expected value problem choose strategy 1.

5.13.1.6 Using Monte Carlo Sampling

As noted above, for many practical problems it is impossible to obtain the universe solution, because the number of possible realizations $|\Omega|$ is way too large. The power of DECIS lies in its ability to compute excellent approximate solutions by employing Monte Carlo sampling techniques. Instead of computing the expected cost and the coefficients and the right-hand sides of the Benders cuts exactly (as it is done when solving the universe problem), DECIS, when using Monte Carlo sampling, estimates the quantities in each iteration using an independent sample drawn from the distribution of the random parameters. In addition to using crude Monte Carlo, DECIS uses importance sampling or control variates as variance reduction techniques.

The details of the algorithm and the different techniques used are described in the DECIS User's Manual. You can choose crude Monte Carlo, referred to as strategy 6, Monte Carlo importance sampling, referred to as strategy 2, or control variates, referred to as strategy 10. Both Monte Carlo importance sampling and control variates have been shown for many problems to give a better approximation compared to employing crude Monte Carlo sampling.

When using Monte Carlo sampling DECIS computes a close approximation to the true solution of the problem, and estimates a close approximation of the true optimal objective value. It also computes a confidence interval within which the true optimal objective of the problem lies, say with 95% confidence. The confidence interval is based on rigorous statistical theory. An outline of how the confidence interval is computed is given in the DECIS User's Manual. The size of the confidence interval depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for the estimation. You can expect the confidence interval to be very small, especially when you employ importance sampling or control variates as a variance reduction technique.

When employing Monte Carlo sampling techniques you have to choose a sample size (set in the parameter file). Clearly, the larger the sample size the better will be the approximate solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Setting the sample size too small may lead to bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

5.13.1.7 Monte Carlo Pre-sampling

We refer to pre-sampling when we first take a random sample from the distribution of the random parameters and then generate the approximate stochastic problem defined by the sample. The obtained approximate problem is then solved exactly using decomposition. This is in contrast to the way we used Monte Carlo sampling in the previous section, where we used Monte Carlo sampling in each iteration of the decomposition.

The details of the techniques used for pre-sampling are discussed in the DECIS User's Manual. DECIS computes the exact solution of the sampled problem using decomposition. This solution is an approximate solution of the original stochastic problem. Besides this approximate solution, DECIS computes an estimate of the expected cost corresponding to this approximate solution and a confidence interval within which the true optimal objective of the original stochastic problem lies with, say, 95% confidence. The

confidence interval is based on statistical theory, while its size depends on the variance of the second-stage cost of the stochastic problem and on the sample size used for generating the approximate problem. In conjunction with pre-sampling no variance reduction techniques are currently implemented.

Using Monte Carlo pre-sampling you have to choose a sample size. Clearly, the larger the sample size you choose, the better will be the solution DECIS computes, and the smaller will be the confidence interval for the true optimal objective value. The default value for the sample size is 100. Again, setting the sample size too small may lead to a bias in the estimation of the confidence interval, therefore the sample size should be at least 30.

To use Monte Carlo pre-sampling choose strategy 8.

5.13.1.8 Regularized Decomposition

When solving practical problems, the number of Benders iterations can be quite large. In order to control the decomposition, with the hope to reduce the iteration count and the solution time, DECIS makes use of regularization. When employing regularization, an additional quadratic term is added to the objective of the master problem, representing the square of the distance between the best solution found so far (the incumbent solution) and the variable x. Using this term, DECIS controls the distance between solutions in successive decomposition iterations.

To enable regularization you have to set the corresponding parameter. You also have to choose the value of the constant rho in the regularization term. The default is regularization disabled. Details of how DECIS carries out regularization are represented in the DECIS User's Manual.

Regularization is only implemented when using MINOS as the optimizer for solving subproblems. Regularization has proven to be helpful for problems that need a large number of Benders iteration when solved without regularization. Problems that need only a small number of Benders iterations without regularization are not expected to improve much with regularization, and may need even more iterations with regularization than without.

5.13.2 GAMS/DECIS

GAMS stands for General Algebraic Modeling Language, and is one of the most widely used modeling languages. Using DECIS directly from GAMS spares you from worrying about all the details of the input formats. It makes the problem formulation much easier but still gives you almost all the flexibility of using DECIS directly.

The link from GAMS to DECIS has been designed in such a way that almost no extensions to the GAMS modeling language were necessary for carrying out the formulation and solution of stochastic programs. In a future release of GAMS, however, additions to the language are planned that will allow you to model stochastic programs in an even more elegant way.

5.13.2.1 Setting up a Stochastic Program Using GAMS/DECIS

The interface from GAMS to DECIS supports the formulation and solution of stochastic *linear* programs. DECIS solves them using two-stage decomposition. The GAMS/DECIS interface resembles closely the structure of the SMPS (stochastic mathematical programming interface) discussed in the DECIS User's Manual. The specification of a stochastic problem using GAMS/DECIS uses the following components:

- the deterministic (core) model,
- the specification of the decision stages,
- the specification of the random parameters, and
- setting DECIS as the optimizer to be used.

5.13.2.2 Starting with the Deterministic Model

The core model is a deterministic linear program where all random parameters are replaced by their mean or by a particular realization. One could also see it as a GAMS model without any randomness. It could be a deterministic model that you have, that you intend to expand to a stochastic one. Using DECIS with GAMS allows you to easily extend a deterministic linear programming model to a stochastic one. For example, the following GAMS model represents a deterministic version of the electric power expansion planning illustrative example discussed in Infanger (1994).

```
*   APL1P test model
*   Dr. Gerd Infanger, November 1997
*   Deterministic Program

set g generators / g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1  0.68,  g2  0.64 /;
parameter ccmin(g) min capacity / g1  1000,  g2  1000 /;
parameter ccmax(g) max capacity / g1 10000,  g2 10000 /;
parameter c(g) investment       / g1   4.0,  g2   2.5 /;

table f(g,dl) operating cost
             h    m    l
   g1        4.3  2.0  0.5
   g2        8.7  4.0  1.0;

parameter d(dl) demand                   / h  1040, m  1040, l  1040 /;
parameter us(dl) cost of unserved demand / h    10, m    10, l    10 /;

free variable tcost            total cost;
positive variable x(g)         capacity of generators;
positive variable y(g, dl)     operating level;
positive variable s(dl)        unserved demand;

equations
cost        total cost
cmin(g)     minimum capacity
cmax(g)     maximum capacity
omax(g)     maximum operating level
demand(dl)  satisfy demand;

cost .. tcost =e=     sum(g, c(g)*x(g))
                    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                    + sum(dl,us(dl)*s(dl));

cmin(g) ..    x(g) =g= ccmin(g);
cmax(g) ..    x(g) =l= ccmax(g);
omax(g) ..    sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

option lp=minos;
solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
```

```
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l -  ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
```

5.13.2.3 Setting the Decision Stages

Next in order to extend a deterministic model to a stochastic one you must specify the decision stages. DECIS solves stochastic programs by two-stage decomposition. Accordingly, you must specify which variables belong to the first stage and which to the second stage, as well as which constraints are first-stage constraints and which are second-stage constraints. First stage constraints involve only first-stage variables, while second-stage constraints involve both first- and second-stage variables. You must specify the stage of a variable or a constraint by setting the stage suffix ".STAGE" to either one or two depending on if it is a first or second stage variable or constraint. For example, expanding the illustrative model above by

```
* setting decision stages
x.stage(g)       = 1;
y.stage(g, dl)   = 2;
s.stage(dl)      = 2;
cmin.stage(g)    = 1;
cmax.stage(g)    = 1;
omax.stage(g)    = 2;
demand.stage(dl) = 2;
```

would make x(g) first-stage variables, y(g, dl) and s(dl) second-stage variables, cmin(g) and cmax(g) first-stage constraints, and omax(g) and demand(g) second-stage constraints. The objective is treated separately, you don't need to set the stage suffix for the objective variable and objective equation.

Note that the use of the `.stage` variable and equation suffix causes the GAMS scaling facility (i.e. the `.scale` suffix) to be unavailable. Stochastic models have to be scaled manually.

5.13.2.4 Specifying the Stochastic Model

DECIS supports any linear dependency model, i.e., the outcomes of an uncertain parameter in the linear program are a linear function of a number of independent random parameter outcomes. DECIS considers only discrete distributions: you must approximate any continuous distributions by discrete ones. The number of possible realizations of the discrete random parameters determines the accuracy of the approximation. A special case of a linear dependency model arises when you have only independent random parameters in your model. In this case the independent random parameters are mapped one to one into the random parameters of the stochastic program. We will present the independent case first and then expand to the case with linear dependency. According to setting up a linear dependency model we present the formulation in GAMS by first defining independent random parameters and then defining the distributions of the uncertain parameters in your model.

Specifying Independent Random Parameters

There are of course many different ways you can set up independent random parameters in GAMS. In the following we show one possible way that is generic and thus can be adapted for different models. The set-up uses the set stoch for labeling outcome named "out" and probability named "pro" of each independent random parameter. In the following we show how to define an independent random parameter, say, v1. The formulation uses the set omega1 as driving set, where the set contains one element for each possible realization the random parameter can assume. For example, the set omega1 has four

elements according to a discrete distribution of four possible outcomes. The distribution of the random parameter is defined as the parameter v1, a two-dimensional array of outcomes "out" and corresponding probability "pro" for each of the possible realizations of the set omega1, "o11", "o12", "o13", and "o14". For example, the random parameter v1 has outcomes of $-1.0, -0.9, -0.5, -0.1$ with probabilities $0.2, 0.3, 0.4, 0.1$, respectively. Instead of using assignment statements for inputting the different realizations and corresponding probabilities you could also use the table statement. Always make sure that the sum of the probabilities of each independent random parameter adds to one.

```
* defining independent stochastic parameters
set stoch /out, pro /;
set omega1 / o11, o12, o13, o14 /;

table v1(stoch, omega1)
      o11   o12   o13   o14
out  -1.0  -0.9  -0.5  -0.1
pro   0.2   0.3   0.4   0.1
;
```

Random parameter v1 is the first out of five independent random parameters of the illustrative model APL1P, where the first two represent the independent availabilities of the generators g1 and g2 and the latter three represent the independent demands of the demand levels h, m, and l. We also represent the definitions of the remaining four independent random parameters. Note that random parameters v3, v4, and v5 are identically distributed.

```
set omega2 / o21, o22, o23, o24, o25 /;
table v2(stoch, omega2)
      o21   o22   o23   o24   o25
out  -1.0  -0.9  -0.7  -0.1  -0.0
pro   0.1   0.2   0.5   0.1   0.1
;


set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;


set omega4 / o41, o42, o43, o44 /;
table v4(stoch,omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;

set omega5 / o51, o52, o53, o54 /;
table v5(stoch,omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;
```

Defining the Distributions of the Uncertain Parameters in the Model

Having defined the independent stochastic parameters (you may copy the setup above and adapt it for your model), we next define the stochastic parameters in the GAMS model. The stochastic parameters of the model are defined by writing a file, the GAMS stochastic file, using the put facility of GAMS. The GAMS stochastic file resembles closely the stochastic file of the SMPS input format. The main difference is that we use the row, column, bounds, and right hand side names of the GAMS model and that we can write it in free format.

Independent Stochastic Parameters

First we describe the case where all stochastic parameters in the model are independent, see below the representation of the stochastic parameters for the illustrative example APL1P, which has five independent stochastic parameters.

First define the GAMS stochastic file "MODEL.STG" (only the exact name in uppercase letters is supported) and set up GAMS to write to it. This is done by the first two statements. You may want to consult the GAMS manual for how to use put for writing files. The next statement "INDEP DISCRETE" indicates that a section of independent stochastic parameters follows. We then write all possible outcomes and corresponding probabilities for each stochastic parameter by using loop statements. Of course one could also write each line separately, but the loops work nicely. Writing a "∗" between the definitions of the independent stochastic parameters is merely for optical reasons and can be omitted.

```
* defining distributions (writing file MODEL.STG)
file stg /MODEL.STG/;
put stg;

put "INDEP DISCRETE" /;
loop(omega1,
put "x g1  omax g1   ", v1("out", omega1), " period2 ", v1("pro", omega1) /;
);
put "*" /;
loop(omega2,
put "x g2  omax g2   ", v2("out", omega2), " period2 ", v2("pro", omega2) /;
);
put "*" /;
loop(omega3,
put "RHS  demand h   ", v3("out", omega3), " period2 ", v3("pro", omega3) /;
);
put "*" /;
loop(omega4,
put "RHS  demand m   ", v4("out", omega4), " period2 ", v4("pro", omega4) /;
)
put "*" /;
loop(omega5,
put "RHS  demand l   ", v5("out", omega5), " period2 ", v5("pro", omega5) /;
);
putclose stg;
```

In the example APL1P the first stochastic parameter is the availability of generator g1. In the model the parameter appears as the coefficient of variable x(g1) in equation omax(g1). The definition using the put statement first gives the stochastic parameter as the intersection of variable x(g1) with equation omax(g1), but without having to type the braces, thus *x g1 omax g1*, then the outcome *v1("out", omega1)* and the probability *v1("pro", omega1)* separated by *"period2"*. The different elements of the statement must be separated by blanks. Since the outcomes and probabilities of the first stochastic parameters are

driven by the set omega1 we loop over all elements of the set omega1. We continue and define all possible outcomes for each of the five independent stochastic parameters.

In the example of independent stochastic parameters, the specification of the distribution of the stochasic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

```
INDEP DISCRETE
x g1  omax g1          -1.00 period2         0.20
x g1  omax g1          -0.90 period2         0.30
x g1  omax g1          -0.50 period2         0.40
x g1  omax g1          -0.10 period2         0.10
*
x g2  omax g2          -1.00 period2         0.10
x g2  omax g2          -0.90 period2         0.20
x g2  omax g2          -0.70 period2         0.50
x g2  omax g2          -0.10 period2         0.10
x g2  omax g2           0.00 period2         0.10
*
RHS  demand h         900.00 period2         0.15
RHS  demand h        1000.00 period2         0.45
RHS  demand h        1100.00 period2         0.25
RHS  demand h        1200.00 period2         0.15
*
RHS  demand m         900.00 period2         0.15
RHS  demand m        1000.00 period2         0.45
RHS  demand m        1100.00 period2         0.25
RHS  demand m        1200.00 period2         0.15
*
RHS  demand l         900.00 period2         0.15
RHS  demand l        1000.00 period2         0.45
RHS  demand l        1100.00 period2         0.25
RHS  demand l        1200.00 period2         0.15
```

For defining stochastic parameters in the right-hand side of the model use the keyword *RHS* as the column name, and the equation name of the equation which right-hand side is uncertain, see for example the specification of the uncertain demands *RHS demand h*, *RHS demand m*, and *RHS demand l*. For defining uncertain bound parameters you would use the keywords *UP*, *LO*, or *FX*, the string *bnd*, and the variable name of the variable whose upper, lower, or fixed bound is uncertain.

Note all the keywords for the definitions are in capital letters, i.e., "INDEP DISCRETE", "RHS", and not represented in the example "UP", "LO", and "FX".

Note that in GAMS equations, variables may appear in the right-hand side, e.g. `"EQ.. X+1 =L= 2*Y"`. When the coefficient 2 is a random variable, we need to be aware that GAMS will generate the following LP row `X - 2*Y =L= -1`. Suppose the probability distribution of this random variable is given by:

```
set s scenario /pessimistic, average, optimistic/;
parameter outcome(s) / pessimistic 1.5
                       average     2.0
                       optimistic  2.3 /;
parameter prob(s)    / pessimistic 0.2
                       average     0.6
                       optimistic  0.2 /;
```

then the correct way of generating the entries in the stochastic file would be:

```
loop(s,
   put  "Y EQ ",(-outcome(s))," PERIOD2 ",prob(s)/;
);
```

Note the negation of the outcome parameter. Also note that expressions in a PUT statement have to be surrounded by parentheses. GAMS reports in the *row listing* section of the listing file how equations are generated. You are encouraged to inspect the row listing to discover or confirm how coefficients appear in a generated LP row.

Dependent Stochastic Parameters

Next we describe the case of general linear dependency of the stochastic parameters in the model, see below the representation of the stochastic parameters for the illustrative example APL1PCA, which has three dependent stochastic demands driven by two independent stochastic random parameters. First we give the definition of the two independent stochastic parameters, which in the example happen to have two outcomes each.

```
* defining independent stochastic parameters
set stoch /out, pro/;

set omega1 / o11, o12 /;
table v1(stoch,omega1)
       o11  o12
out    2.1  1.0
pro    0.5  0.5 ;

set omega2 / o21, o22 /;
table v2(stoch, omega2)
       o21  o22
out    2.0  1.0
pro    0.2  0.8 ;
```

We next define the parameters of the transition matrix from the independent stochastic parameters to the dependent stochastic parameters of the model. We do this by defining two parameter vectors, where the vector $hm1$ gives the coefficients of the independent random parameter $v1$ in each of the three demand levels and the vector $hm2$ gives the coefficients of the independent random parameter $v2$ in each of the three demand levels.

```
parameter hm1(dl) / h  300., m  400., l  200. /;
parameter hm2(dl) / h  100., m  150., l  300. /;
```

Again first define the GAMS stochastic file "MODEL.STG" and set GAMS to write to it. The statement *BLOCKS DISCRETE* indicates that a section of linear dependent stochastic parameters follows.

```
* defining distributions (writing file MODEL.STG)
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /;
scalar h1;
loop(omega1,
put "BL  v1   period2 ", v1("pro", omega1)/;
```

```
loop(dl,
h1 = hm1(dl) * v1("out", omega1);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
loop(omega2,
put " BL v2   period2 ", v2("pro", omega2) /;
loop(dl,
h1 = hm2(dl) * v2("out", omega2);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
putclose stg;
```

Dependent stochastic parameters are defined as functions of independent random parameters. The keyword *BL* labels a possible realization of an independent random parameter. The name besides the *BL* keyword is used to distinguish between different outcomes of the same independent random parameter or a different one. While you could use any unique names for the independent random parameters, it appears natural to use the names you have already defined above, e.g., $v1$ and $v2$. For each realization of each independent random parameter define the outcome of every dependent random parameter (as a function of the independent one). If a dependent random parameter in the GAMS model depends on two or more different independent random parameters the contributions of each of the independent parameters are added. We are therefore in the position to model any linear dependency model. (Note that the class of models that can be accommodated here is more general than linear. The functions, with which an independent random variable contributes to the dependent random variables can be any ones in one argument. As a general rule, any stochastic model that can be estimated by linear regression is supported by GAMS/DECIS.)

Define each independent random parameter outcome and the probability associated with it. For example, the statement starting with *BL v1 period2* indicates that an outcome of (independent random parameter) *v1* is being defined. The name *period2* indicates that it is a second-stage random parameter, and *v1("pro", omega1)* gives the probability associated with this outcome. Next list all random parameters dependent on the independent random parameter outcome just defined. Define the dependent stochastic parameter coefficients by the GAMS variable name and equation name, or "RHS" and variable name, together with the value of the parameter associated with this realization. In the example, we have three dependent demands. Using the scalar $h1$ for intermediately storing the results of the calculation, looping over the different demand levels *dl* we calculate *h1 = hm1(dl) * v1("out", omega1)* and define the dependent random parameters as the right-hand sides of equation *demand*(*dl*).

When defining an independent random parameter outcome, if the block name is the same as the previous one (e.g., when *BL v1* appears the second time), a different outcome of the same independent random parameter is being defined, while a different block name (e.g., when *BL v2* appears the first time) indicates that the first outcome of a different independent random parameter is being defined. You must ensure that the probabilities of the different outcomes of each of the independent random parameters add up to one. The loop over all elements of *omega*1 defines all realizations of the independent random parameter $v1$ and the loop over all elements of *omega*2 defines all realizations of the independent random parameter $v2$.

Note that for the first realization of an independent random parameter, you *must* define all dependent parameters and their realizations. The values entered serve as a base case. For any other realization of an independent random parameter you only need to define the dependent parameters that have different coefficients than have been defined in the base case. For those not defined in a particular realization, their values of the base case are automatically added.

In the example of dependent stochastic parameters above, the specification of the distribution of the stochastic parameters using the put facility creates the following file "MODEL.STG", which then is processed by the GAMS/DECIS interface:

```
BLOCKS DISCRETE
BL  v1   period2         0.50
RHS demand h       630.00
RHS demand m       840.00
RHS demand l       420.00
BL  v1   period2         0.50
RHS demand h       300.00
RHS demand m       400.00
RHS demand l       200.00
 BL v2   period2         0.20
RHS demand h       200.00
RHS demand m       300.00
RHS demand l       600.00
 BL v2   period2         0.80
RHS demand h       100.00
RHS demand m       150.00
RHS demand l       300.00
```

Again all the keywords for the definitions are in capital letters, i.e., "BLOCKS DISCRETE", "BL", "RHS", and not represented in the example "UP", "LO", and "FX".

Note that you can only define random parameter coefficients that are nonzero in your GAMS model. When setting up the deterministic core model put a nonzero entry as a placeholder for any coefficient that you wish to specify as a stochastic parameter. Specifying a random parameter at the location of a zero coefficient in the GAMS model causes DECIS to terminate with an error message.

5.13.2.5 Setting DECIS as the Optimizer

After having finished the stochastic definitions you must set DECIS as the optimizer. This is done by issuing the following statements:

```
* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;
```

The statement *option lp = decism* sets DECIS with the MINOS LP engine as the optimizer to be used for solving the stochastic problem. Note that if you do not use DECIS, but instead use any other linear programming optimizer, your GAMS model will still run and optimize the deterministic core model that you have specified. The statement *apl1p.optfile = 1* directs GAMS to process the file `DECIS.OPT`, in which you may define any DECIS parameters.

Setting Parameter Options in the GAMS Model

The options iteration limit and resource limit can be set directly in your GAMS model file. For example, the following statements

```
option iterlim = 1000;
option reslim = 6000;
```

limit the number of decomposition iterations to be less than or equal to 1000, and the elapsed time for running DECIS to be less than or equal to 6000 seconds or 100 minutes.

Setting Parameters in the DECIS Options File

In the DECIS options file you can specify parameters regarding the solution algorithm used and control the output of the DECIS program. There is a record for each parameter you want to specify. Each record consists of the value of the parameter you want to specify and the keyword identifying the parameter, separated by a blank character or a comma. You may specify parameters with the following keywords: istrat, nsamples, nzrows, iwrite, ibug, iscratch, ireg, rho, tolben, and tolw *in any order*. Each keyword can be specified in lower case or upper case text in the format (A10). Since DECIS reads the records in free format you don't have to worry about the format, but some computers require that the text is inputted in quotes. Parameters that are not specified in the parameter file automatically assume their default values. See details of these parameters from Section Description of GAMS/DECIS Options.

Example

In the following example the parameters istrat = 7, nsamples = 200, and nzrows = 200 are specified. All other parameters are set at their default values. DECIS first solves the expected value problem and then the stochastic problem using crude Monte Carlo sampling with a sample size of nsamples = 200. DECIS reserves space for a maximum of nzrows = 50 cuts.

```
7       "ISTRAT"
200     "NSAMPLES"
50      "NZROWS"
```

Setting MINOS Parameters in the MINOS Specification File

When you use MINOS as the optimizer for solving the master and the subproblems, you must specify optimization parameters in the MINOS specification file "MINOS.SPC". Each record of the file corresponds to the specification of one parameter and consists of a keyword and the value of the parameter in free format. Records having a "$*$" as their first character are considered as comment lines and are not further processed. For a detailed description of these parameters, see the MINOS Users' Guide (Murtagh and Saunders (1983) [142] . The following parameters should be specified with some consideration:

- AIJ TOLERANCE — Specifies the nonzero tolerance for constraint matrix elements of the problem. Matrix elements a_{ij} that have a value for which $|a_{ij}|$ is less than "AIJ TOLERANCE" are considered by MINOS as zero and are automatically eliminated from the problem. It is wise to specify "AIJ TOLERANCE 0.0 "
- SCALE — Specifies MINOS to scale the problem ("SCALE YES") or not ("SCALE NO"). It is wise to specify "SCALE NO".
- ROWS — Specifies the number of rows in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ROWS" should be specified as the number of constraints in the core problem or greater.
- COLUMNS — Specifies the number of columns in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "COLUMNS" should be specified as the number of variables in the core problem or greater.
- ELEMENTS — Specifies the number of nonzero matrix coefficients in order for MINOS to reserve the appropriate space in its data structures when reading the problem. "ELEMENTS" should be specified as the number of nonzero matrix coefficients in the core problem or greater.

Example

The following example represents typical specifications for running DECIS with MINOS as the optimizer.

```
BEGIN SPECS
PRINT LEVEL                1
LOG FREQUENCY             10
SUMMARY FREQUENCY         10
MPS FILE                  12
ROWS                   20000
COLUMNS                50000
ELEMENTS              100000
ITERATIONS LIMIT       30000
*
FACTORIZATION FREQUENCY  100
AIJ TOLERANCE            0.0
*
SCALE                     NO
END OF SPECS
```

Setting CPLEX Parameters Using System Environment Variables

When you use CPLEX as the optimizer for solving the master and the subproblems, optimization parameters must be specified through system environment variables. You can specify the parameters "CPLEXLICDIR", "SCALELP", "NOPRESOLVE", "ITERLOG", "OPTIMALITYTOL", "FEASIBILITYTOL", and "DUALSIMPLEX".

- CPLEXLICDIR — Contains the path to the CPLEX license directory. For example, on a Unix system with the CPLEX license directory in /usr/users/cplex/cplexlicdir you issue the command *setenv CPLEXLICDIR /usr/users/cplex/cplexlicdir*.
- SCALELP — Specifies CPLEX to scale the master and subproblems before solving them. If the environment variable is not set no scaling is used. Setting the environment variable, e.g., by issuing the command *setenv SCALELP yes*, scaling is switched on.
- NOPRESOLVE — Allows to switch off CPLEX's presolver. If the environment variable is not set, presolve will be used. Setting the environment variable, e.g., by setting *setenv NOPRESOLVE yes*, no presolve will be used.
- ITERLOG — Specifies the iteration log of the CPLEX iterations to be printed to the file "MODEL.CPX". If you do not set the environment variable no iteration log will be printed. Setting the environment variable, e.g., by setting *setenv ITERLOG yes*, the CPLEX iteration log is printed.
- OPTIMALITYTOL — Specifies the optimality tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting *setenv OPTIMALITYTOL 1.0E-7* sets the CPLEX optimality tolerance to 0.0000001.
- FEASIBILIITYTOL — Specifies the feasibility tolerance for the CPLEX optimizer. If you do not set the environment variable the CPLEX default values are used. For example, setting *setenv FEASIBILITYTOL 1.0E-7* sets the CPLEX optimality tolerance to 0.0000001.
- DUALSIMPLEX — Specifies the dual simplex algorithm of CPLEX to be used. If the environment variable is not set the primal simplex algorithm will be used. This is the default and works beautifully for most problems. If the environment variable is set, e.g., by setting *setenv DUALSIMPLEX yes*, CPLEX uses the dual simplex algorithm for solving both master and subproblems.

5.13.2.6 GAMS/DECIS Output

After successfully having solved a problem, DECIS returns the objective, the optimal primal and optimal dual solution, the status of variables (if basic or not), and the status of equations (if binding or not) to GAMS. In the case of first-stage variables and equations you have all information available in GAMS, as you would with any other solver, but instead of obtaining the optimal values for a deterministic core problem you actually have the optimal values for the stochastic problem. However, for second-stage variables and constraints the e xpected values of the optimal primal and optimal dual solution are reported. This saves space and is useful for the calculation of risk measures. However, the information as to what the optimal primal and dual solutions were in the different scenarios of the stochastic programs is not reported back to GAMS. In a future release of the GAMS/DECIS interface the GAMS language is planned to be extended to handle the scenario second-stage optimal primal and dual values at least for selected variables and equations.

While running, DECIS outputs important information about the progress of the execution to your computer screen. After successfully solving a problem, DECIS also outputs its optimal solution to the solution output file "MODEL.SOL". The debug output file "MODEL.SCR" contains important information about the optimization run, and the optimizer output files "MODEL.MO" (when using DECIS with MINOS) or "MODEL.CPX" (when using DECIS with CPLEX) contain solution output from the optimizer used. In the DECIS User's Guide you find a detailed discussion of how to interpret the screen output, the solution report and the information in the output files.

The Screen Output

The output to the screen allows you to observe the progress of a DECIS run. After the program logo and the copyright statement, you see four columns of output being written to the screen as long as the program proceeds. The first column (from left to right) represents the iteration count, the second column the lower bound (the optimal objective of the master problem), the third column the best upper bound (exact value or estimate of the total expected cost of the best solution found so far), and the fourth column the current upper bound (exact value or estimate of the total expected cost of current solution). After successful completion, DECIS quits with "Normal Exit", otherwise, if an error has been encountered, the programs stops with the message "Error Exit".

Example

When solving the illustrative example APL1P using strategy 5, we obtain the following report on the screen:

```
 T H E   D  E  C  I  S   S Y S T E M
 Copyright (c) 1989  -- 1999 by Dr. Gerd Infanger
 All rights reserved.

iter                lower           best upper        current upper

   0          -0.9935E+06
   1          -0.4626E+06           0.2590E+05           0.2590E+05
   2           0.2111E+05           0.2590E+05           0.5487E+06
   3           0.2170E+05           0.2590E+05           0.2697E+05
   4           0.2368E+05           0.2384E+05           0.2384E+05
   5           0.2370E+05           0.2384E+05           0.2401E+05
   6           0.2370E+05           0.2370E+05           0.2370E+05

iter                lower           best upper        current upper

   6           0.2370E+05
   7           0.2403E+05           0.2470E+05           0.2470E+05
   8           0.2433E+05           0.2470E+05           0.2694E+05
   9           0.2441E+05           0.2470E+05           0.2602E+05
```

```
 10          0.2453E+05          0.2470E+05          0.2499E+05
 11          0.2455E+05          0.2470E+05          0.2483E+05
 12          0.2461E+05          0.2467E+05          0.2467E+05
 13          0.2461E+05          0.2467E+05          0.2469E+05
 14          0.2461E+05          0.2465E+05          0.2465E+05
 15          0.2463E+05          0.2465E+05          0.2467E+05
 16          0.2463E+05          0.2465E+05          0.2465E+05
 17          0.2464E+05          0.2465E+05          0.2465E+05
 18          0.2464E+05          0.2464E+05          0.2464E+05
 19          0.2464E+05          0.2464E+05          0.2464E+05
 20          0.2464E+05          0.2464E+05          0.2464E+05
 21          0.2464E+05          0.2464E+05          0.2464E+05
 22          0.2464E+05          0.2464E+05          0.2464E+05

Normal Exit
```

The Solution Output File

The solution output file contains the solution report from the DECIS run. Its name is "MODEL.SOL". The file contains the best objective function value found, the corresponding values of the first-stage variables, the corresponding optimal second-stage cost, and a lower and an upper bound on the optimal objective of the problem. In addition, the number of universe scenarios and the settings for the stopping tolerance are reported. In the case of using a deterministic strategy for solving the problem, exact values are reported. When using Monte Carlo sampling, estimated values, their variances, and the sample size used for the estimation are reported. Instead of exact upper and lower bounds, probabilistic upper and lower bounds, and a 95% confidence interval, within which the true optimal solution lies with 95% confidence, are reported. A detailed description of the solution output file can be found in the DECIS User's Guide.

The Debug Output File

The debug output file contains the standard output of a run of DECIS containing important information about the problem, its parameters, and its solution. It also contains any error messages that may occur during a run of DECIS. In the case that DECIS does not complete a run successfully, the cause of the trouble can usually be located using the information in the debug output file. If the standard output does not give enough information you can set the debug parameter ibug in the parameter input file to a higher value and obtain additional debug output. A detailed description of the debug output file can be found in the DECIS User's Guide.

The Optimizer Output Files

The optimizer output file "MODEL.MO" contains all the output from MINOS when called as a subroutine by DECIS. You can specify what degree of detail should be outputted by setting the appropriate "PRINT LEVEL" in the MINOS specification file. The optimizer output file "MODEL.CPX" reports messages and the iteration log (if switchwd on using the environment variable) from CPLEX when solving master and sub problems.

5.13.3 Description of GAMS/DECIS Options

5.13.3.1 DECIS Solver Options

Option	Description	Default
IBug	Debug output	0

Option	Description	Default
IReg	Indicator for regularized decomposition - MINOS only	0
IScratch	Internal unit number for output and debug	17
IStrat	Defines the solution strategy used	3
IWrite	Subproblem output	0
NSamples	Sample size used for the estimation	100
NZRows	Number of rows reserved for cuts in the master problem	100
Rho	rho parameter of regularization term in the objective function	1000
TolBen	Tolerance for stopping the decomposition algorithm	1e-7
TolW	tolerance when writing debug solution output	1e-9

IBug *(integer)*: Debug output ↩

Default: 0

value	meaning
0	DECIS does not write any debug output
1	Solution of the master problem on each iteration
2	Value 1 plus scenario index and the optimal objective value for each subproblem solved
3	Value 2 plus information regarding importance sampling
4	Value 3 plus optimal dual variables of the cuts
5	Value 4 plus coeffcients and the right-hand side of the cuts
6	Value 5 plus dump of the master problem and the subproblem in MPS format

IReg *(boolean)*: Indicator for regularized decomposition - MINOS only ↩

Default: 0

IScratch *(integer)*: Internal unit number for output and debug ↩

Range: {1, ..., ∞}

Default: 17

IStrat *(integer)*: Defines the solution strategy used ↩

Default: 3

value	meaning
1	Solves the expected value problem
2	Solves the stochastic problem using Monte Carlo importance sampling
3	Refers to istrat = 1 plus istrat = 2
4	Solves the stochastic universe problem
5	Refers to istrat = 1 plus istrat = 4
6	Solves the stochastic problem using crude Monte Carlo sampling
7	Refers to istrat = 1 plus istrat = 6
8	Solves the stochastic problem using Monte Carlo pre-sampling
9	Refers to istrat = 1 plus istrat = 8
10	Solves the stochastic problem using control variates
11	Refers to istrat = 1 plus istrat = 10

IWrite *(integer)*: Subproblem output ↩

Default: 0

value	meaning
0	No optimizer output is written
1	Optimizer output is written to the file

NSamples *(integer)*: Sample size used for the estimation ↩

Range: {30, ..., ∞}

Default: 100

NZRows *(integer)*: Number of rows reserved for cuts in the master problem ↩

Range: {1, ..., ∞}

Default: 100

Rho *(real)*: rho parameter of regularization term in the objective function ↩

Default: 1000

TolBen *(real)*: Tolerance for stopping the decomposition algorithm ↩

Default: 1e-7

TolW *(real)*: tolerance when writing debug solution output ↩

Default: 1e-9

5.13.4 Appendix A - GAMS/DECIS Illustrative Examples

5.13.4.1 Example APL1P

```
*   APL1P test model
*   Dr. Gerd Infanger, November 1997

set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1  0.68,  g2  0.64 /;
parameter ccmin(g) min capacity / g1  1000,  g2  1000 /;
parameter ccmax(g) max capacity / g1 10000,  g2 10000 /;
parameter c(g) investment       / g1   4.0,  g2   2.5 /;

table f(g,dl) operating cost
            h    m    l
   g1       4.3  2.0  0.5
   g2       8.7  4.0  1.0;

parameter d(dl) demand                  / h  1040, m  1040, l  1040 /;
parameter us(dl) cost of unserved demand / h    10, m    10, l    10 /;

free variable tcost           total cost;
positive variable x(g)        capacity of generators;
positive variable y(g, dl)    operating level;
positive variable s(dl)       unserved demand;
```

```
equations
cost        total cost
cmin(g)     minimum capacity
cmax(g)     maximum capacity
omax(g)     maximum operating level
demand(dl)  satisfy demand;

cost .. tcost =e=     sum(g, c(g)*x(g))
                    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                    + sum(dl,us(dl)*s(dl));

cmin(g) ..    x(g) =g= ccmin(g);
cmax(g) ..    x(g) =l= ccmax(g);
omax(g) ..    sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g)       = 1;
y.stage(g, dl)   = 2;
s.stage(dl)      = 2;
cmin.stage(g)    = 1;
cmax.stage(g)    = 1;
omax.stage(g)    = 2;
demand.stage(dl) = 2;

* defining independent stochastic parameters
set stoch /out, pro /;

set omega1 / o11, o12, o13, o14 /;
table v1(stoch, omega1)
      o11   o12   o13   o14
out  -1.0  -0.9  -0.5  -0.1
pro   0.2   0.3   0.4   0.1
;

set omega2 / o21, o22, o23, o24, o25 /;
table v2(stoch, omega2)
      o21   o22   o23   o24   o25
out  -1.0  -0.9  -0.7  -0.1  -0.0
pro   0.1   0.2   0.5   0.1   0.1
;


set omega3 / o31, o32, o33, o34 /;
table v3(stoch, omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;


set omega4 / o41, o42, o43, o44 /;
table v4(stoch,omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;

set omega5 / o51, o52, o53, o54 /;
table v5(stoch,omega1)
      o11   o12   o13   o14
out   900  1000  1100  1200
pro  0.15  0.45  0.25  0.15
;

* defining distributions
file stg /MODEL.STG/;
put stg;
put "INDEP DISCRETE" /;
loop(omega1,
put "x g1  omax g1   ", v1("out", omega1), " period2 ", v1("pro", omega1) /;
```

```
);
put "*" /;
loop(omega2,
put "x g2  omax g2   ", v2("out", omega2), " period2 ", v2("pro", omega2) /;
);
put "*" /;
loop(omega3,
put "RHS demand h   ", v3("out", omega3),  " period2 ", v3("pro", omega3) /;
);
put "*" /;
loop(omega4,
put "RHS demand m   ", v4("out", omega4),  " period2 ", v4("pro", omega4) /;
);
put "*" /;
loop(omega5,
put "RHS demand l   ", v5("out", omega5),  " period2 ", v5("pro", omega5) /;
);
putclose stg;


* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;

solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l -  ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
```

5.13.4.2 Example APL1PCA

```
*   APL1PCA test model
*   Dr. Gerd Infanger, November 1997

set g generators /g1, g2/;
set dl demand levels /h, m, l/;

parameter alpha(g) availability / g1  0.68,  g2  0.64 /;
parameter ccmin(g) min capacity / g1  1000,  g2  1000 /;
parameter ccmax(g) max capacity / g1 10000,  g2 10000 /;
parameter c(g) investment       / g1   4.0,  g2   2.5 /;

table f(g,dl) operating cost
            h    m    l
   g1       4.3  2.0  0.5
   g2       8.7  4.0  1.0;

parameter d(dl) demand                   / h  1040, m  1040, l  1040 /;
parameter us(dl) cost of unserved demand / h    10, m    10, l    10 /;

free variable tcost            total cost;
positive variable x(g)         capacity of generators;
positive variable y(g, dl)     operating level;
positive variable s(dl)        unserved demand;

equations
cost        total cost
cmin(g)     minimum capacity
cmax(g)     maximum capacity
omax(g)     maximum operating level
demand(dl)  satisfy demand;

cost .. tcost =e=     sum(g, c(g)*x(g))
                    + sum(g, sum(dl, f(g,dl)*y(g,dl)))
                    + sum(dl,us(dl)*s(dl));

cmin(g) ..    x(g) =g= ccmin(g);
cmax(g) ..    x(g) =l= ccmax(g);
omax(g) ..    sum(dl, y(g,dl)) =l= alpha(g)*x(g);
demand(dl) .. sum(g, y(g,dl)) + s(dl) =g= d(dl);

model apl1p /all/;

* setting decision stages
x.stage(g)       = 1;
y.stage(g, dl)   = 2;
s.stage(dl)      = 2;
cmin.stage(g)    = 1;
cmax.stage(g)    = 1;
omax.stage(g)    = 2;
demand.stage(dl) = 2;


* defining independent stochastic parameters
set stoch /out, pro/;

set omega1 / o11, o12 /;
table v1(stoch,omega1)
       o11  o12
out    2.1  1.0
pro    0.5  0.5 ;

set omega2 / o21, o22 /;
table v2(stoch, omega2)
       o21  o22
out    2.0  1.0
pro    0.2  0.8 ;

parameter hm1(dl) / h  300., m  400., l  200. /;
parameter hm2(dl) / h  100., m  150., l  300. /;

* defining distributions (writing file MODEL.STG)
```

```
file stg / MODEL.STG /;
put stg;

put "BLOCKS DISCRETE" /;
scalar h1;
loop(omega1,
put "BL  v1   period2 ", v1("pro", omega1)/;
loop(dl,
h1 = hm1(dl) * v1("out", omega1);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
loop(omega2,
put " BL v2   period2 ", v2("pro", omega2) /;
loop(dl,
h1 = hm2(dl) * v2("out", omega2);
put "RHS demand ", dl.tl:1, " ", h1/;
);
);
putclose stg;

* setting DECIS as optimizer
* DECISM uses MINOS, DECISC uses CPLEX
option lp=decism;
apl1p.optfile = 1;

solve apl1p using lp minimizing tcost;

scalar ccost capital cost;
scalar ocost operating cost;
ccost = sum(g, c(g) * x.l(g));
ocost = tcost.l -  ccost;
display x.l, tcost.l, ccost, ocost, y.l, s.l;
```

5.13.5 Appendix B - Error Messages

1. ERROR in MODEL.STO: kwd, word1, word2 was not matched in first realization of block
   The specification of the stochastic parameters is incorrect. The stochastic parameter has not been specified in the specification of the first outcome of the block. When specifying the first outcome of a block always include all stochastic parameters corresponding to the block.

2. Option word1 word2 not supported
   You specified an input distribution in the stochastic file that is not supported. Check the DECIS manual for supported distributions.

3. Error in time file
   The time file is not correct. Check the file MODEL.TIM. Check the DECIS manual for the form of the time file.

4. ERROR in MODEL.STO: stochastic RHS for objective, row name2
   The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand side for the objective row (row name2). Check file MODEL.STO.

5. ERROR in MODEL.STO: stochastic RHS in master, row name2
   The specification in the stochastic file is incorrect. You attempted to specify a stochastic right-hand side for the master problem (row name2). Check file MODEL.STO.

6. ERROR in MODEL.STO: col not found, name1
   The specification in the stochastic file is incorrect. The entry in the stochastic file, name1, is not found in the core file. Check file MODEL.STO.

7. ERROR in MODEL.STO: invalid col/row combination, (name1/name2)
   The stochastic file (MODEL.STO) contains an incorrect specification.

8. ERROR in MODEL.STO: no nonzero found (in B or D matrix) for col/row (name1, name2)
   There is no nonzero entry for the combination of name1 (col) and name2(row) in the B-matrix or in the D-matrix. Check the corresponding entry in the stochastic file (MODEL.STO). You may want to include a nonzero coefficient for (col/row) in the core file (MODEL.COR).

9. ERROR in MODEL.STO: col not found, name2
   The column name you specified in the stochastic file (MODEL.STO) does not exist in the core file (MODEL.COR). Check the file MODEL.STO.

10. ERROR in MODEL.STO: stochastic bound in master, col name2
    You specified a stochastic bound on first-stage variable name2. Check file MODEL.STO.

11. ERROR in MODEL.STO: invalid bound type (kwd) for col name2
    The bound type, kwd, you specified is invalid. Check file MODEL.STO.

12. ERROR in MODEL.STO: row not found, name2
    The specification in the stochastic file is incorrect. The row name, name2, does not exist in the core file. Check file MODEL.STO.

13. ERROR: problem infeasible
    The problem solved (master- or subproblem) turned out to be infeasible. If a subproblem is infeasible, you did not specify the problem as having the property of "complete recourse". Complete recourse means that whatever first-stage decision is passed to a subproblem, the subproblem will have a feasible solution. It is the best way to specify a problem, especially if you use a sampling based solution strategy. If DECIS encounters a feasible subproblem, it adds a feasibility cut and continues the execution. If DECIS encounters an infeasible master problem, the problem you specified is infeasible, and DECIS terminates. Check the problem formulation.

14. ERROR: problem unbounded
    The problem solved (master- or subproblem) turned out to be unbounded. Check the problem formulation.

15. ERROR: error code: inform
    The solver returned with an error code from solving the problem (master- or subproblem). Consult the users' manual of the solver (MINOS or CPLEX) for the meaning of the error code, inform. Check the problem formulation.

16. ERROR: while reading SPECS file
    The MINOS specification file (MINOS.SPC) containes an error. Check the specification file. Consult the MINOS user's manual.

17. ERROR: reading mps file, mpsfile
    The core file mpsfile (i.e., MODEL.COR) is incorrect. Consult the DECIS manual for instructions regarding the MPS format.

18. ERROR: row 1 of problem is not a free row
    The first row of the problem is not a free row (i.e., is not the objective row). In order to make the first row a free row, set the row type to be 'N'. Consult the DECIS manual for the MPS specification of the problem.

19. ERROR: name not found = nam1, nam2
    There is an error in the core file (MODEL.COR). The problem cannot be decomposed correctly. Check the core file and check the model formulation.

20. ERROR: matrix not in staircase form
    The constraint matrix of the problem as specified in core file (MODEL.COR) is not in staircase form. The first-stage rows and columns and the second-stage rows and columns are mixed within each other. Check the DECIS manual as to how to specify the core file. Check the core file and change the order of rows and columns.

5.13.6 DECIS License and Warranty

The software, which accompanies this license (the `"Software"`) is the property of Gerd Infanger and is protected by copyright law. While Gerd Infanger continues to own the Software, you will have certain rights to use the Software after your acceptance of this license. Except as may be modified by a license addendum, which accompanies this license, your rights and obligations with respect to the use of this Software are as follows:

- You may
  1. Use one copy of the Software on a single computer,
  2. Make one copy of the Software for archival purposes, or copy the software onto the hard disk of your computer and retain the original for archival purposes,
  3. Use the Software on a network, provided that you have a licensed copy of the Software for each computer that can access the Software over that network, item After a written notice to Gerd Infanger, transfer the Software on a permanent basis to another person or entity, provided that you retain no copies of the Software and the transferee agrees to the terms of this agreement.
- You may not
  1. Copy the documentation, which accompanies the Software,
  2. Sublicense, rent or lease any portion of the Software,
  3. Reverse engineer, de-compile, disassemble, modify, translate, make any attempt to discover the source code of the Software, or create derivative works from the Software.

5.13.6.1 Limited Warranty:

Gerd Infanger warrants that the media on which the Software is distributed will be free from defects for a period of thirty (30) days from the date of delivery of the Software to you. Your sole remedy in the event of a breach of the warranty will be that Gerd Infanger will, at his option, replace any defective media returned to Gerd Infanger within the warranty period or refund the money you paid for the Software. Gerd Infanger does not warrant that the Software will meet your requirements or that operation of the Software will be uninterrupted or that the Software will be error-free.

THE ABOVE WARRANTY IS EXCLUSIVE AND IN LIEU OF ALL OTHER WARRANTIES, WHETHER EXPRESS OR IMPLIED, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.

5.13.6.2 Disclaimer of Damages:

REGARDLESS OF WHETHER ANY REMEDY SET FORTH HEREIN FAILS OF ITS ESSENTIAL PURPOSE, IN NO EVENT WILL GERD INFANGER BE LIABLE TO YOU FOR ANY SPECIAL, CONSEQUENTIAL, INDIRECT OR SIMILAR DAMAGES, INCLUDING ANY LOST PROFITS OR LOST DATA ARISING OUT OF THE USE OR INABILITY TO USE THE SOFTWARE EVEN IF GERD INFANGER HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IN NO CASE SHALL GERD INFANGER'S LIABILITY EXCEED THE PURCHASE PRICE FOR THE SOFTWARE. The disclaimers and limitations set forth above will apply regardless of whether you accept the Software.

5.13.6.3 General:

This Agreement will be governed by the laws of the State of California. This Agreement may only be modified by a license addendum, which accompanies this license or by a written document, which has been signed by both you and Gerd Infanger. Should you have any questions concerning this Agreement, or if you desire to contact Gerd Infanger for any reason, please write:

Gerd Infanger, 1590 Escondido Way, Belmont, CA 94002, USA.

Note

5.14 DICOPT

5.14.1 Introduction

`DICOPT` is a program for solving mixed-integer nonlinear programming (MINLP) problems that involve linear binary or integer variables and linear and nonlinear continuous variables. While the modeling and solution of MINLP optimization problems has not yet reached the stage of maturity and reliability achieved by linear, integer, or non-linear programming modeling, these problems still have rich areas of application. For example, they often arise in engineering design, management sciences, and finance. `DICOPT` (DIscrete and Continuous OPTimizer) was originally developed by Jagadisan Viswanathan, Ignacio E. Grossmann, and Aldo Vecchietti at the Engineering Design Research Center (EDRC) at Carnegie Mellon University. Currently, DICOPT is maintained by GAMS. The program is based on the extensions of the outer-approximation algorithm for the equality relaxation strategy. The MINLP algorithm inside `DICOPT` solves a series of NLP and MIP sub-problems. These sub-problems can be solved using any NLP (Nonlinear Programming) or MIP (Mixed-Integer Programming) solver that runs under GAMS. The performance will heavily depend on the choice of the selected subsolvers.

Although the algorithm has provisions to handle non-convexities, it does not necessarily obtain the global optimum.

The `GAMS/DICOPT` system has been designed with two main goals in mind:

- to build on existing modeling concepts, introduce a minimum of extensions to the existing modeling language, and provide upward compatibility to ensure easy transition from existing modeling applications to nonlinear mixed-integer formulations
- to use existing optimizers to solve the `DICOPT` sub-problems. This allows one to match the best algorithms to the problem at hand and guarantees that any new development and enhancements in the NLP and MIP solvers become automatically and immediate available to `DICOPT`.

5.14.2 Requirements

In order to use `DICOPT` you will need to have access to a licensed GAMS BASE system as well as at least one licensed MIP solver and one licensed NLP solver. For difficult models it is advisable to also have access to multiple solvers.

5.14.3 How to Run a Model with GAMS/DICOPT

DICOPT is only able to solve MINLP and MIQCP models. If you did not specify DICOPT as the default solver you can use the following statement in your GAMS model:

```
option minlp = dicopt;
```

This should appear before the solve statement. DICOPT automatically uses the default MIP and NLP solver to solve its sub-problems. One can override this with GAMS statements like:

```
option nlp = conopt;  { or any other nlp solver }
option mip = gurobi;  { or any other mip solver }
```

These options can also be specified on the command line:

```
> gams mymodel minlp=dicopt nlp=conopt mip=gurobi
```

In the Integrated Development Environment (IDE) the command line option can be specified in the edit line in the right upper corner of the main window.

Possible NLP solvers include conopt, ipopt, knitro, minos, and snopt. Possible MIP solvers include cplex, gurobi, and xpress.

With an option file it is even possible to use alternate solvers in different cycles. Section DICOPT Options explains this is in detail.

5.14.4 Overview of DICOPT

DICOPT solves models of the form:

$$\begin{array}{rl} \text{min or max} & f(x,y) \\ \text{subject to} & g(x,y) \sim b \\ & \ell_x \leq x \leq u_x \\ & y \in \lceil \ell_y \rceil, ..., \lfloor u_y \rfloor \end{array} \qquad \text{(MINLP)}$$

where x are the continuous variables and y are the discrete variables. The symbol $\sim$ is used to denote a vector of relational operators $\{\leq, =, \geq\}$. The constraints can be either linear or non-linear. Bounds ℓ and u on the variables are handled directly. $\lceil x \rceil$ indicates the smallest integer, greater than or equal to x. Similarly, $\lfloor x \rfloor$ indicates the largest integer, less than or equal to x. The discrete variables can be either integer variables or binary variables.

5.14.5 The Algorithm

The algorithm in DICOPT is based on three key ideas:

- Outer Approximation
- Equality Relaxation
- Augmented Penalty

Outer Approximation refers to the fact that the surface described by a convex function lies above the tangent hyper-plane at any interior point of the surface. (In 1-dimension, the analogous geometrical result is that the tangent to a convex function at an interior point lies below the curve). In the algorithm outer-approximations are attained by generating linearizations at each iterations and accumulating them in order to provide successively improved linear approximations of nonlinear convex functions that underestimate the objective function and overestimate the feasible region.

Equality Relaxation is based on the following result from non-linear programming. Suppose the MINLP problem is formulated in the form:

$$\begin{array}{rl} \text{minimize or maximize} & f(x) + c^T y \\ \text{subject to} & G(x) + Hy \sim b \\ & \ell \leq x \leq u \\ & y \in \{0,1\} \end{array}$$

i.e. the discrete variables are binary variables and they appear linearly in the model.

Let $y^{(0)}$ be any fixed binary vector and let $x^{(0)}$ be the solution of the corresponding NLP subproblem:

$$\begin{array}{rl} \text{minimize} & c^T y^{(0)} + f(x) \\ \text{subject to} & Ay^{(0)} + h(x) = 0 \\ & By^{(0)} + g(x) \leq 0 \\ & \ell \leq x \leq u \end{array}$$

Further let

$$\begin{array}{rl} T^{(0)} & = \text{diag}(t_{i,i}) \\ t_{i,i} & = \text{sign}(\lambda_i) \end{array}$$

where λ_i is the Lagrange multiplier of the i-th equality constraint.

If f is pseudo-convex, h is quasi-convex, and g is quasi-convex, then x^0 is also the solution of the following NLP:

$$\begin{array}{rl} \text{minimize} & c^T y^{(0)} + f(x) \\ \text{subject to} & T^{(0)}(Ay^{(0)} + h(x)) \leq 0 \\ & By^{(0)} + g(x) \leq 0 \\ & \ell \leq x \leq u \end{array}$$

In colloquial terms, under certain assumptions concerning the convexity of the nonlinear functions, an equality constraint can be `relaxed` to be an inequality constraint. This property is used in the MIP master problem to accumulate linear approximations.

Augmented Penalty refers to the introduction of (non-negative) slack variables on the right hand sides of the just described inequality constraints and the modification of the objective function when assumptions concerning convexity do not hold.

The algorithm underlying `DICOPT` starts by solving the NLP in which the 0-1 conditions on the binary variables are relaxed. If the solution to this problem yields an integer solution the search stops. Otherwise, it continues with an alternating sequence of nonlinear programs (NLP) called subproblems and mixed-integer linear programs (MIP) called master problems. The NLP subproblems are solved for fixed 0-1 variables that are predicted by the MIP master problem at each (major) iteration. For convex problems the master problem also provides a lower bound on the objective function. This lower bound (in the case of minimization) increases monotonically as iterations proceed due to the accumulation of linear approximations. Note that in the case of maximization this bound is an upper bound which can be used as a stopping criterion through a DICOPT option `stop 1` (see section DICOPT Options). Another stopping criterion that tends to work very well for non-convex problems (and even for convex problems) is based on the heuristic: stop as soon as the NLP subproblems start worsening (i.e. the current NLP subproblem has an optimal objective function that is worse than the previous NLP subproblem). This stopping criterion relies on the use of the augmented penalty, and is used in the description of the algorithm below. This is also the default stopping criterion in the implementation of DICOPT. The algorithm can be stated briefly as follows:

1. Solve the NLP relaxation of the MINLP program. If $y^{(0)} = y$ is integer, stop(`integer optimum found`). Else continue with step 2.
2. Find an integer point $y^{(1)}$ with an MIP master problem that features an augmented penalty function to find the minimum over the convex hull determined by the half-spaces at the solution $(x^{(0)}, y^{(0)})$.
3. Fix the binary variables $y = y^{(1)}$ and solve the resulting NLP. Let $(x^{(1)}, y^{(1)})$ be the corresponding solution.
4. Find an integer solution $y^{(2)}$ with a MIP master problem that corresponds to the minimization over the intersection of the convex hulls described by the half-spaces of the KKT points at $y^{(0)}$ and $y^{(1)}$.
5. Repeat steps 3 and 4 until there is an increase in the value of the NLP objective function. (Repeating step 4 means augmenting the set over which the minimization is performed with additional linearizations - i.e. half-spaces - at the new KKT point).

In the MIP problems integer cuts are added to the model to exclude previously determined integer vectors $y^{(1)}, y^{(2)}, ..., y^{(K)}$.

For a detailed description of the theory and references to earlier work, see [193] [110] [52] .

The algorithm has been extended to handle general integer variables and integer variables appearing nonlinearly in the model.

5.14.6 Modeling

5.14.6.1 Relaxed Model

Before solving a model with DICOPT, the user is strongly advised to experiment with the relaxed model where the integer restrictions are ignored. This is the `RMINLP` model. As the DICOPT will start solving the relaxed problem and can use an existing relaxed optimal solution, it is a good idea to solve the `RMINLP` always before attempting to solve the `MINLP` model, i.e., the following fragment is not detrimental with respect to performance:

```
  model m /all/;
  option nlp=conopt;
  option mip=cplex;
  option rminlp=conopt;
  option minlp=dicopt;
*
* solve relaxed model
*
  solve m using rminlp minimizing z;
  abort$(m.modelstat > 2.5) "Relaxed model could not be solved";

*
* solve minlp model
*
  solve m using minlp minimizing z;
```

The second `SOLVE` statement will only be executed if the first `SOLVE` was successful, i.e., if the model status was one (optimal) or two (locally optimal).

In general it is not a good idea to try to solve an MINLP model if the relaxed model cannot be solved reliably. As the RMINLP model is a normal NLP model, some obvious points of attention are:

- Scaling. If a model is poorly scaled an NLP solver may not be able find the optimal or even a feasible solution. Some NLP solvers have automatic scaling algorithms, but often it is better to attack this problem on the modeling level. The GAMS scaling facility can help in this respect.
- Starting point. If a poor starting point is used, the NLP solver may not be able to find a feasible or optimal solution. A starting point can be set by setting level values, e.g. `X.L = 1;`. The GAMS default levels are zero, with is often not a good choice.
- Adding bounds. Add bounds so that all functions can be properly evaluated. If you have a function $\sqrt{x}$ or $\log(x)$ in the model, you may want to add a bound `X.LO=0.001;`. If a function like $\log(f(x))$ is used, you may want to introduce an auxiliary variable and equation $y = f(x)$ with an appropriate bound `Y.LO=0.001;`.

In some cases the relaxed problem is the most difficult model. If more than one NLP solver is available you may want to try them in a sequence:

```
  model m /all/;
  option nlp=conopt;
  option mip=cplex;
  option rminlp=conopt;
  option minlp=dicopt;
*
* solve relaxed model
*
  solve m using rminlp minimizing z;
  if (m.modelstat > 2.5,
     option rminlp=minos;
     solve m using rminlp minimizing z;
  );
  if (m.modelstat > 2.5,
     option rminlp=snopt;
     solve m using rminlp minimizing z;
  );

*
* solve minlp model
*
  solve m using minlp minimizing z;
```

In this fragment we first try to solve the relaxed model using CONOPT. If that fails we try MINOS, and if that solve also fails we try SNOPT.

It is worthwhile to spend some time getting the relaxed model to solve reliably and speedily. In most cases, modeling improvements in the relaxed model, such as scaling, will also benefit the subsequent NLP sub-problems. In general these modeling improvements turn out to be rather solver independent: changes that improve the performance with CONOPT will also help solving the model with MINOS.

5.14.6.2 OPTCR and OPTCA

The DICOPT algorithm assumes that the integer sub-problems are solved to optimality. The GAMS options for OPTCR and OPTCA are therefore ignored: subproblems are solved with both tolerances set to zero. If you want to solve a MIP sub-problem with an optimality tolerance you can use the DICOPT option file to set OPTCR or OPTCA. For more information see section DICOPT Options.

For models with many discrete variables, it may be necessary to introduce an OPTCR or OPTCA option in order to solve the model in acceptable time. For models with a limited number of integer variables the default to solve MIP sub-models to optimality may be acceptable.

5.14.6.3 Integer Formulations

A number of MIP formulations are not very obvious and pose a demand on the modeler's knowledge and experience. A good overview of integer programming modeling is given in [202] .

Many integer formulations use a so-called big- M construct. It is important to choose small values for those big- M numbers. As an example consider the fixed charge problem where $y_i \in \{0,1\}$ indicate if facility i is open or closed, and where x_i is the production at facility i. Then the cost function can be modeled as:

$$\begin{aligned} C_i &= f_i y_i + v_i x_i \\ x_i &\leq M_i y_i \\ & y_i \in \{0,1\} \\ & 0 \leq x_i \leq cap_i \end{aligned}$$

where f_i is the fixed cost and v_i the variables cost of operating a facility i. In this case the chosen M_i should be large enough that x_i is not restricted if $y_i = 1$. On the other hand, it should be as small as possible. This leads to a choice to have M_i equal to the (tight) upperbound of variable x_i (i.e. the capacity cap_i of facility i).

5.14.6.4 Non-smooth Functions

GAMS alerts NLP modelers against the use of non-smooth functions such as min(), max(), smin(), smax() and abs(). In order to use these functions, a non-linear program needs to be declared as a DNLP model instead of a regular NLP model:

```
option dnlp=conopt;
model m /all/;
solve m minimizing z using dnlp;
```

This construct will warn the user that problems may arise due to the use of non-smooth functions.

A possible solution is to use a smooth approximation. For instance, the function $f(x) = |x|$ can be approximated by $g(x) = \sqrt{x^2 + \varepsilon}$ for some $\varepsilon > 0$. This approximation does not contain the point $(0, 0)$. An alternative approximation can be devised that has this property:

$$f(x) \approx \frac{2x}{1 + e^{-x/h}} - x$$

MINLP models do not have such protection against non-smooth functions, but the use of such functions is just as problematic here. It is possible to use discrete variables with MINLP models in order to model if-then-else situations. For instance, in the case of the absolute value we can replace x by $x^+ - x^-$ and $|x|$ by $x^+ + x^-$ by using:

$$\begin{aligned} x &= x^+ - x^- \\ |x| &= x^+ + x^- \\ x^+ &\leq \delta M \\ x^- &\leq (1 - \delta) M \\ x^+, x^- &\geq 0 \\ \delta &\in \{0, 1\} \end{aligned}$$

where δ is a binary variable.

5.14.7 GAMS Options

GAMS options are specified in the GAMS model source, using either the `option` statement or a model suffix.

5.14.7.1 The OPTION Statement

An option statement sets a global parameter. An option statement should appear *before* the `solve` statement, as in:

```
model m /all/;
option iterlim=100;
solve m using minlp minimizing z;
```

Option statements that affect the behavior of `DICOPT` are listed below:

- **option domlim** = n;
  This option sets a limit on the total accumulated number of non-linear function evaluation errors that are allowed while solving the NLP subproblems or inside `DICOPT` itself. An example of a function evaluation error or domain error is taking the square root of a negative number. This situations can be prevented by adding proper bounds. The default is zero, i.e. no function evaluation errors are allowed.

  In case a domain error occurs, the listing file will contain an appropriate message, including the equation that is causing the problem. For instance:

```
**** ERRORS(S) IN EQUATION loss(cc,sw)
     2 instance(s) of - UNDEFINED REAL POWER (RETURNED  0.0E+00)
```

If such errors appear you can increase the DOMLIM limit, but often it is better to prevent the errors from occurring in the first place. In many cases this can be accomplished by adding appropriate bounds. Sometimes extra variables and equations need to be added to accomplish this. For instance, with an expression like $\log(x-y)$, you may want to introduce a variable $z > \varepsilon$ and an equation $z = x - y$, so that the expression can be rewritten as $\log(z)$.

- **option iterlim =** n**;**
  This option sets a limit on the total accumulated (minor) iterations performed in the MIP and NLP subproblems.
- **option minlp =** *dicopt***;**
  This option selects DICOPT to solve MINLP problems.
- **option mip =** s**;**
  This option sets the MIP solver to be used for the MIP master problems. Note that changing from one MIP solver to another can lead to different results, and may cause DICOPT to follow a different path.
- **option nlp =** s**;**
  This option sets the NLP solver to be used for the NLP sub-problems. Note that changing from one NLP solver to another can lead to different results, and may cause DICOPT to follow a different path.
- **option optca =** x**;**
  This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.
- **option optcr =** x**;**
  This option is ignored. MIP master problems are solved to optimality unless specified differently in the DICOPT option file.
- **option reslim =** x**;**
  This option sets a limit on the total accumulated time (in seconds) spent inside DICOPT and the subsolvers.
- **option sysout = on;**
  This option will print extra information to the listing file.

In the list above (and in the following) n indicates an integer number. GAMS will also accept fractional values, which will be rounded. Options marked with an x parameter expect a real number. Options with an s parameter expect a string argument.

5.14.7.2 The Model Suffix

Some options are set by assigning a value to a model suffix, as in:

```
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

Model suffixes that affect the behavior of DICOPT are listed below:

- m**.dictfile = 1;**
  This option tells GAMS to write a dictionary file containing information about GAMS identifiers (equation and variables names). Such information is needed when the DICOPT option nlptracelevel is used, otherwise the option can be ignored.

- ***m*.iterlim = *n*;**
  Sets the total accumulated (minor) iteration limit. This option overrides the global iteration limit set by an option statement, e.g.,

  ```
  model m /all/;
  m.iterlim = 100;
  option iterlim = 1000;
  solve m using minlp minimizing z;
  ```

  will cause DICOPT to use an iteration limit of 100.

- ***m*.optfile = 1;**
  This option instructs DICOPT to read an option file dicopt.opt. This file should be located in the current directory (or the project directory when using the GAMS IDE). The contents of the option file will be echoed to the listing file and to the screen (the log file):

  ```
  --- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
  > maxcycles 10
  --- DICOPT: Starting major iteration 1
  ```

  If the option file does not exist, the algorithm will proceed using its default settings. An appropriate message will be displayed in the listing file and in the log file:

  ```
  --- DICOPT: Reading option file D:\MODELS\SUPPORT\DICOPT.OPT
  --- DICOPT: File does not exist, using defaults...
  --- DICOPT: Starting major iteration 1
  ```

- ***m*.optfile = *n*;**
  If $n > 1$ then the option file that is read is called dicopt.op n (for $n = 2, ..., 9$) or dicopt.o n (for $n = 10, ..., 99$). E.g. m.optfile=2; will cause DICOPT to read dicop.op2.

- ***m*.prioropt = 1;**
  This option turns on the use of priorities on the discrete variables. Priorities influence the branching order chosen by the MIP solver during solution of the MIP master problems. The use of priorities can greatly impact MIP solver performance. The priorities themselves have to be specified using the .prior variables suffix, e.g. x.prior(i,j) = ord(i);. Contrary to intuition, variables with a lower value for their priority are branched on before variables with a higher priority, i.e., the most important variables should get lower priority values.

- ***m*.reslim = *x*;**
  Sets the total accumulated time limit. This option overrides the global time limit set by an option statement.

5.14.8 DICOPT Options

This sections describes the options that can be specified in the DICOPT option file. This file is usually called dicopt.opt. The optfile model suffix must be set to tell DICOPT to read this file:

```
model m /all/;
m.optfile=1;
solve m using minlp minimizing z;
```

The option file is searched for in the current directory, or in the project directory when the IDE is used.

The option file is a standard text file, with a single option on each line. All options are case-insensitive. A line is a comment line if it starts with an asterisk, $*$ in column one. A valid option file can look like:

```
* stop only on infeasible MIP or hitting a limit
stop 0
* use minos to solve first NLP sub problem
* and conopt for all subsequent ones
nlpsolver minos conopt
```

A convenient way to write the option file from within a GAMS model is to use the following construct:

```
$onecho > dicopt.opt
stop 0
nlpsolver minos conopt
$offecho
```

This will make the model self-contained. Notice, however, that this overwrites an existing file `dicopt.opt`.

Available `DICOPT` options are listed below:

5.14.8.1 Algorithmic options

Option	Description	Default
continue	How to proceed in case of NLP errors	2
convex	If enabled, the defaults for a number of other options are set to values appropriate to convex MINLPs	0
dumpsubprob	Whether to dump MIP and NLP subproblems into GAMS files	0
infbnd	Bound to use for unbounded integer variables in integer cuts	10000
infeasder	Use derivatives of infeasible nonlinear subproblems	0
maxcycles	Maximum number of cycles	20
relaxed	How to start DICOPT	1
solvelink	Solvelink for NLP and MIP subsolver	5
stop	Stopping criterion	2
usexinit	Use the user initial point as starting point for all NLP solves	0
weight	Penalty parameter, set above 1e20 to disable penalty relaxation	1000

5.14.8.2 Tolerances

Option	Description	Default
epsmip	Tolerance on test on monotonic improvement of MIP master problem	1.0e-6
epsx	Tolerance for integer values when loading relaxed solution	1.0e-3

5.14.8.3 MIP masterproblem options

Option	Description	Default
mipiterlim	List of iteration limits	
mipoptfile	List of option files for MIP solver	
mipreslim	List of resource limits	
mipsolver	List of MIP solvers	
optca	List of OPTCA values	
optcr	List of OPTCR values	

5.14.8.4 NLP subproblem options

Option	Description	Default
domlim	List of allowed number of domain errors	
nlpiterlim	List of iteration limits	
nlpoptfile	List of option files for NLP solver	
nlpreslim	List of resource limits	
nlpsolver	List of NLP solvers	
nlptracefile	Base name of trace files	`nlptrace`
nlptracelevel	Trace level	0

5.14.8.5 Feasibility Pump

Option	Description	Default
feaspump	Whether to run the Feasibility Pump	0
fp_acttol	Tolerance on when to consider an equation as active	1E-6
fp_cutoffdecr	Additional relative decrement of cutoff value for the original objective function	0.1
fp_dumpsubprob	Whether to dump subproblems (NLPs and MIPs) into GAMS files	0
fp_integercuts	Whether to add integer cuts after an NLP subproblem has been solved to optimality	1
fp_iterlimit	Major Iteration limit	20
fp_mipgap	Optimality tolerance (relative gap) to use for solving MIP projection problem	0.01
fp_projcuts	Whether to add cut derived from projection of MIP solution onto NLP feasible set	1
fp_projzerotol	Tolerance on when to consider optimal value of projection problem as zero, which may trigger the solution of a Sub-NLP	1E-4
fp_sollimit	Stop when a number of (improving) solutions has been bound	maxint
fp_stalllimit	Stop when no improving solution has been bound for a number of FP iterations, -1 to disable	5

Option	Description	Default
fp_subsolverlog	Whether to show the log of subsolvers	0
fp_timelimit	Time limit	maxdouble
fp_transfercuts	Whether to transfer cuts from the Feasibility Pump MIP to the DICOPT MIP (all except from the round in which the FP MIP became infeasible)	1

5.14.8.6 Detailed Options Description

continue *(integer)*: How to proceed in case of NLP errors ↩

This option can be used to let DICOPT continue in cases of NLP solver failure. The preferred approach is to fix the model so that NLP subproblems solve without problems. In some cases, however, (partial) failures of an NLP solver in solving the NLP subproblems can be ignored, as DICOPT may recover later on. Adding the option `continue 0` during model debugging enables DICOPT to function in a more specific way.

Default: 2

value	meaning
0	Stop on solver failure Stop on solver failure. DICOPT will terminate when an NLP subproblem can not be solved to optimality. Some NLP solvers terminate with a status other than optimal if not all of the termination criteria are met. For instance, the change in the objective function is negligible (indicating convergence) but the reduced gradients are not within the required tolerance. Such a solution may or may not be close to the (local) optimum. Using `continue 0` will prevent DICOPT from accepting such a solution.
1	Accept non-optimal feasible solutions NLP subproblem failures resulting in a non-optimal but feasible solutions are accepted. Sometimes an NLP solver cannot make further progress towards meeting all optimality conditions, although the current solution is feasible. Such a solution can be accepted by this option.
2	Ignore infeasible solutions NLP subproblem failures resulting in a non-optimal but feasible solution are accepted (as in option `continue 1`). NLP subproblem failures resulting in an infeasible solution are ignored. The corresponding configuration of discrete variables is forbidden to be used again. An integer cut to accomplish this is added to subsequent MIP master problems. Note that the relaxed NLP solution should be feasible. This setting is the default.

convex *(boolean)*: If enabled, the defaults for a number of other options are set to values appropriate to convex MINLPs ↩

If this option is enable, the default option values will be changed such that DICOPT will stop on crossover (stop set to 1), linearizations from infeasible NLP subproblems will be added to the MIP master problem (infeasder set to 1), penalty relaxation is no longer applied to linearizations (weight set to maxdouble), and the feasibility pump is run if there are no semicontinuous or semiinteger variables and no special ordered sets (feaspump set to 1).

Default: 0

domlim *(string)*: List of allowed number of domain errors ↩

domlim i_1 i_2 ... i_n. Sets a limit of the number of function and derivative evaluation errors for a particular cycle. A number of -1 means that the global GAMS option `domlim` is used. The last number i_n sets a domain error limit for all cycles $n, n+1, \ldots$.

Example: `domlim 0 100 0`
The NLP solver in the second cycle is allowed to make up to 100 evaluation errors, while all other cycles must be solved without evaluation errors.

The default is to use the global GAMS `domlim` option.

dumpsubprob *(boolean)*: Whether to dump MIP and NLP subproblems into GAMS files ↩

For subproblems of the feasibility pump, use option fp_dumpsubprob.

Default: `0`

epsmip *(real)*: Tolerance on test on monotonic improvement of MIP master problem ↩

This option can be used to relax the test on MIP objective functions. The objective function values of the MIP master problems should form a monotonic worsening curve. This is not the case if the MIP master problems are not solved to optimality. If the options `OPTCR` or `OPTCA` are set to a nonzero value, this test is bypassed. If the test fails, `DICOPT` will fail with a message:

```
The MIP solution became better after adding integer cuts.  Something
is wrong.  Please check if your model is properly scaled.  Also check
your big M formulations -- the value of M should be relatively small.
This error can also occur if you used a MIP solver option file with
a nonzero OPTCR or OPTCA setting.  In that case you may want to increase
the EPSMIP setting using a DICOPT option file.
```

The value of

$$\frac{\text{PreviousObj} - \text{CurrentObj}}{1 + |\text{PreviousObj}|}$$

is compared against `epsmip`. In case the test fails but you want `DICOPT` to continue anyway, you may want to increase the value of `epsmip`. The current values used in the test (previous and current MIP objective, `epsmip`) are printed along with the above message to provide information about how much you should increase `epsmip` to pass the test. Normally, you should not have to change this value.

Default: `1.0e-6`

epsx *(real)*: Tolerance for integer values when loading relaxed solution ↩

This tolerance is used to distinguish integer variables that are set to an integer value by the user, or integer variables that are fractional. See the option relaxed.

Default: `1.0e-3`

feaspump *(integer)*: Whether to run the Feasibility Pump ↩

Default: `0`

value	meaning
0	Do not run the Feasibility Pump
1	Run the Feasibility Pump if there are no semicontinuous or semiinteger variables and no special ordered sets
2	Always run the Feasibility Pump

fp_acttol *(real)*: Tolerance on when to consider an equation as active ↩

Default: `1E-6`

fp_cutoffdecr *(real)*: Additional relative decrement of cutoff value for the original objective function ↩

Default: `0.1`

fp_dumpsubprob *(boolean)*: Whether to dump subproblems (NLPs and MIPs) into GAMS files ↩

Default: `0`

fp_integercuts *(integer)*: Whether to add integer cuts after an NLP subproblem has been solved to optimality ↩

Default: `1`

value	meaning
0	Do not use integer cuts
1	Use integer cuts only for mixed-binary problems
2	Always use integer cuts

fp_iterlimit *(integer)*: Major Iteration limit ↩

Default: `20`

fp_mipgap *(real)*: Optimality tolerance (relative gap) to use for solving MIP projection problem ↩

Default: `0.01`

fp_projcuts *(boolean)*: Whether to add cut derived from projection of MIP solution onto NLP feasible set ↩

Default: `1`

fp_projzerotol *(real)*: Tolerance on when to consider optimal value of projection problem as zero, which may trigger the solution of a Sub-NLP ↩

Default: `1E-4`

fp_sollimit *(integer)*: Stop when a number of (improving) solutions has been bound ↩

Default: `maxint`

fp_stalllimit *(integer)*: Stop when no improving solution has been bound for a number of FP iterations, -1 to disable ↩

Range: {`-1`, ..., ∞}

Default: `5`

fp_subsolverlog *(boolean)*: Whether to show the log of subsolvers ↩

Default: `0`

fp_timelimit *(real)*: Time limit ↩

Default: `maxdouble`

fp_transfercuts *(boolean)*: Whether to transfer cuts from the Feasibility Pump MIP to the DICOPT MIP (all except from the round in which the FP MIP became infeasible) ↩

Default: `1`

infbnd *(real)*: Bound to use for unbounded integer variables in integer cuts ↩

Value to use for missing bounds on discrete variables when constructing integer cuts.

Default: `10000`

infeasder *(integer)*: Use derivatives of infeasible nonlinear subproblems ↩

This option is to determine whether linearizations of infeasible NLP subproblems are added or not added to the MIP master problem.

Default: `0`

value	meaning
`0`	No linearizations of infeasible NLP subproblems This is the default option in which no linearizations are added in the infeasible NLP subproblems. In this case a simple integer cut is added to remove from consideration the 0-1 vector that gave rise to the infeasible NLP. Since this may slow the convergence, it is recommended to reformulate the MINLP with "elastic" constraints (i.e., adding slacks to infeasible constraints and adding a penalty for them in the objective) to ensure that the NLP subproblems are mathematically feasible.
`1`	Add linearization for infeasible NLP subproblems This will add linearizations derived from the infeasible NLP subproblem to the master problem. This option is recommended to speed up convergence when the MINLP is known to be convex (i.e. its continuous relaxation is convex). The possibility of cutting-off the global optimum is increased if it is used for a nonconvex MINLP.

maxcycles *(integer)*: Maximum number of cycles ↩

The maximum number of cycles or major iterations performed by DICOPT.

Default: `20`

mipiterlim *(string)*: List of iteration limits ↩

mipiterlim i_1 i_2 ... i_n sets an iteration limit on individual MIP master problems. The last number i_n is valid for all subsequent cycles $n, n+1, \ldots$. A number of -1 indicates that there is no (individual) limit on the corresponding MIP master problem. A global iteration limit is maintained through the GAMS option iterlim.

Example: `mipiterlim 10000 -1`
The first MIP master problem cannot use more than 10000 iterations, while subsequent MIP master problems are not individually restricted.

Example: `mipiterlim 10000`
Sets an iteration limit of 10000 on all MIP master problems.

When this option is used it is advised to have the option `continue` set to its default of 2. The default for this option is not to restrict iteration counts on individual solves of MIP master problems. The default for this option is not to restrict iteration counts on individual solves of MIP master problems.

mipoptfile *(string)*: List of option files for MIP solver ↩

mipoptfile s_1 s_2 ... s_n specifies the option file to be used for the MIP master problems. Several option files can be specified, separated by a blank. If a digit `1` is entered the default option file for the MIP solver in question is being used. The digit `0` indicates that no option file is to be used. The last option file is also used for subsequent MIP master problems.

Example: `mipoptfile mip.opt mip2.opt 0`
This option will cause the first MIP master problem solver to read the option file `mip.opt` and the second one to read the option file `mip2.opt`; subsequent MIP master problem solvers will not use any option file.

Example: `mipoptfile 1`
This will cause the MIP solver for all MIP subproblems to read a default option file (e.g. `cplex.opt`, `xpress.opt`, `gurobi.opt` etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

mipreslim *(string)*: List of resource limits ↩

mipreslim x_1 x_2 ... x_n sets a resource (time) limit on individual MIP master problems. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$. A number -1.0 means that the corresponding MIP master problem is not individually time restricted. A global time limit is maintained through the GAMS option reslim.

Example: `mipreslim -1 10000 -1`
The MIP master problem in cycle 2 cannot use more than 100 seconds, while subsequent MIP master problems are not individually restricted.

Example: `mipreslim 1000`
Sets a time limit on all MIP master problems of 1000 seconds.

When this option is used it is advised to have the option `continue` set to its default of 2. The default for this option is not to restrict individually the time a solver can spent on the MIP master problem.

mipsolver *(string)*: List of MIP solvers ↩

This option specifies with MIP solver to use for the MIP master problems.

Example: `mipsolver cplex xpress`
This instructs `DICOPT` to use Cplex for the first MIP and XPRESS for the second and subsequent MIP problems. The last entry may be used for more than one problem.

The names to be used for the solvers are the same as one uses in the GAMS statement `OPTION MIP=...;`. The default is to use the default MIP solver.
Note that changing from one MIP solver to another can lead to different results, and may cause `DICOPT` to follow a different path.

nlpiterlim *(string)*: List of iteration limits ↩

nlpiterlim i_1 i_2 ... i_n sets an iteration limit on individual NLP subproblems. The last number i_n is valid for all subsequent cycles $n, n+1, \ldots$. A number of -1 indicates that there is no (individual) limit on the corresponding NLP subproblem. A global iteration limit is maintained through the GAMS option reslim.

Example: `nlpiterlim 1000 -1`
The first (relaxed) NLP subproblem cannot use more than 1000 iterations, while subsequent NLP subproblems are not individually restricted.

Example: `nlpiterlim 1000`
Sets an iteration limit of 1000 on all NLP subproblems.

When this option is used it is advised to have the option continue set to its default of 2. This default does not restrict the amount of iterations an NLP solver can spend on an NLP subproblem, other than the global iteration limit.

nlpoptfile *(string)*: List of option files for NLP solver ↩

nlpoptfile s_1 s_2 ... s_n specifies the option file to be used for the NLP subproblems. Several option files can be specified, separated by a blank. If a digit `1` is entered, the default option file for the NLP solver in question is being used. The digit `0` indicates that no option file is to be used. The last option file is also used for subsequent NLP subproblems.

Example: `nlpoptfile nlp.opt nlp2.opt 0`
This option will cause the first NLP subproblem solver to read the option file `nlp.opt` and the second one to read the option file `nlp2.opt`; subsequent NLP subproblem solvers will not use any option file.

Example: `nlpoptfile 1`
This will cause the NLP solver for all NLP subproblems to read a default option file (e.g. `conopt.opt`, `minos.opt`, `snopt.opt` etc.).

Option files are located in the current directory (or the project directory when using the IDE). The default is not to use an option file.

nlpreslim *(string)*: List of resource limits ↩

nlpreslim x_1 x_2 ... x_n sets a resource (time) limit on individual NLP subproblems. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$. A number -1.0 means that the corresponding NLP subproblem is not individually time restricted. A global time limit is maintained through the GAMS option reslim.

Example: `nlpreslim 100 -1`
The first (relaxed) NLP subproblem can not use more than 100 seconds, while subsequent NLP subproblems are not individually restricted.

Example: `nlpreslim 1000`
Sets a time limit of 1000 seconds on all NLP subproblems.

When this option is used it is advised to have the option `continue` set to its default of 2. This default does not restrict the time an NLP solver can spend on an NLP subproblem (other than the global resource limit).

nlpsolver *(string)*: List of NLP solvers ↩

nlpsolver s_1 s_2 ... s_n. This option specifies which NLP solver to use for the NLP subproblems.

Example: `nlpsolver conopt minos snopt`
tells `DICOPT` to use `CONOPT` for the relaxed NLP, MINOS for the second NLP subproblem, and `SNOPT` for the third and subsequent ones. The last entry is used for more than one subproblem: for all subsequent ones `DICOPT` will use the last specified solver.

The names to be used for the solvers are the same as those used in the GAMS statement`OPTION NLP=...;`. The default is to use the default NLP solver. Note that changing from one NLP solver to another can lead to different results, and may cause `DICOPT` to follow a different path.

nlptracefile *(string)*: Base name of trace files ↩

Name of the files written if the option `nlptracelevel` is set. Only the stem is needed: if the name is specified as `nlptracefile nlptrace`, then files of the form `nlptrace.001`, `nlptrace.002`, etc. are written. These files contain the settings of the integer variables so that NLP subproblems can be investigated independently of `DICOPT`.

Default: `nlptrace`

nlptracelevel *(integer)*: Trace level ↩

This sets the level for NLP tracing, which writes a file for each NLP sub-problem, so that NLP sub-problems can be investigated outside the `DICOPT` environment. See also the option DICOPTnlptracefile "nlptracefile".

By including a trace file in your original problem and changing it into an MINLP problem, the subproblem will be solved directly by an NLP solver. This option only works if the names in the model (names of variables and equations) are exported by GAMS. This can be accomplished by using the `m.dictfile` model suffix, as in `m.dictfile=1;`. In general it is more convenient to use the `CONVERT` solver to generate isolated NLP models (see section Model Debugging).

Default: `0`

value	meaning
`0`	No trace info is written No trace files are written. This is the default.
`1`	GAMS file with fixed integer variables A GAMS file for each NLP subproblem is written which fixes the discrete variables.
`2`	Include levels of continuous variables As `nlptracelevel 1`, but in addition level values of the continuous variables are written.
`3`	Include all levels and marginals As `nlptracelevel 2`, but in addition marginal values for the equations and variables are written.

optca *(string)*: List of OPTCA values ↩

optca x_1 x_2 ... x_n. The absolute optimality criterion for the MIP master problems. The GAMS option optca is ignored, as, by default, `DICOPT` wants to solve MIP master problems to optimality. It is possible to stop the MIP solver earlier to allow it to solve a large problem, by specifying a value for `optca` or `optcr` in a `DICOPT` option file. With setting a value for `optca`, the MIP solver is instructed to stop as soon as the gap between the best possible integer solution and the best found integer solution is less than x, i.e. stop as soon as

$$|\text{BestFound} - \text{BestPossible}| \leq x$$

It is possible to specify a different `optca` value for each cycle. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$.

Example: `optca 10`
Stop the search in all MIP problems as soon as the absolute gap is less than 10.

Example: `optca 0 10 0`
Sets a nonzero `optca` value of 10 for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

optcr *(string)*: List of OPTCR values ↩

optcr x_1 x_2 ... x_n. The relative optimality criterion for the MIP master problems. The GAMS option optcr is ignored, as by default `DICOPT` wants to solve MIP master problems to optimality. To allow it to solve a large problem it is possible to stop the MIP solver earlier by specifying a value for `optca` or `optcr` in a `DICOPT` option file. With setting a value for `optcr`, the MIP solver is instructed to stop as soon as the relative gap between the best possible integer solution and the best found integer solution is less than x, i.e., stop as soon as

$$\frac{|\text{BestFound} - \text{BestPossible}|}{|\text{BestPossible}|} \leq x$$

Note that the relative gap cannot be evaluated if the best possible integer solution is zero. In these cases the absolute optimality criterion `optca` can be used. It is possible to specify a different `optcr` value for each cycle. The last number x_n is valid for all subsequent cycles $n, n+1, \ldots$.

Example: `optcr 0.1`
Stop the search in all the MIP problems as soon as the relative gap is smaller than 10%.

Example: `optcr 0 0.01 0`
Sets a nonzero `optcr` value of 1% for cycle 2, while all other MIP master problems are solved to optimality.

The default is zero.

relaxed *(integer)*: How to start DICOPT ↩

In some cases it may be possible to use a known configuration of the discrete variables. Some users have very difficult problems, where the relaxed problem cannot be solved but where NLP sub-problems with the integer variables fixed are much easier. If a reasonable integer configuration is known in advance in theses cases we can bypass the relaxed NLP and tell `DICOPT` to directly start with this integer configuration. The integer variables need to be specified by the user before the solve statement by assigning values to the levels, as in `Y.L(I) = INITVAL(I);`.

Default: `1`

value	meaning
`0`	Start with all integers fixed to the starting value The first NLP sub-problem will be executed with all integer variables fixed to the values specified by the user. If you don't assign a value to an integer variable,it will retain it's current value, which is zero by default
`1`	Start with relaxed NLP The first NLP problem is the relaxed NLP problem: all integer variables are relaxed between their bounds. This is the default.
`2`	Start with mixture of fixed and relaxed integers The first NLP subproblem will be executed with some variables fixed and some relaxed. The program distinguishes the fixed from the relaxed variables by

solvelink *(integer)*: Solvelink for NLP and MIP subsolver ↩

This option defines the solvelink used for the NLP and MIP subsolver.

Default: `5`

value	meaning
`1`	Call NLP and MIP solver via script
`2`	Call NLP and MIP solver via module
`5`	Call NLP and MIP solver in memory

stop *(integer)*: Stopping criterion ↩

This option defines the stopping criterion to be used. The search is always stopped when the (minor) iteration limit (the `iterlim` option), the resource limit (the `reslim` option), or the major iteration limit (see maxcycles) is hit or when the MIP master problem becomes infeasible.
Note: In general a higher number stops earlier, although in some cases stopping rule 2 may terminate the search earlier than rule 1. Section Modeling shows some experiments with these stopping criteria.

Default: `2`

value	meaning
`0`	Stop on maxcycles Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that `DICOPT` does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general `DICOPT` will often find the optimal solution using one of the more optimistic stopping rules. Do not stop unless an iteration limit, resource limit, or major iteration limit is hit or an infeasible MIP master problem becomes infeasible. This option can be used to verify that DICOPT does not stop too early when using one of the other stopping rules. In general it should not be used on production runs, as in general DICOPT will find often the optimal solution using one of the more optimistic stopping rules.
`1`	Stop on crossover Stop as soon as the bound defined by the objective of the last MIP master problem is worse or close (w.r.t. GAMS option optcr) to the best NLP solution found (a `crossover` occurred). For convex problems this gives a global solution, provided the weights are large enough and optcr is set to 0. This stopping criterion should only be used if it is known or it is very likely that the nonlinear functions are convex. In the case of non-convex problems the bounds of the MIP master problem are not rigorous. Therefore, the global optimum can be cut off with the setting `stop 1`.
`2`	Stop on worsening Stop as soon as the NLP subproblems stop improving. This `worsening` criterion is a heuristic. For non-convex problems in which valid bounds can not be obtained the heuristic often works very well. Even on convex problems, in many cases it terminates the search very early while providing an optimal or a very good integer solution. The criterion is not checked before major iteration three.
`3`	Stop on crossover or worsening Stop as soon as a crossover occurs or when the NLP subproblems start to worsen. (This is a combination of 1 and 2).

usexinit *(boolean)*: Use the user initial point as starting point for all NLP solves ↩

Default: 0

weight *(real)*: Penalty parameter, set above 1e20 to disable penalty relaxation ↩

The value of the penalty coefficients.

Default: 1000

5.14.9 DICOPT Output

DICOPT generates lots of output on the screen. DICOPT itself and also the NLP and MIP solvers that handle the sub-problems write messages to the screen. The most important part is the last part of the screen output.

In this section we will discuss the output DICOPT writes to the screen and the listing file using the model procsel.gms (this model is part of the GAMS model library). A DICOPT log is written and the reason why DICOPT terminated is explanied.

```
--- DICOPT: Checking convergence
--- DICOPT: Search stopped on worsening of NLP subproblems
--- DICOPT: Log File:
 Major Major     Objective    CPU time  Itera- Evaluation Solver
 Step  Iter       Function     (Sec)    tions   Errors
  NLP    1         5.35021      0.05        8      0      conopt
  MIP    1         2.48869      0.28        7      0      cplex
  NLP    2         1.72097<     0.00        3      0      conopt
  MIP    2         2.17864      0.22       10      0      cplex
  NLP    3         1.92310<     0.00        3      0      conopt
  MIP    3         1.42129      0.22       12      0      cplex
  NLP    4         1.41100      0.00        8      0      conopt
--- DICOPT: Terminating...
--- DICOPT: Stopped on NLP worsening

      The search was stopped because the objective function
      of the NLP subproblems started to deteriorate.

--- DICOPT: Best integer solution found: 1.923099
--- Reading solution for model process
*** Status: Normal completion
```

Notice that the integer solutions are provided by the NLP's except for major iteration one (the first NLP is the relaxed NLP). For all NLP's except the relaxed one the binary variables are fixed, according to a pattern determined by the previous MIP which operates on a linearized model. The integer solutions marked with a ' <' are an improvement. We see that the NLP in cycle 4 starts to deteriorate, and DICOPT stops based on its default stopping rule.

Note that if the criterion stop 1 had been used the search would have been terminated at iteration 3. The reason is that the upper bound to the profit predicted by the MIP (1.42129) exceeds the best current NLP solution (1.9231). Since it can be shown that the MINLP involves convex nonlinear functions, 1.9231 is the global optimum and the criterion stop 1 is rigorous.

In case the DICOPT run was not successful, or if one of the subproblems could not be solved, the listing file will contain all the status information provided by the solvers of the subproblems. For each iteration the configuration of the binary variables will also be printed. This extra information can also be requested via the GAMS option:

```
option sysout = on ;
```

5.14.10 Special Notes

This section covers some special topics of interest to users of DICOPT.

5.14.10.1 Stopping Rule

Although the default stopping rule behaves quite well in practice there some cases where it terminates too early. In this section we discuss the use of the stopping criteria.

When we run the example procsel.gms with stopping criterion 0, we see the following DICOPT log:

```
--- DICOPT: Starting major iteration 10
--- DICOPT: Search terminated: infeasible MIP master problem
--- DICOPT: Log File:
 Major Major     Objective    CPU time  Itera- Evaluation Solver
 Step  Iter       Function     (Sec)    tions   Errors
  NLP    1         5.35021      0.06        8      0      conopt
  MIP    1         2.48869      0.16        7      0      cplex
  NLP    2         1.72097<     0.00        3      0      conopt
  MIP    2         2.17864      0.10       10      0      cplex
  NLP    3         1.92310<     0.00        3      0      conopt
  MIP    3         1.42129      0.11       12      0      cplex
  NLP    4         1.41100      0.00        8      0      conopt
  MIP    4         0.00000      0.22       23      0      cplex
  NLP    5         0.00000      0.00        3      0      conopt
  MIP    5        -0.27778      0.16       22      0      cplex
  NLP    6        -0.27778      0.00        3      0      conopt
  MIP    6        -1.00000      0.16       21      0      cplex
  NLP    7        -1.00000      0.00        3      0      conopt
  MIP    7        -1.50000      0.22       16      0      cplex
  NLP    8        -1.50000      0.00        3      0      conopt
  MIP    8        -2.50000      0.11       16      0      cplex
  NLP    9        -2.50000      0.00        3      0      conopt
  MIP    9        *Infeas*      0.11        0      0      cplex
--- DICOPT: Terminating...
--- DICOPT: Stopped on infeasible MIP

      The search was stopped because the last MIP problem
      was infeasible. DICOPT will not be able to find
      a better integer solution.

--- DICOPT: Best integer solution found: 1.923099
--- Restarting execution
--- PROCSEL.GMS(98) 0 Mb
--- Reading solution for model process
*** Status: Normal completion
```

This example shows some behavioral features that are not uncommon for other MINLP models. First, DICOPT often finds the best integer solution in the first few major iterations. Second, in many cases as soon as the NLP's start to give worse integer solution no better integer solution will be found. This observation is the motivation to make stopping option 2, where DICOPT stops as soon as the NLP's start to deteriorate, the default stopping rule. In this example DICOPT would have stopped in major iteration 4 (you can verify this in the previous section). In many cases this will give the best integer solution. For this problem, DICOPT has indeed found the global optimum.

Based on experience with other models, we find that the default stopping rule (stop when the NLP becomes worse) performs well in practice. In many cases it finds the global optimum solution for both convex and non-convex problems. In some cases, however, it may provide a sub-optimal solution. In those cases where you want more reassurance that no good integer solutions are missed you can use one of the other stopping rules.

Changing the MIP or NLP solver can change the path that `DICOPT` follows, since the sub-problems may have non-unique solutions. The optimum stopping rule for a particular problem depends on the MIP and NLP solvers used.

The bounds of the MIP master problem are not rigorous in the case of non-convex problems. Therefore, the global optimum can be cut-off with `stop 1`. However, this option is the best stopping criterion for convex problems.

5.14.10.2 Solving the NLP Problems

Using a combination of NLP solvers has been found effective in cases where the relaxed NLP and/or the other NLP sub-problems are very difficult. For example, `MINOS` has many more difficulties to establish if a model is infeasible, so one would like to use `CONOPT` for NLP subproblems that are either infeasible or barely feasible. The `nlpsolver` option can be used to specify the NLP solver to be used for each iteration.

Infeasible NLP sub-problems can be problematic for `DICOPT` Those subproblems cannot be used to form a new linearization. Effectively only the current integer configuration is excluded from further consideration by adding appropriate integer cuts, but otherwise an infeasible NLP sub-problem provides no useful information to be used by the `DICOPT` algorithm. If your model shows many infeasible NLP sub-problems you can try to use the `infeasder` option. Otherwise a strategy that can help is to introduce explicit slack variables and add them with a penalty to the objective function.

Assume your model is of the form:

$$\begin{array}{ll} \min & f(x,y) \\ & g(x,y) \sim b \\ & \ell \leq x \leq u \\ & y \in \{0,1\} \end{array}$$

where $\sim$ is a vector of relational operators $\{\leq, =, \geq\}$. x are continuous variables and y are the binary variables. If many of the NLP subproblems are infeasible, we can try the following `elastic` formulation:

$$\begin{array}{ll} \min & f(x,y) + M \sum_i (s_i^+ + s_i^-) \\ & y = y^B + s^+ - s^- \\ & g(x,y) \sim b \\ & \ell \leq x \leq u \\ & 0 \leq y \leq 1 \\ & 0 \leq s^+, s^- \leq 1 \\ & y^B \in \{0,1\} \end{array}$$

I.e., the variables y are relaxed to be continuous with bounds $[0,1]$, and binary variables y^B are introduced that are related to the variables y through a set of the slack variables s^+, s^-. The slack variables are added to the objective with a penalty parameter M. The choice of a value for M depends on the size of $f(x,y)$, on the behavior of the model, etc. Typical values are 100 or 1000.

5.14.10.3 Solving the MIP Master Problems

MIP master problems may become expensive to solve when there are many discrete variables. One of the first things to try is to see if a different MIP solver can solve your particular problems more efficiently.

Different formulations can have dramatic impact on the performance of MIP solvers. Therefore it is advised to try out several alternative formulations. The use of priorities can have a big impact on some models. It is possible to specify a nonzero value for `OPTCA` and `OPTCR` in order to prevent the MIP solver from spending an unacceptable long time proving optimality of MIP master problems.

If the MIP master problem is infeasible the `DICOPT` solver will terminate. In this case you may want to try the same reformulation discussed in the previous paragraph.

5.14.10.4 Feasibility Pump

The feasibility pump is similar to the Outer-approximation, but its objective is to completely focus on finding good feasible solutions rather than optimal ones. The main idea of this algorithm is to decompose the original mathematical programming problem in two parts: integer feasibility and constraint feasibility. By solving an MIP subproblem its solution is an integer feasible solution, which may violate the constraints; and by solving a continuous relaxation of the original MINLP (NLP subproblem) the solution is constraint feasible but might not be integral. By minimizing in successive iterations the distance between these two types of solutions it is expected to achieve a solution that is both constraint and integral feasible.

The feasibility pump can be used as a standalone solver for convex MINLP problems. This is achieved by iteratively applying the method, while including a bound to the objective function. This bound is obtained by the best known solution and an epsilon improvement. This will result in the global optimum of a convex MINLP [28]. The drawback of this algorithm is that it may require many iterations, since each time the objective function is restricted to improve only by epsilon.

In DICOPT, the feasibility pump can be applied before the Outer-approximation method by setting option feaspump. In the feasibility pump, large improvements in the objective function are enforced at each iteration (option fp_cutoffdecr). After the method finishes, all the cuts and the best known solution are passed to the Outer-approximation method to prove optimality. More details on the implementation of the feasibility pump in DICOPT can be found in [19].

5.14.10.5 Model Debugging

In this paragraph we discuss a few techniques that can be helpful in debugging your MINLP model.

- Start with solving the model as an `RMINLP` model. Make sure this model solves reliably before solving it as a proper `MINLP` model. If you have access to different NLP solvers, make sure the `RMINLP` model solves smoothly with all NLP solvers. `CONOPT`, especially, can generate useful diagnostics such as Jacobian elements (i.e. matrix elements) that become too large.
- Try different NLP and MIP solvers on the subproblems. Example: use the GAMS statement `OPTION NLP=KNITRO;` to solve all NLP subproblem using the solver `KNITRO`.
- The GAMS option statement `OPTION SYSOUT = ON;` can generate extra solver information that can be helpful for diagnosing problems.
- If many of the NLP subproblems are infeasible, add slacks as described in section Solving the NLP Problems.
- Run `DICOPT` in pedantic mode by using the `DICOPT` option: `CONTINUE 0`. Make sure all NLP subproblems solve to optimality.

- Don't allow any nonlinear function evaluation errors, i.e. keep the DOMLIM limit at zero. See the discussion on DOMLIM in section The OPTION Statement.
- If you have access to another MINLP solver such as AlphaECP, SHOT, or SBB or even global solvers like Antigone or BARON, try to use a different solver on your model. To select another solver (here SBB) use the following GAMS option statement: `OPTION MINLP=SBB;`.
- Individual NLP or MIP subproblems can be extracted from the MINLP by using the CONVERT solver, which will write a model in scalar GAMS notation that can then be solved using any GAMS NLP or MIP solver. E.g., to generate the second NLP subproblem, you can use the following DICOPT option: `NLPSOLVER CONOPT CONVERT`. The model will be written to the file GAMS.GMS. A disadvantage of this technique is that some precision is lost due to the fact that files are being written in plain ASCII. The advantage is that you can visually inspect these files and look for possible problems such as poor scaling.

5.15 EXAMINER

5.15.1 Introduction

This document describes GAMS/Examiner, a tool for examining points and making an unbiased, independent assessment of their merit. In short, it checks if solutions are *really* solutions. As an example, it can take a solution point reported as optimal by a solver and examine it for primal feasibility, dual feasibility, and optimality. Examiner has a number of different modes, allowing it to check the input point from GAMS/Base as well as the solution passed by a solver back to GAMS.

Many of the tests done by Examiner (perhaps all of them!) are already being done by the GAMS solvers, so Examiner is in a sense redundant. However, the ability to make an independent, transparent check of a solver's solution is very useful in solver development, testing, and debugging. It is also useful when comparing the solutions returned by two different solvers. Finally, a tool like the Examiner allows one to examine solutions using different optimality tolerances and optimality criteria in a way that is not possible when working with the solvers directly.

GAMS/Examiner is installed automatically with your GAMS system. Without a full GAMS/Base license, examiner will run in student or demonstration mode (i.e. it will examine small models only).

5.15.2 Usage

Examiner can be used with all supported model types. Since Examiner doesn't really solve any problems, it is not a good choice for a default solver, and when installing GAMS it does not appear as an option in the list of possible solver defaults. However, you can choose Examiner via the command line:

```
gams trnsport LP=examiner;
```

or via a GAMS option statement

```
option LP=examiner;
```

somewhere before the `solve` statement.

Since Examiner is not really a solver, many of the usual GAMS options controlling solvers have no impact on it. However, the `sysout` option is interpreted in the usual way.

The optimality checks done in Examiner are first-order optimality checks done at a given point. A discussion here of these conditions and all they imply would be redundant: any good intro text in optimization will cover them. For linear programming, first-order optimality is all one needs to prove global optimality. For nonlinear programming, these conditions may or may not be necessary or sufficient for optimality; this depends on the convexity of the feasible set and objective and the form of the constraints. For integer programming models, these checks only make sense if we turn the global problem into a local one by adding bounds to the model, essentially fixing each discrete variable to its current value: these bounds are added automatically by Examiner.

Examiner runs in two basic modes of operation: it can examine the input point passed from GAMS/Base to the solver, and it can examine the point passed from the solver back to GAMS. Each mode can be used independent of the other. By default it will operate in the first mode, examining the initial "solution" passed to it by GAMS, but only if GAMS indicates it is passing an advanced basis to the solver (cf. the GAMS User Guide and the `bratio` option). If you wish to use the second `solver-check` mode, you may specify an appropriate subsolver using the `subsolver` option (see section Options). If no `subsolver` is selected, the default solver for the model type being solved is used. In most cases you will want to use an option file to specify exactly what type of examination you wish to perform. The rules for using an option file are described in The Solver Options File.

5.15.2.1 Solution Points: Definition

There are a number of different ways a solution point can be defined. Of course the different definitions will typically result in the same points being produced, but there are cases where this will not be precisely so. Since Examiner is intended to explore and analyze these cases, we must make these definitions precise. The following four points are defined and used in Examiner:

1. The `gamspoint` is the input point provided by GAMS to Examiner. The GAMS input point includes level & marginal values for the rows and columns: Examiner uses these exactly as given.
2. The `initpoint` is determined by the variable levels (primal vars) and equation marginals (dual vars) provided by GAMS to Examiner. These values are used to *compute* the equation levels and variable marginals / reduced costs using the function evaluator in Examiner, rather than using the values passed in by GAMS.
3. The `solupoint` is similar to the `initpoint`: it uses the variable levels (primal vars) and equation marginals (dual vars) to *compute* the equation levels and variable marginals. The variable levels and equation marginals used are those returned by the subsolver.
4. The `solvpoint` is the point returned by the subsolver. The subsolver returns both level and marginal values for the rows and columns: Examiner uses these, exactly as given.

5.15.2.2 Checks Performed

There are a number of checks that can be performed on any of the solution points. By default, Examiner tries to choose the appropriate checks. For example, if a primal simplex solver returns a model status of nonoptimal, the only checks that make sense are feasibility in the primal variables and constraints. However, this automatic choice of appropriate checks is not possible when checking points passed in from GAMS/Base.

1. **Primal variable feasibility**: check that all primal variables are within bounds.

2. **Primal constraint feasibility**: check that all primal constraints are satisfied.
3. **Dual variable feasibility**: check that all dual variables are within bounds.
4. **Dual constraint feasibility**: check that all dual constraints are satisfied.
5. **Primal complementary slackness**: check complementarity between the primal variables and the dual constraints / reduced costs.
6. **Dual complementary slackness**: check complementarity between the dual variables / equation marginals and the equation slacks.
7. **Equilibrium condition complementarity**: check complementarity of the equation/variable pairs in complementarity models (MCP, MPEC).

The checks above are implemented with default tolerances. These tolerances can be changed via an option file (see section Options).

Different ways exist to check the items mentioned above. For example, different norms can be used to measure the error of the residual when checking for primal feasibility. Currently, we have only implemented one way to make these checks.

For optimization models, it is possible to define a merit function that measures how well a given point satisfies conditions 1 through 6 above. Such a function is always nonnegative, and its value represents the error between the given point and a true solution. If such an input point is well-defined (e.g. if only one point is selected for examination) then Examiner will compute the merit function value at this point and return it in the RObj model attribute.

5.15.2.3 Scaling

By default, Examiner makes its checks on the original, unscaled model. In many cases, however, it is important to take scaling into account. Consider the effect of row scaling on the simple constraint $x^2 \leq 9$ where $x = 3.5$. Multiplying this constraint through by large or small constants changes the amount of the constraint violation proportionately, but the distance to feasibility is not changed. Applying row scaling to the original model eliminates this problem.

Most solvers scale a model before solving it, so any feasibility or optimality checks and tolerances are applied to the scaled model. The process of unscaling the model can result in a loss of feasibility or optimality. Even though we do not have access to the scales applied by the solver and cannot precisely construct the same scaled model, we can get a better idea of how the solver performed by looking at a model scaled by Examiner than by looking at the original.

It is also interesting to see what the model scaling looks like, even if we do not apply the scales to do the Examiner checks. If the row scales are in a nice range, say [.1,100], we can have some confidence that the model is well-scaled. In contrast, if the row scales are in the range `[1,1e8]` we may question the precision of the solution provided.

For each row, Examiner computes the true row scale as

$$\max(\|RHS_i\|, \max_j(\|A_{ij}\| \cdot \max(1, \|x_j\|)))$$

In this way variables with a large level value lead to large scale factors. To make the scale factor independent of the variable values, use an option file line of `"AbsXScale 0"`. This replaces the term $\max(1, \|x_j\|)$ above with 1.

Since the user may wish to limit the size of the scale factors applied, the true row scales are projected onto the scale factor bounds to get the applied scale factors. The scale factors are applied when making a scaled check by dividing the rows by the scale factors and multiplying the corresponding Lagrange multipliers by these same factors. When making unscaled checks information about the true scales is still included in the output to give the user a hint about potential scaling issues.

Note that the scaled and unscaled checks are made independently. By default only the unscaled checks are performed. If you turn the scaled checks on via an option file line `"scaled 1"`, this will not turn off the unscaled checks. You will need an option file line of `"unscaled 0"` to turn off unscaled checks.

5.15.3 Options

The following options control the behavior of GAMS/Examiner. Many of these are boolean (i.e. on/off) options. In this case, zero indicates off, nonzero on. For details on how to create and use an option file, see the section on the Solver Option File.

5.15.3.1 General Options

Option	Description	Default
absXScale	Whether to make scale factors dependent on x values. If on, the matrix coefficients are multiplied by max(1,abs(x)) when computing the scale factors. If off, the matrix coefficients are taken as is. See Section Scaling.	1
dumpGamsPoint	Whether to dump the GamsPoint to a basis file in GAMS source format.	0
dumpInitPoint	Whether to dump the InitPoint to a basis file in GAMS source format.	0
dumpSoluPoint	Whether to dump the SoluPoint to a basis file in GAMS source format.	0
dumpSolvPoint	Whether to dump the SolvPoint to a basis file in GAMS source format.	0
examineGamsPoint	Whether to examine the GamsPoint.	0
examineInitPoint	Whether to examine the InitPoint. By default, this option is on if GAMS/Base passes an advanced basis, and off otherwise.	auto
examineSoluPoint	Whether to examine the SoluPoint. By default, this option is on if a subsolver has been selected, and off otherwise.	auto
examineSolvPoint	Whether to examine the SolvPoint. By default, this option is on if a subsolver has been selected, and off otherwise.	auto
fCheckAll	If set, forces all checks on or off.	auto
fCheckATTR	If set, forces the model attributes check on or off.	auto
fCheckDCMP	If set, forces the dual complementary slackness check on or off.	auto
fCheckDCON	If set, forces the dual constraint feasibility check on or off.	auto
fCheckDVAR	If set, forces the dual variable feasibility check on or off.	auto
fCheckPCMP	If set, forces the primal complementary slackness check on or off.	auto
fCheckPCON	If set, forces the primal constraint feasibility check on or off.	auto
fCheckPVAR	If set, forces the primal variable feasibility check on or off.	auto
objvarAutoAdjust	Adjust objective variable to satisfy objective equation	0
perpSys	Controls output during examination of solution points. If on, print out the point in a way that allows for easy visual inspection and verification of the KKT or first order optimality conditions. First, the primal level values and bounds are printed next to the reduced costs. Next, the duals levels and bounds are printed next to the row slacks.	0
returnGamsPoint	Whether to return the GamsPoint as a solution to GAMS/Base.	0
returnInitPoint	Whether to return the InitPoint as a solution to GAMS/Base.	auto

Option	Description	Default
returnSoluPoint	Whether to return the SoluPoint as a solution to GAMS/Base.	auto
returnSolvPoint	Whether to return the SolvPoint as a solution to GAMS/Base.	auto
scaled	Whether to apply checks to a scaled version of the model.	0
scaleLB	Lower bound for applied row scales.	1
scaleUB	Upper bound for applied row scales.	maxdouble
showSlacks	explicitly show the infeasibilities or slacks for failed checks	0
subSolver	Indicates what subsolver to run. By default, the subsolver used is the default subsolver for the model type in question.	auto
subSolverOpt	optfile value to pass to the subsolver	auto
trace	If set, trace information will be computed and appended to this file.	none
unScaled	Whether to apply checks to the original, unscaled version of the model.	1

5.15.3.2 Tolerance Options

Option	Description	Default
dualCSTol	Tolerance on dual complementary slackness, i.e. between the dual variables and the primal constraints.	1e-7
dualFeasTol	Tolerance on dual feasibility, i.e. to check feasibility of the dual variables and the dual constraints.	1e-6
ECTol	Tolerance on equilibrium condition complementarity. Applicable to MCP and MPEC models, where the equilibrium conditions are given by the equation-variable pairs in the model statement.	1e-6
primalCSTol	Tolerance on primal complementary slackness, i.e. between the primal variables and the dual constraints.	1e-7
primalFeasTol	Tolerance on primal feasibility, i.e. to check feasibility of the primal variables and constraints.	1e-6
showTol	relative tolerance for showSlacks - we only want to see explicit slacks that are relatively small	1e-4

5.16 GAMSCHK

Abridged GAMSCHK USER DOCUMENTATION - Version 1.1
A System for Examining the Structure and Solution Properties of GAMS Model Instances

Author

Bruce A. McCarl, Professor, Department of Agricultural Economics, Texas A&M University

Date

April 2013

5.16.1 GAMSCHK USER DOCUMENTATION

This document describes procedures designed to aid users who wish to examine empirical GAMS models for possible flaws. The conceptual basis for many of the routines herein is supplied in McCarl and Spreen, and McCarl et.al.

The function of the specific components of GAMSCHK are to:

- List coefficients for user selected equations and/or variables using the DISPLAYCR procedure.
- List the characteristics of selected groups of variables and/or equations using MATCHIT.
- List the characteristics of equation and variable blocks using BLOCKLIST.
- Examine a GAMS model to see whether any variables and equations contain specification errors using ANALYSIS.
- Generate schematics depicting the characteristics of coefficients by variable and equation blocks using BLOCKPIC.
- Generate a schematic for small GAMS models or portions of larger models depicting the location of coefficients by sign and magnitude using PICTURE.
- Reconstruct the reduced cost of variables and the activity within equations after a model solution using POSTOPT.
- Help resolving problems with unbounded or infeasible models using NONOPT and ADVISORY.

5.16.2 General Notes on Package Usage

GAMSCHK must replace a solver. This is done using a GAMS option statement of the form:

```
OPTION LP= GAMSCHK;
```

or

```
OPTION NLP=GAMSCHK;
```

or

```
OPTION MIP=GAMSCHK;
```

which replaces either the solver of the particular model type with GAMSCHK. In turn, the user will invoke the solver using the statement:

```
SOLVE MODELNAME USING LP MINIMIZING OBJNAME;
```

where MODELNAME is the name used in the GAMS MODEL statement; OBJNAME is the objective variable name for the model; and the type of solver that GAMSCHK has replaced which must also be able to solve this type of problem (LP, NLP, MIP, ...) is identified. The following are examples of GAMS sequences which can be added to the GAMS file:

```
OPTION NLP=GAMSCHK;
SOLVE TRANSPORT USING NLP MINIMIZING Z;
```

or

```
OPTION LP=GAMSCHK;
SOLVE FEED USING LP MINIMIZING COST;
```

or

```
OPTION MIP=GAMSCHK;
SOLVE RESOURCE USING MIP MAXIMIZING PROFIT;
```

5.16.2.1 Selecting a Procedure and Providing Input - the ∗.GCK File

GAMSCHK requires that the user indicate which procedures are to be employed. This is specified through the use of the ∗.GCK file where the ∗ refers to the filename from the GAMS execution instruction. [2] The general form of that file is:

```
FIRST PROCEDURE NAME
  ITEM SELECTION INPUT

SECOND PROCEDURE NAME
  ITEM SELECTION INPUT
```

Spaces and capitalization are ignored in this input. For example, a ∗.GCK file could look like

```
 DISPLAYCR
          variables
              SELL(*,*,FANCY)
              maketable
          Invariables
              transport(plant2,*,fancy)
          Equations
              objT
              notthere
          inequations
              resourceq(plant1)
PICTURE
```

The first procedure name in this case is DISPLAYCR and the following 10 lines indicate the items to be selected. Then, we also request PICTURE. Selection entries are treated using several assumptions. In particular:

1. If the ∗.GCK file is empty then it is assumed that the BLOCKPIC procedure is selected.
2. Spaces maybe freely used in the GCK input file.
3. Upper, lower, or mixed case input is accepted.
4. GAMSCHK recognizes certain words. These words are listed in Appendix A: Reserved Names and cannot be used as variable or equation names.

5.16.2.2 The ∗.GCK file: General Notes on Item Selection

Some of the procedures permit selection of variables, equations or functions. Specifically, the DISPLAYCR, PICTURE, POSTOPT, and MATCHIT procedures accept input identifying the variables and equations to be utilized. Also NONOPT accepts limited input controlling its function. General observations about the selection requests are

1. Variables can be chosen by entering the word VARIABLE or VARIABLES possibly with a modifier, followed by variable selection statements.

[2] Thus, if the GAMS instructions are in the file called MYMODEL, and GAMS is invoked using the DOS command GAMS MYMODEL, then the GCK file would be called MYMODEL.GCK. If GAMS instructions are on the filename with a period in it then the name up to the period will be used, i.e., the GCK file associated with MYMODEL.IT would be MYMODEL.GCK

2. Variables can also be selected using the INEQUATION or INEQUATIONS syntax followed by names of equations. Use of this syntax results in selection of variables with coefficients in the named equations.

3. Equations are selected by entering the keyword, EQUATION or EQUATIONS possibly with a modifier, followed by equation selection statements.

4. Equations can also be selected using the INVARIABLE or INVARIABLES syntax followed by names of variables. Use of this syntax results in selection of equations in which the named variables have coefficients.

5. Certain item selection modifier keywords can be used depending on procedure. The INTERSECT keyword works with procedures DISPLAYCR and POSTOPT. The INEQUATION and INVARIABLE keywords work with procedures DISPLAYCR, PICTURE and POSTOPT. LISTEQUATION and LISTVARIABLE keywords work with the MATCHIT procedure. INSOLUTION, NOTINSOLUTION, BINDING, and NOTBINDING keywords work with POSTOPT. The keywords VERBOSE and IDENTIFY work with NONOPT.

6. If variable or equation names do not follow the keyword, then usually all variables or equations are assumed selected.

When variables or equations are to be selected after an item selection keyword, a number of input conventions apply. These conventions are:

1. If a variable or equation name is entered without any following parentheses, then all cases for that variable or equation are selected.

2. The selection entries identify specific elements from among the sets over which the variables and equations are defined. In specifying these elements one can use various wild card entries as discussed below or an element name. Note GAMS set or subset names cannot be used. Set membership information is not available to the GAMSCHK routines.

3. Wild cards can be used to select items. An "$*$" will select any item. For example, "B$*$" will select anything starting with a B. "A?B" will select anything beginning with A, ending with B with one intervening alpha numeric character.

4. When individual elements are specified, you need not enclose them in quotes (").

5. Quotes must be specified to include set item names with spaces, and special characters. In that case wild cards do not work and all input up to the next quote is simply copied.

6. When the selected item has more dimensions than specified, then all later dimensions are handled as if a wild card were specified. For example, when a variable X is defined with reference to 4 sets in the GAMS instructions, but only 3 parameters are specified in the GAMSCHK input, then the request is handled as if all elements of the 4th are desired.

7. When the selected item has less dimensions in GAMS than in the item selection input, then all additional dimensions are ignored. Thus, when a variable X is defined with reference to 3 sets in GAMS, but 4 parameters are specified in the item selection file, then the 4th specification is ignored.

8. Multiple selection statements can appear on successive lines of the $*$.GCK file. Output is ordered according to the way items are found in the GAMS file which is determined by the ordering of variables, equations, and set elements in the original GAMS input.

9. Error messages will be generated when an entry cannot be matched to a GAMS element.

10. Examples include

Example	Explanation
X(∗,CLEVELAND)	which indicates that X will be selected for any element of the first set where the element in the second set equals CLEVELAND
X(SEATTLE)	when X is two dimensional selects all cases where the first set element is SEATTLE
X(SEATTLE,CHICAGO,Z)	when X is two dimensional selects the case where the first set element equals SEATTLE, and the second element equals CHICAGO. The third is ignored.
X	all X's will be selected
X(S∗, C.O, Z)	when X is three dimensional selects where all X's with first element starting with S, second element beginning with C and ending with O and third element Z will be selected.
∗	all variables or equations will be selected
{empty selection set}	all variables or equations will be selected

5.16.2.3 Procedure Output

In all cases the output generated by the procedure will be written to the ∗.LST file associated with the GAMS call. Thus, if the file is called MODEL with the ∗.GCK file (MODEL.GCK), then all output will be on MODEL.LST.

5.16.2.4 Nonlinear Terms

GAMS models examined with GAMSCHK may involve nonlinear terms. In such cases, GAMSCHK uses the value of the nonlinear term sent forth from GAMS which is an accurate marginal, not total value. GAMS develops this value based on the current level value of the variable. This will either be: a) the starting point selected by GAMS, if the model has not been solved, or b) the current solution value, if the model has been solved. The most accurate portrayals of the coefficients will be generated after the model has been solved through a GAMS SOLVE command before invoking GAMSCHK. Some cases may require a solution and/or the specification of a good starting point before using GAMSCHK. Also, nonlinear terms potentially cause misleading coefficients as those values are local marginal, not global, values determined by the current levels of the variables. Nonlinear terms are marked with ∗∗∗ in the DISPLAYCR, POSTOPT, and NONOPT output.

5.16.2.5 Entering Comments in the ∗.GCK File

The ∗.GCK file has been programmed so that users can enter comments. These comments can take one of two forms. Comments that begin with a hash mark are copied to the output when the program runs. Comments which begin with a question mark are simply overlooked. Thus, one can temporarily comment GAMSCHK selection statements making them inactive by putting in question marks. If multiple procedures are being run or if some sort of output is decided to screen in the computer output then the hash marks can be entered.

5.16.2.6 Controlling Page Width in the ∗.GCK File

When running multiple procedures, in particular the pictures with other procedures, it is often desirable to have some procedures run with wide page widths, but the rest with a narrower page width. The GCK file provides the option to narrow the page width using a PW= command. In particular, what one can do is run GAMS with a large page width, i.e. run GAMS BLOCK pw=200, then insert in the GCK file instructions which narrow that page width for selected procedures. Users should note that the page width can never be made any wider than the default page width when running with GAMS. Information in excess of the page width will be ignored. Thus, if the model is run under the default status which has a page width of 75 characters then GAMSCHK will reduce the page width down to the maximum page width allowed. Consequently, the pw= command can only be used to narrow the page width from the default page width, not increase it.

5.16.2.7 Running Multiple Procedures

GAMSCHK can run multiple procedures during one job. This is done by simply stacking the sequence of the commands in the .GCK file.

5.16.3 Use of the Procedures

The following section describes the procedures available in GAMSCHK and their input requirements.

5.16.3.1 DISPLAYCR

Brief Purpose: DISPLAYCR displays all coefficients from the empirical model for a set of user selected equations and variables. All nonzero coefficients under each selected variable or in each selected equation are displayed with the associated variable or equation name and coefficient value. The selection entries may refer to all terms in equations under variables or only those coefficients at the intersection of the selected variables and equations.

Usage Notes: This option mirrors the GAMS LIMCOL and LIMROW options, but allows the user to select the specific items to be displayed. Partial displays within a variable or equation are also allowed using INTERSECT. Use of VARIABLE and EQUATION keywords followed by selection statements allows one to select variables and equations. Use of the INVARIABLE command allows users to select the equations which are associated with a particular variable. For example, if one is having trouble with a particular variable and wants to look at competition in the equations in which it appears, then selecting the variable under the INVARIABLE command will display the complete contents of all the equations in which the selected variables have coefficients. Similarly, the INEQUATION command will display the complete contents of all variables which fall in a particular equation. Nonlinear terms are marked with ***. When the keyword INTERSECT is found then only the coefficients at the intersection of the specified equations and variables are selected. Use of INTERSECT with the INVARIABLE syntax results in the named variables and the equations in which they fall being selected. Similarly, use of INTERSECT with the INEQUATION syntax results in selection of the named equations and the variables which fall in those equations

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: The keyword DISPLAYCR is entered followed by optional lines of item selection input identifying the variables and equations to be displayed. This file can contain the keywords VARIABLE, INVARIABLE, EQUATION, and INEQUATION, with each followed by a specification of the items to be selected using the procedure input specification conventions that were described above. The keyword INTERSECT can also be used. Several special cases are relevant:

- If none of the above keywords are found after DISPLAYCR and another procedure name does not follow, then the input is assumed to identify variables.
- If input is found but the VARIABLE or INEQUATION keyword cannot be found then no variables are assumed selected.
- If the VARIABLE keyword is entered, but is followed by the end of file or an Appendix A reserved word and INEQUATION does not appear, then all variables are assumed selected.
- If the EQUATION or INVARIABLE keyword cannot be found, then no equations are assumed selected.
- If the EQUATION keyword is entered, but is followed by the end of the file or a reserved word and the INVARIABLE command does not occur, then all equations are assumed selected.

- The keyword INVARIABLE is allowed. It should be followed by variable selection statements. In turn, DISPLAYCR selects all equations which have nonzero entries under the INVARIABLE selections.
- The keyword INEQUATION may be used. It should be followed by equation selection statements. In turn, DISPLAYCR selects all variables which have nonzero entries in the INEQUATION selections.
- The keyword INTERSECT causes only coefficients at the intersection of the specified equations and variables to be displayed. This occurs for all specifications in this run of DISPLAYCR. One should use DISPLAYCR again if some intersecting and some non-intersecting displays are desired.
- When INTERSECT appears along with INVARIABLE, the named variable is selected along with all the equations in which it falls. Similarly, when INTERSECT and INEQUATION appear then all the named equations and the variables appearing in them are selected.

5.16.3.2 MATCHIT

Brief Purpose: MATCHIT retrieves the names and characteristics of selected variables and equations. The characteristics reported tell whether the items are nonlinear as well as reporting scaling characteristics and counts of the coefficients. MATCHIT will summarize the items which match a request or list all the items individually.

Usage Notes: The input to MATCHIT can include the keywords VARIABLE and EQUATION along with those keywords with the prefix LIST attached. When the LIST prefix is not used, the procedure summarizes the characteristics of all items which match the item requests counting the number of matching items, the number of those items which are nonlinear, the total coefficients under or in those items, the number of positive, negative, and nonlinear coefficients that fall under or in those items. This does not list the names of the individual items which match. If the LIST prefix is used (entering LISTVARIABLE or LISTEQUATION) then the individual matching items are printed in the order in which they are encountered. For each matching item the information tells whether it is nonlinear, how many total coefficients it has, the count of positive, negative, and nonlinear coefficients falling under it, and the minimum and maximum absolute values of coefficients under it (excluding the objective function coefficient).

Note that when GAMS internal scaling features are employed then by default scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: This file contains the keyword MATCHIT, followed by optional item selection input data. The optional input identifies the variables and equations to be displayed. This input can contain the keywords VARIABLE or LISTVARIABLE followed by a specification of the variables to be selected using the procedure input specification conventions that were described above. This can be followed by the keyword EQUATION or LISTEQUATION and the specified entries.

Several special cases are relevant:

- If the procedure name is not followed by any selection input, then a count of all variables and equations appears.
- If the input is found, but the input does not begin with VARIABLE, EQUATION, LISTVARIABLE, or LISTEQUATION keywords, then the input is assumed to contain variable names.
- If the VARIABLE keyword is entered, but is not followed by variable selection statements, and LISTVARIABLE does not appear, then all variables are assumed selected.
- If the EQUATION or LISTEQUATION keyword cannot be found, then equations are assumed selected.

- If the EQUATION keyword is entered, but is not followed by equation selection statements or a LISTEQUATION entry, then all equations are assumed selected.
- The keyword LISTVARIABLE is allowed. It should be followed by variable selection statements. In turn, MATCHIT lists all variables which fall under the request.
- The keyword LISTEQUATION may also be used. It should be followed by equation selection statements. In turn, MATCHIT lists all equations which fall under the request.

5.16.3.3 ANALYSIS

Brief Purpose: Analyzes the structure of all variables and equations. Information is given on errors involving obvious model misspecifications causing redundancy, zero variable values, infeasibility, unboundedness, or obvious constraint relaxations in linear programs. The checks are those identified in Tables 1, 2 and 3.

Usage Notes: The analysis tests given in Tables 1 and 2 are utilized to determine if individual variables or equations in the model possess obvious specification errors. One test, for example, considers whether or not in a maximization problem a variable appears which has a positive return in the objective function, but no coefficients in the constraints indicating an obviously unbounded model. Similarly, information is provided on whether certain equations can never be satisfied. For example, tests examine whether an equality equation appears with a negative right hand side and all positives on the left hand side. Also tests see whether the bounds on variables preclude equation satisfaction or make equations redundant (Table 3). In ANALYSIS these tests are applied to each and every variable and equation. The BLOCKPIC and BLOCKLIST routines utilize the tests on a block by block basis. Thus, the messages will be triggered only if every variable or equation in that block has the same problem. Also interactions between variables and equations are not checked so ANALYSIS only finds flaws contained in individual variables/equations.

Input File: The keyword ANALYSIS is all that is accepted.

5.16.3.4 BLOCKLIST

Brief Purpose: The BLOCKLIST procedure displays the number and characteristics of the items in each GAMS variable and equation block.

Usage Notes: The characteristic information gives:

1. The variable sign restriction or equation inequality type.
2. The number of variables or equations in this block;
3. The number of variables or equations with at least one nonlinear term in this block.
4. The number of positive coefficients under the variables or in the equations.
5. The number of negative coefficients under the variables or in the equations.
6. The number of nonlinear coefficients under the variables or in the equations.
7. The largest coefficient in absolute value in this block;
8. The smallest coefficient in absolute value in this block. Analysis tests are also performed as discussed under the ANALYSIS procedure.

Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: No input other than the procedure name is needed.

5.16.3.5 BLOCKPIC

Brief Purpose: Generates model schematics and scaling information. The schematics depict coefficient signs, total and average number of coefficients within each GAMS equation and variable block.

Usage Notes: These schematics are designed to aid users in identifying flaws in coefficient placement and sign. The summary information on problem scaling characteristics is designed to help users in scaling data. The scaling information is usually reported after any GAMS scaling (using the variablename.scale and equationname.scale features) but before solver scaling. (The user can change whether descaling is done - see the options file). Analysis tests are done using the procedures in Tables 1 and 2.

Note that when GAMS internal scaling features are employed the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: The keyword BLOCKPIC is all that is recognized.

5.16.3.6 PICTURE

Brief Purpose: Generates a schematic depicting the location, sign and magnitude of coefficients for selected variables and equations. Users can use this schematic to help identify flaws in coefficient placement, magnitude, or sign. Reports are also generated on the number of individual elements in the pictured portions of each variable and equation.

Usage Notes: This output can be quite large, so PICTURE should only be used for small models or model components. Note that when GAMS internal scaling features are employed, the default option is that the scaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: Optional input instructions may appear after the PICTURE keyword. This input selects the variables and equations to be included. Only coefficients at the intersection of the selected variables and equations are portrayed. The selected item in the .GCK file can contain the keywords VARIABLE, or INVARIABLE followed by a specification of the selected variables using the procedure input specification conventions above. This can be followed by the keywords EQUATION or INEQUATION and the specified entries. Several special cases are also relevant:

- If the VARIABLE or INEQUATION keywords cannot be found, then all variables are assumed selected.
- If the EQUATION or INVARIABLE keywords cannot be found, then all equations are assumed to selected.
- If the none of the VARIABLE, INVARIABLE, EQUATION, or INEQUATION keywords are found, everything is pictured and all other input is ignored.
- When the INVARIABLE keyword is used, then all equations in which those variables have coefficients are selected along with the named variables.
- When the INEQUATION keyword is used, then all variables which have coefficients in the named equations are selected along with the named equations.

5.16.3.7 POSTOPT

Brief Purpose: Does post optimality computations. In that capacity POSTOPT either:

- Reconstructs the reduced cost of variables after a GAMS model solution. Modelers can use this information to discover why certain variables are nonbasic or why certain shadow prices take on particular values, or
- Reconstructs the usage and supply across an equation after a GAMS model solution. Modelers can use this information to discover why certain variables or slacks take on particular values, as well as to find out where items within equations are produced and/or used.

Usage Notes: POSTOPT uses essentially the same input conventions as does DISPLAYCR. Thus, the usage notes in that selection are also relevant here. In addition:

1. POSTOPT requires a solution has been obtained GAMSCHK will automatically cause a solver to be invoked unless suppressed by the options file;
2. Nonlinear terms may not be accurate in the row sums as their marginal value not their total value is used but GAMS will have adjusted the right-hand sides for their presence; and
3. Attention can be restricted to only certain types of variables or equations. Variables that are INSOLUTION (Nonzero or with Zero marginals), NOTINSOLUTION (zero with a nonzero marginal) can be requested, BINDING or NONBINDING equations can be focused on.

Note that when GAMS internal scaling features are employed, the default option is that the unscaled output is displayed. This can be altered using the DESCALE feature of the solver options file.

Input File: An optional input file is read in, indicating the specific variables desired using the conventions explained under DISPLAYCR above. In addition:

- One can enter INSOLUTION to restrict attention to variables which are nonzero or have zero marginals.
- One can enter NOTINSOLUTION to restrict attention to zero variables.
- The above entries restrict alteration in all VARIABLE or INEQUATION selection statements in a POSTOPT run.
- One can enter BINDING to only consider equations with zero slack. Similarly, NONBINDING considers equations with nonzero slack.
- The above equation specifications restrict all sections by all EQUATION or INVARIABLES items in a POSTOPT run.

5.16.3.8 ADVISORY

Brief Purpose: To identify variables which could be unbounded or equations and variable bounds which could cause a model to be infeasible.

Usage Notes: The ADVISORY procedure causes a presolution report on the set of all: a) variables which could be unbounded and/or b) equations and variable bounds which could cause infeasibility. The tests used are summarized in Table 3. This procedure identifies all variables which would need to be bounded as well as all constraints which need artificial variables if one wishes to diagnose problems in a model. The same output is also generated by NONOPT but the ADVISORY version does not require a solution.

Input file: Just the word ADVISORY

5.16.3.9 NONOPT

Brief Purpose: To help diagnose unbounded and infeasible models.

Usage Notes: The NONOPT procedure can be used in either an informative mode or with models which terminate as unbounded or infeasible. NONOPT will look through an optimal model reporting all variables which may be potentially unbounded or infeasible and all equations which may be infeasible using the checks explained under the ADVISORY section. Also in an unbounded model NONOPT can report the names of unbounded or infeasible variables or equations as well as either budgeting or row summing them. NONOPT runs after a solution and causes a solve to occur.

Input File: NONOPT may be followed by optional keywords IDENTIFY or VERBOSE. The IDENTIFY keyword causes GAMSCHK to report potential unbounded variables and/or infeasible equations. VERBOSE causes full budgets and row summing as done by the POSTOPT procedure on infeasible equations, and/or variables as well as unbounded variables and/or equations. Only the last encountered of the VERBOSE or IDENTIFY keywords will be obeyed. The details on these options are as follows:

1. If the IDENTIFY keyword is used, then the rules in Table 3 are applied to the model. Identify also anticipates that large upper bounds and/or artificial variables may be present. In an optimal condition all variable and equation levels that have exponents greater than the user supplied level filter in the options file (or 6 by default) are identified as items which could be involved with an unbounded model. Similarly, all variables or equations with marginals greater in exponent than the user supplied marginal exponent filter will be identified as items potentially involved with an infeasible model.
2. When the VERBOSE keyword is read then all variables and equations which are listed as nonoptimal or infeasible are treated using the budgeting and row summing aspects of POSTOPT.
3. When no keyword is found and the model solution is not optimal then the nonoptimal equations, infeasible equations and/or nonoptimal variables automatically listed.

5.16.4 Options File

GAMSCHK accepts an option file controlling solver choice (when needed); descaling; and size of the nonoptimal filters; the number of variable and column blocks selection entries allowed. The file is called GAMSCHK.OPT

5.16.4.1 Solver Choice Options

GAMSCHK calls for the solution of the problem when the POSTOPT or NONOPT procedures are used. In doing this, GAMSCHK internally selects the default GAMS solver for a problem class. Users may override this choice using the solver options file. Users may also force or suppress the solution process.

There are 16 solver related keywords allowed in the options file. These are as follows:

OPTION	Purpose
LP	Gives name of solver for LP problems
MIP	Gives name of solver for MIP problems
RMIP	Gives name of solver for RMIP problems
NLP	Gives name of solver for NLP problems
MCP	Gives name of solver for MCP problems
MPEC	Gives name of solver for MPEC problems

OPTION	Purpose
RMPEC	Gives name of solver for RMPEC problems
CNS	Gives name of solver for CNS problems
DNLP	Gives name of solver for DNLP problems
MINLP	Gives name of solver for MINLP problems
RMINLP	Gives name of solver for RMINLP problems
SOLVERNAME	Gives name of solver to be used regardless of problem type
NOSOLVE	Suppresses solution of the problem
SOLVE	Forces solution of the problem
DESCALE	Controls treatment of scaling
OPTFILE	Solver options file number

In the first five cases, the option name is followed by the name of one of the licensed solvers. If the options file is empty, then the default solver will be used. If a solver name is given, then that solver will be used provided it matches the name of a solver GAMS recognizes.

5.16.4.2 When Should I Use SOLVE or NOSOLVE

Ordinarily GAMSCHK will cause a solver to be used if either the POSTOPT or the NONOPT options are used. However, users can force solutions under other cases or suppress solutions if desired.

One should only force a solution (using the SOLVE option) when one wishes to use the solution information after GAMSCHK is done either to examine the solution output or do post optimality calculations. Forcing a solution will not cause GAMSCHK to have improved representations of nonlinear terms. That will only occur when a SOLVE statement is executed before the SOLVE statement involving GAMSCHK.

5.16.4.3 Control of Number of Variable and Row Selections Allowed

The GAMSCHK program uses an upper estimate on the number of variable or equation blocks. In rare circumstances users may wish to override this choice. The options for this are:

OPTION	Purpose
VARBLOCK	Maximum number of variable blocks allowed
EQUBLOCK	Maximum number of equation blocks allowed

These options are followed by a number, but should not be routinely used.

5.16.4.4 Scaling

GAMS users may be utilizing internal features which involve scaling through the Modelname.SCALEOPT=1, VariableName.SCALE, and EquationName.SCALE options. GAMSCHK can work with these options to create output which reflects scaled, unscaled or partially unscaled output. In particular, the command DESCALE can be entered with one of three options: NEVER, ALL, or PART. If you enter NEVER, then none of the model output will be descaled. If you enter ALL, then all of the model output will be descaled. The third option is to use PART. In that case the NONOPT and POSTOPT output will be descaled whereas scaled information will be displayed for PICTURE, BLOCKPIC, BLOCKLIST, MATCHIT and DISPLAYCR. The PART option allows investigation of scaling. If you do not enter a DESCALE option then all information will be reported as if the PART option was chosen.

5.16.4.5 NONOPT Filters

The NONOPT model in "IDENTIFY" mode checks through a model solution to identify large marginals and/or large variable values. The limits on these checks are provided by two options:

OPTION	Purpose
LEVELFILT	Numerical value of exponent on "unbounded levels"
MARGFILT	Numerical value of exponent on "infeasible marginals"

These options provide upper bounds on the exponents of the absolute values for the levels and marginals. They are followed by an integer which gives the exponent. Thus, entries like

```
LEVELFILT  7
MARGFILT   7
```

will cause the reporting of all marginals and levels which are greater in absolute value than 10^7.

5.16.4.6 Example Options File

The GAMSCHK option file is called GAMSCHK.OPT. An example of a file could look like the following 6 lines:

```
LP          SOPLEX
MIP         CBC
VARBLOCK    50
SOLVE
DESCALE     PART
LEVELFILT   4
```

5.16.4.7 Solver Options File

One other important aspect regarding the options file involves the use of a problem solver options file when a solver such as SOPLEX, CBC, IPOPT etc. is also being used. As seen above the GAMSCHK.OPT does not recognize option commands such as those which would be submitted to the programming model solvers - SOPLEX for example. In all cases GAMSCHK will cause the default option file for the solver to be used when invoking the solver. Thus if SOPLEX and the options file is invoked is being used, SOPLEX options are controlled by the option file SOPLEX.OPT while GAMSCHK.OPT controls GAMSCHK operation. Users can change the number of the solver options file being used by using the OPTFILE parameter in the options file. OPTFILE 2 would cause use of solver options file .OP2.

5.16.5 Known Bugs

There are a few bugs that can cause GAMSCHK to report improper outputs or results. A list of the known bugs, their symptoms and a remedy is given below.

Symptom	Cause	Remedy
Zero Shadow Prices in POSTOPT	Old GAMS version or no Prior Solve	1) Make sure the model was solved, 2) if it was, do not suppress solve in option file, or 3) update to most recent GAMS version
Descaling Does Not Work	Old Version of GAMS	Update
GAMS Blows up after GAMSCHK Runs	Old GAMS version	Ignore, *.LST file, results are fine, can be fixed by updating to the most recent version of GAMS
POSTOPT has error in budgets equal to twice objective function coefficient for nonlinear maximizations	Old GAMS MINOS version	Switch to a minimization formulation or update GAMS/MINOS
ROWSUM does not fully account for the value of nonlinear terms in POSTOPT	Value of nonlinear terms sent from GAMS are only a marginal value	None planned. GAMSCHK would need reprogramming
Error message about size of VARBLOCK or EQNBLOCK	exceeded maximum number of blocks	Modify option file, enlarging or eliminating parameters
GAMSCHK won't run	Files are not properly installed	Recheck installation. If still doesn't work report to author
Zero shadow prices when using NOSOLVE	Old version of GAMS solvers or Shadow prices suppressed	Try changing GAMSCOMP.TXT lines 2 or 0 to 12 or 10, if that doesn't work update GAMS.

5.16.6 Tables

Table 1: Conditions under which a modeler should be advised of potential difficulty for equations without nonlinear terms.

- `a/` The PS cases indicate, because the variables in this equation follow this pattern, that:
  1. The variables appearing with nonzeros in this equation are forced to equal zero.
  2. This equation can never be satisfied and is obviously infeasible.
  3. This equation is redundant. The nonnegativity conditions are a stronger restriction.
- `b/` In the examples x denotes indexed non-negative variables, y indexed non-positive variables, and z a single unrestricted variable.
- `c/` Here and in the cases below at least one nonzero must occur.
- `d/` These entries give examples of the problem covered by each warning. Namely, in the first case examining only the nonnegative variables suppose all those variables have signs ≥ 0 but the right-hand-side is zero. Thus, we have $X \geq 0$ and $X \leq 0$ which implies $X = 0$. A warning is generated in that case.
- `e/` Only one coefficient is allowed.

Type of constraint	Count of coefficients under a variable of this type with a particular sign						Sign of RHS	Type of PS[a/]	Examples[b]
	Nonnegative		Nonpositive		Unrestricted				
	+	-	+	-	+	-			
≤	≥ 0 [c/]	0	0	≥ 0	0	0	0	Zero Variables - Case 1	$\sum x \leq 0^{d/},\ -\sum y \leq 0,$
	≥ 0	0	0	≥ 0	0	0	-	Infeasible -Case 2	$\sum x \leq -k,\ -\sum y \leq -k,$
	0	≥ 0	≥ 0	0	0	0	+ or 0	Redundant -Case 3	$-\sum x \leq +k,\ \sum y \leq +k,$
=	≥0	0	0	≥0	0	0	0	Zero Variables - Case 1	$\sum x = 0,\ -\sum y = 0,\ \sum$
	0	≥0	≥0	0	0	0	0	Zero Variables - Case 1	$-\sum x = 0,\ \sum y = 0,\ -$
	≥0	0	0	≥0	0	0	-	Infeasible - Case 2	$\sum x - \sum y =$
	0	≥0	≥0	0	0	0	+	Infeasible - Case 2	$-\sum x + \sum y$
	0	0	0	0	≥$0^{e/}$	≥$0^{e/}$	0	Zero Variable - Case 1	$z = 0,\ -z =$
≥	0	≥0	≥0	0	0	0	0	Zero Variables - Case 1	$-\sum x \geq 0,\ \sum y \geq 0,\ -$
	0	≥0	≥0	0	0	0	0 or +	Infeasible -Case 2	$-\sum x \geq k,\ \sum y \geq k,\ -$
	≥0	0	0	≥0	0	0	- or 0	Redundant -Case 3	$\sum x \geq -k,\ -\sum y \geq -k,$

Table 2: Conditions under which a modeler should be warned about variables in a maximization problem.

Type of Variable	Objective function coefficient sign	Number of a_{ij}'s of a sign in ≥ rows +	≥ rows -	= rows +	= rows -	≤ rows +	≤ rows -	PS[a/]	Example
Nonnegative	+	≥0	0	0	0	0	≥0	Unbounded Variable case 1	$max\ x^{b/}$ $x+DQ\geq$ $-x+EQ\geq$
	-	0	≥0	0	0	≥0	0	Zero optimal solution case 2	$max\ -x$ $-x+DQ\geq$ $x+EQ\leq$
	0	≥0	0	0	0	0	≥0	Variable Relaxes constraint case 3	$max\ ox$ $x+DQ\geq$ $-x+DQ\geq$
	0	≥0	0	≥0[c/]	≥0[c/]	0	≥0	Variable Relaxes constraint case 4	$max\ ox$ $x+DQ\geq$ $x+FQ=$ $-x+EQ\geq$
Nonpositive	-	0	≥0	0	0	≥0	0	Unbounded Variable case 1	$max\ -y$ $-y+DQ\geq$ $y+EQ\leq$
	+	≥0	0	0	0	0	≥0	Zero optimal solution case 2	$max\ y^{b/}$ $y+DQ>$ $-y+EQ\geq$
	0	0	≥0	0	0	≥0	0	Variable Relaxes constraint case 3	$max\ ox$ $-y+DQ\geq$ $y+EQ\leq$
	0	≥0	0	≥0[c/]	≥0[c/]	0	≥0	Variable Relaxes constraint case 4	$max\ ox$ $-y+DQ\geq$ $y+FQ=$ $y+EQ\leq$
Unrestricted	+/-	0	0	0	0	0	0	Unbounded Variable case 1	$max\ \pm x$

- **a/** PS cases are: The variables which satisfy this condition are:
  1. Unbounded as they contribute to the objective function while satisfying the constraints.
  2. Obviously zero since they consume constraint resources and have a cost in the objective function.
  3. Warning this variable relaxes all constraints in which it appears
  4. Warning this variable relaxes all the equality constraints in which it appears in one direction
- **b/** Here x(y) has a positive objective term and can be increased without ever violating any constraints so x(y) is unbounded.
- **c/** Only one coefficient can be present in the equality rows

Table 3: Conditions When Model Elements Could be Unbounded or Infeasible.

Conditions for Potential Unbounded Variables – Presence of Bounds

Types of Variables	Sign of Objective in Max Problem	Upper	Lower
$\geq 0^{a/}$	+	None	—$^{b/}$
≤ 0	-	—	None
Unrestricted	+	None	—
Unrestricted	—	—	None

- **a/** If a non negative variable has a positive objective function coefficient without an upper bound, then the variable could be unbounded.
- **b/** Any reasonable value can exist for this item

Conditions for Potential Infeasibility Caused by Bounds on Variables

	Existence of Bounds	
Types of Variables	Lower	Upper
$\geq^{c/} 0$	+	—
≤ 0	—	—
Unrestricted	+	—
Unrestricted	—	—

- **c/** If a nonnegative variable has a positive lower bound then it could cause infeasibility.

Conditions for Potential Infeasibility in Equations

Type of Equations	RHS
$\leq^{d/}$	-
$\geq$	+
$=$	+ or -

- **d/** When a less than or equal equation is present it may not be able to be satisfied if it has a negative RHS.

Table 4: Conditions for Potential Infeasibility or Redundancy in Equations Based on Bounds on Variables.

	TYPE OF CONSTRAINT		PS
	≤b	≥b	
SUM OF THE SMALLEST VALUE[a/]	>b	—	INFEASIBLE
	—	>b	REDUNDANT
SUM OF THE LARGEST VALUE[b/]	—	<b	INFEASIBLE
	<b	—	REDUNDANT

Note:

- **a/** Suppose X_j is bounded with LB_j (lower bound) $\leq X_j \leq UB_j$ (upper bound), and we have the sum evaluated at the lower bounds will be the smallest value which could happen in that sum. If the constraint is $< b$, then if the sum is $> b$, we know that this constraint will never be satisfied. If the constraint is $> b$, and the sum is $> b$, we know that this constraint will not limit any possible X value. Hence, it is redundant.
- **b/** Suppose X_j is bounded as follows, LB_j (lower bound) $\leq X_j \leq UB_j$ (upper bound), and we have the sum evaluated at the upper bounds which is either $> b$ or $< b$, in that sum. If the sum is $< b$, and the constraint holds it $< b$ then we know that this constraint will not limit any possible X value. Hence, it is redundant. If the constraint holds it greater than b, but the sum is $< b$, we know that this constraint will never be satisfied.
- **c/** Thanks to Paul Preckel for bringing these tests to the authors' attention.

5.16.7 Appendix A: Reserved Names

```
VARIABLE
VARIABLES
EQUATION
EQUATIONS
INVARIABLE
INVARIABLES
INEQUATION
INEQUATIONS
LISTVARIABLE
LISTVARIABLES
LISTEQUATION
LISTEQUATIONS
POSTOPT
DISPLAYCR
PICTURE
BLOCKPIC
ANALYSIS
MATCHIT
BLOCKLIST
NONOPT
INSOLUTION
NOTINSOLUTION
NONINSOLUTON
VERBOSE
ADVISORY
BINDING
NONBINDINGdo
NOTBINDING
INTERSECT
IDENTIFY
PW=
```

5.16.8 Appendix B: GAMSCHK One Page Summary

Invoking GAMSCHK **OPTION LP=GAMSCHK** **Keywords allowed in GCK file**

Keyword	Allowed SubKEYWORDS	Brief Description
DISPLAYCR		Displays coefficients of selected variables and equations
	VARIABLE*	Indicates variable selections follow
	INVARIABLE*	Indicates equations are wanted in which selected variables fall
	EQUATION*	Indicates equation selections follow
	INEQUATION*	Indicates variables are wanted that fall in selected equations
	INTERSECT++	Show coefficients which appear at intersections of selected var/eqn
MATCHIT		List variable and equation names and summarize characteristics
	VARIABLE*	Summarizes all variables matching selection statements
	LISTVARIABLE*	Lists each variable matching a selection statement
	EQUATION*	Summarizes all equations matching a selection statement
	LISTEQUATION*	Lists each equation matching a selection statement
ANALYSIS		Checks for obvious structural defects
BLOCKLIST		Summarizes characteristics of variable and equation blocks
BLOCKPIC		Generates block level schematics
PICTURE		Generates tableau schematics
	VARIABLE*	Indicates variable selections follow
	INVARIABLE*	Indicates equations are wanted in which selected variables fall
	EQUATION*	Indicates equation selections follow
	INEQUATION*	Indicates variables are wanted that fall in selected equations
POSTOPT		Reconstructs reduced cost and equation activity
	VARIABLE*	Indicates variable selections follow
	INVARIABLE*	Indicates variable selections follow
	EQUATION*	Indicates equation selections follow
	INEQUATION*	Indicates variables are wanted that fall in selected equations
	INTERSECT++	Show coefficients which appear at intersections of selected var/eqn
	NOTINSOLUTION++	Only nonzero vars or those with zero reduced cost
	INSOLUTION++	Only zero vars will be selected
	BINDING++	Only eqns with zero slack will be computed
	NONBINDING++	Only eqns with nonzero slack will be computed
ADVISORY		List potential infeasible and unbounded items
NONOPT		Lists potential or actual nonoptimal items
	IDENTIFY	Same as ADVISORY but after solution
	VERBOSE	Does POSTOPT computations on nonoptimals

Other Notes

- Items marked above with an $*$ are followed by item selection statements.
- Items marked with ++ modifiy the types of variables, equations and coefficients selected.
- In item selection an $*$ is a wild card for multiple characters while a . is a wildcard for one character.
- Spaces and capitalization don't matter in any of the input.
- Options file controls scaling, solver choice, nonopt filters and maximum allowed selections.
- Page width is controlled by a PW= keyword but cannot exceed GAMS page width.
- Lines beginning with a ? or a # are treated as comments.

5.16.9 Appendix C: Summary of GAMSCHK Options

Option	Description	Default
CNS	solver for CNS problems	
DESCALE	controls treatment of scaling	part
DNLP	solver for DNLP problems	
EQUBLOCK	maximum number of equation blocks allowed Range: {-∞, ..., ∞}	-5
LEVELFILT	numerical value of exponent on "unbounded levels" Range: {-5, ..., ∞}	6
LP	solver for LP problems	
MARGFILT	numerical value of exponent on "infeasible marginals" Range: {-5, ..., ∞}	6
MCP	solver for MCP problems	
MINLP	solver for MINLP problems	
MIP	solver for MIP problems	
MPEC	solver for MPEC problems	
NLP	solver for NLP problems	
NOSOLVE	suppresses solution of the problem	
OPTFILE	solver options file number	
RMINLP	solver for RMINLP problems	
RMIP	solver for RMIP problems	
RMPEC	solver for RMPEC problems	
SOLVE	forces solution of the problem	
SOLVERNAME	solver for any problems	
VARBLOCK	maximum number of variable blocks allowed Range: {-∞, ..., ∞}	-5

5.16.10 GAMSCHK References

- Brooke, A., D. Kendrick, and A. Meeraus. GAMS: A User's Guide. The Scientific Press, South San Francisco, CA, 1988.
- McCarl, B.A. "So Your GAMS Model Didn't Work Right: A Guide to Model Repair." Texas A&M University, College Station, TX, 1994.
- McCarl, B.A., and T.H. Spreen. "Applied Mathematical Programming Using Algebraic Systems." Draft Book, Department of Agricultural Economics, Texas A&M University, College Station, TX, 1996.
- McCarl, B.A. GAMSCHK. Slides of INFORMS talk, 1996.
- McCarl, B.A. GAMSCHK. Older version of GAMSCHK User Documentation with additional GAMS examples.

5.17 Gurobi

Gurobi Optimization, [www.gurobi.com] (http://www.gurobi.com)

5.17.1 Introduction

The Gurobi suite of optimization products include state-of-the-art simplex and parallel barrier solvers for linear programming (LP) and quadratic programming (QP), parallel barrier solver for quadratically constrained programming (QCP), as well as parallel mixed-integer linear programming (MILP), mixed-integer quadratic programming (MIQP), mixed-integer quadratically constrained programming (MIQCP) and (mixed-integer) nonlinear programming (NLP) solvers.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

While numerous solving options are available, Gurobi automatically calculates and sets most options at the best values for specific problems. All Gurobi options available through GAMS/Gurobi are summarized at the end of this chapter.

We offer a GAMS/Gurobi-Link license that works in combination with a Gurobi callable library license from Gurobi Optimization Inc.

Attention

> The free bare-bone link mode (previously GAMS/OSIGUROBI) that allowed to solve LP and MIP when the user had a separate GUROBI license has been removed. If you relied on using this bare-bone link option, then do not hesitate to contact sales@gams.com to arrange for a GAMS/Gurobi-Link license.

5.17.2 How to Run a Model with Gurobi

The following statement can be used inside your GAMS program to specify using Gurobi

```
Option LP = Gurobi;  { or MIP or RMIP or QCP or MIQCP or RMIQCP }
```

The above statement should appear before the `solve` statement. If Gurobi was specified as the default solver during GAMS installation, the above statement is not necessary.

5.17.3 Overview of GAMS/Gurobi

5.17.3.1 Linear, Quadratic and Quadratic Constrained Programming

Gurobi can solve LP and convex QP problems using several alternative algorithms, while the only choice for solving convex QCP is the parallel barrier algorithm. The majority of LP problems solve best using Gurobi's state-of-the-art dual simplex algorithm, while most convex QP problems solve best using the parallel barrier algorithm. Certain types of LP problems benefit from using the parallel barrier or the primal simplex algorithms, while for some types of QP, the dual or primal simplex algorithm can be a better choice. If you are solving LP problems on a multi-core system, you should also consider using the concurrent optimizer. It runs different optimization algorithms on different cores, and returns when the first one finishes.

GAMS/Gurobi also provides access to the Gurobi infeasibility finder. The infeasibility finder takes an infeasible linear program and produces an irreducibly inconsistent set of constraints (IIS). An IIS is a set of constraints and variable bounds which is infeasible but becomes feasible if any one member of the set is dropped. GAMS/Gurobi reports the IIS in terms of GAMS equation and variable names and includes the IIS report as part of the normal solution listing. The infeasibility finder is activated by the option IIS. Another option for analyzing infeasible model the FeasOpt option which instructs GAMS/Gurobi to find a minimal feasible relaxation of an infeasible model. See section Feasible Relaxation for details.

GAMS/Gurobi supports sensitivity analysis (post-optimality analysis) for linear programs which allows one to find out more about an optimal solution for a problem. In particular, objective ranging and constraint ranging give information about how much an objective coefficient or a right-hand-side and variable bounds can change without changing the optimal basis. In other words, they give information about how sensitive the optimal basis is to a change in the objective function or the bounds and right-hand side. GAMS/Gurobi reports the sensitivity information as part of the normal solution listing. Sensitivity analysis is activated by the option Sensitivity.

The Gurobi presolve can sometimes diagnose a problem as being infeasible *or* unbounded. When this happens, GAMS/Gurobi can, in order to get better diagnostic information, rerun the problem with presolve turned off. The rerun without presolve is controlled by the option ReRun. In default mode only problems that are small (i.e. demo sized) will be rerun.

Gurobi can either presolve a model or start from an advanced basis or primal/dual solution pair. Often the solve from scratch of a presolved model outperforms a solve from an unpresolved model started from an advanced basis/solution. It is impossible to determine a priori if presolve or starting from a given advanced basis/solution without presolve will be faster. By default, GAMS/Gurobi will automatically use an advanced basis or solution from a previous `solve` statement. The GAMS BRatio option can be used to specify when not to use an advanced basis/solution. The GAMS/Gurobi option UseBasis can be used to ignore or force a basis/solution passed on by GAMS (it overrides BRatio). In case of multiple solves in a row and slow performance of the second and subsequent solves, the user is advised to set the GAMS BRatio option to 1.

5.17.3.2 Mixed-Integer Programming

The methods used to solve pure integer and mixed integer programming problems require dramatically more mathematical computation than those for similarly sized pure linear or quadratic programs. Many relatively small integer programming models take enormous amounts of time to solve.

For problems with discrete variables, Gurobi uses a branch and cut algorithm which solves a series of subproblems, LP subproblems for MILP, QP subproblems for MIQP, and QCP subproblems or LP outer approximation subproblems for MIQCP. Because a single mixed integer problem generates many subproblems, even small mixed integer problems can be very compute intensive and require significant amounts of physical memory. With option nonConvex Gurobi can also solve nonconvex (MI)QP and (MI)QCP problems using a spatial branch-and-bound method.

GAMS/Gurobi supports Special Order Sets of type 1 and type 2 as well as semi-continuous and semi-integer variables.

You can provide a known solution (for example, from a MIP problem previously solved or from your knowledge of the problem) to serve as the first integer solution.

If you specify some or all values for the discrete variables together with GAMS/Gurobi option MipStart, Gurobi will check the validity of the values as an integer-feasible solution. If this process succeeds, the solution will be treated as an integer solution of the current problem.

The Gurobi MIP solver includes shared memory parallelism, capable of simultaneously exploiting any number of processors and cores per processor. The implementation is deterministic: two separate runs on the same model will produce identical solution paths.

5.17.3.3 Nonlinear Programming

Gurobi can solve (mixed-integer) nonlinear programs to global optimality either directly or by approximating the model by piecewise-linear functions and/or reformulating nonlinear constraints into supported linear and/or quadratic constraints (see also funcnonlinear.)

For Gurobi to accept a nonlinear constraint, it has to be in one of the forms listed below. Furthermore, GAMS/Gurobi can automatically reformulate a nonlinear constraint into the supported form by enabling nlreform.

- **MAX constraint:**

```
eq1.. r =e= max(x1,x2,x3,...,c);
eq2.. r =e= smax(i, x(i));
```

- **MIN constraint:**

```
eq1.. r =e= min(x1,x2,x3,...,c);
eq2.. r =e= smin(i, x(i));
```

- **AND constraint:**

```
eq1.. r =e= b1 and b2 and b3 and ...;
eq2.. r =e= sand(i, b(i));
```

- **OR constraint:**

```
eq1.. r =e= b1 or b2 or b3 or ...;
eq2.. r =e= sor(i, b(i));
```

- **ABS constraint:**

```
eq.. r =e= abs(x);
```

- **EXP constraint:**

```
eq1.. r =e= exp(x);
eq2.. r =e= x**a;
eq3.. r =e= a**x;
```

  Here, `a` $>$ `0` and for `eq3` the lower bound of `x` must be nonnegative.

- **LOG constraint:**

```
eq1.. r =e= log(x);
eq2.. r =e= log2(x);
eq3.. r =e= log10(x);
```

- **SIN / COS / TAN constraint:**

```
eq1.. r =e= sin(x);
eq2.. r =e= cos(x);
eq3.. r =e= tan(x);
```

- **NORM constraint:**

```
eq1_1.. r =e= sum(i, abs(x(i)));
eq1_2.. r =e= abs(x1) + abs(x2) + abs(x3) + abs(x4);

eq2_1.. r =e= edist(x1,x2,x3,...);
eq2_2.. r =e= sqrt(sum(i, sqr(x(i))));
eq2_3.. r =e= sqrt(sqr(x1) + sqr(x2) + sqr(x3) + sqr(x4));

eq3_1.. r =e= smax(i, abs(x(i)));
eq3_2.. r =e= max(abs(x1), abs(x2), abs(x3), abs(x4));
```

  Note that a 2-norm constraint can lead to a non-convex quadratic model which is much harder to solve than a convex quadratic or linear model.

- **POLY constraint:**

```
eq.. r =e= poly(x, a0, a1, a2, ...);
```

- **SIGMOID constraint:**

```
eq1.. r =e= sigmoid(x);
eq2.. r =e= 1 / (1 + exp(-x));
```

Note

If nlreform is disabled, nonlinear constraints must perfectly match one of the above forms. The model is then directly passed to Gurobi without reformulation. If the constraint doesn't match one of the above forms, GAMS/Gurobi will raise a capability error. For example, it is not possible to interchange left-hand-side and right-hand-side of the above constraints. For more flexibility, enable nlreform.

Attention

When reformulating the model using nlreform, the nonlinear constraint is split into multiple smaller constraints. Therefore, the termination tolerances, in particular feasibilitytol, are applied differently. As a result, the solution returned to GAMS may not satisfy the requested tolerances.

5.17.3.4 Feasible Relaxation

The Infeasibility Finder identifies the causes of infeasibility by means of inconsistent set of constraints (IIS). However, you may want to go beyond diagnosis to perform automatic correction of your model and then proceed with delivering a solution. One approach for doing so is to build your model with explicit slack variables and other modeling constructs, so that an infeasible outcome is never a possibility. An automated approach offered in GAMS/Gurobi is known as FeasOpt (for Feasible Optimization) and turned on by parameter FeasOpt in a GAMS/Gurobi option file.

With the FeasOpt option GAMS/Gurobi accepts an infeasible model and selectively relaxes the bounds and constraints in a way that minimizes a weighted penalty function. In essence, the feasible relaxation tries to suggest the least change that would achieve feasibility. It returns an infeasible solution to GAMS and marks the relaxations of bounds and constraints with the INFES marker in the solution section of the listing file.

By default all equations are candidates for relaxation and weighted equally but none of the variables can be relaxed. This default behavior can be modified by assigning relaxation preferences to variable bounds and constraints. These preferences can be conveniently specified with the `.feaspref` option. The input value denotes the users willingness to relax a constraint or bound. The larger the preference, the more likely it will be that a given bound or constraint will be relaxed. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed. It is not necessary to specify a unique preference for each bound or range. In fact, it is conventional to use only the values 0 (zero) and 1 (one) except when your knowledge of the problem suggests assigning explicit preferences.

Preferences can be specified through a GAMS/Gurobi solver option file using dot options. The syntax is:

(variable or equation)`.feaspref`*(value)*

For example, suppose we have a GAMS declaration:

```
Set i /i1*i5/;
Set j /j2*j4/;
variable v(i,j); equation e(i,j);
```

Then, the relaxation preference in the *gurobi.opt* file can be specified by:

```
feasopt 1
v.feaspref            1
v.feaspref('i1',*)    2
v.feaspref('i1','j2') 0

e.feaspref(*,'j1')    0
e.feaspref('i5','j4') 2
```

First we turn the feasible relaxtion on. Futhermore, we specify that all variables `v(i,j)` have preference of 1, except variables over set element `i1`, which have a preference of 2. The variable over set element `i1` and `j2` has preference 0. Note that preferences are assigned in a procedural fashion so that preferences assigned later overwrite previous preferences. The same syntax applies for assigning preferences to equations as demonstrated above. If you want to assign a preference to all variables or equations in a model, use the keywords `variables` or `equations` instead of the individual variable and equations names (e.g. `variables.feaspref 1`).

The parameter FeasOptMode allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require only as much relaxation as it found necessary in the first phase. Values of the parameter FeasOptMode indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations). Please check description of parameter FeasOptMode for details. Also check example models `feasopt*` in the GAMS Model library.

5.17.3.5 Parameter Tuning Tool

The Gurobi Optimizer provides a wide variety of parameters that allow you to control the operation of the optimization engines. The level of control varies from extremely coarse-grained (e.g., the Method parameter, which allows you to choose the algorithm used to solve continuous models) to very fine-grained (e.g., the MarkowitzTol parameter, which allows you to adjust the precise tolerances used during simplex basis factorization). While these parameters provide a tremendous amount of user control, the immense space of possible options can present a significant challenge when you are searching for parameter settings that improve performance on a particular model. The purpose of the Gurobi tuning tool is to automate this search.

The Gurobi tuning tool performs multiple solves on your model, choosing different parameter settings for each, in a search for settings that improve runtime. The longer you let it run, the more likely it is to find a significant improvement.

A number of tuning-related parameters allow you to control the operation of the tuning tool. The most important is probably TuneTimeLimit, which controls the amount of time spent searching for an improving parameter set. Other parameters include TuneTrials (which attempts to limit the impact of randomness on the result), TuneResults (which limits the number of results that are returned), and TuneOutput (which controls the amount of output produced by the tool).

While parameter settings can have a big performance effect for many models, they aren't going to solve every performance issue. One reason is simply that there are many models for which even the best possible choice of parameter settings won't produce an acceptable result. Some models are simply too large and/or difficult to solve, while others may have numerical issues that can't be fixed with parameter changes.

Another limitation of automated tuning is that performance on a model can experience significant variations due to random effects (particularly for MIP models). This is the nature of search. The Gurobi algorithms often have to choose from among multiple, equally appealing alternatives. Seemingly innocuous changes to the model (such as changing the order of the constraint or variables), or subtle changes to the algorithm (such as modifying the random number seed) can lead to different choices. Often times, breaking a single tie in a different way can lead to an entirely different search. We've seen cases where subtle changes in the search produce 100X performance swings. While the tuning tool tries to limit the impact of these effects, the final result will typically still be heavily influenced by such issues.

The bottom line is that automated performance tuning is meant to give suggestions for parameters that could produce consistent, reliable improvements on your models. It is not meant to be a replacement for efficient modeling or careful performance testing.

5.17.3.6 Compute Server

The Gurobi Compute Server allows you to use one or more servers to offload all of your Gurobi computations.

Gurobi compute servers support queuing and load balancing. You can set a limit on the number of simultaneous jobs each compute server will run. When this limit has been reached, subsequent jobs will be queued. If you have multiple compute servers, the current job load is automatically balanced among the available servers. By default, the Gurobi job queue is serviced in a First-In, First-Out (FIFO) fashion. However, jobs can be given different priorities. Jobs with higher priorities are then selected from the queue before jobs with lower priorities.

Gurobi Compute Server licenses and software are not included in GAMS/Gurobi. Contact Gurobi directly to inquire about the software and license.

5.17.3.7 Distributed Parallel Algorithms

Gurobi Optimizer implements a number of distributed algorithms that allow you to use multiple machines to solve a problem faster. Available distributed algorithms are:

- A **distributed MIP solver**, which allows you to divide the work of solving a single MIP model among multiple machines. A manager machine passes problem data to a set of worker machines in order to coordinate the overall solution process.
- A **distributed concurrent solver**, which allows you to use multiple machines to solve an LP or MIP model. Unlike the distributed MIP solver, the concurrent solver doesn't divide the work associated with solving the problem among the machines. Instead, each machine uses a different strategy to solve the whole problem, with the hope that one strategy will be particularly effective and will finish much earlier than the others. For some problems, this concurrent approach can be more effective than attempting to divide up the work.
- **Distributed parameter tuning**, which automatically searches for parameter settings that improve performance on your optimization model. Tuning solves your model with a variety of parameter settings, measuring the performance obtained by each set, and then uses the results to identify the settings that produce the best overall performance. The distributed version of tuning performs these trials on multiple machines, which makes the overall tuning process run much faster.

These distributed parallel algorithms are designed to be almost entirely transparent to the user. The user simply modifies a few parameters, and the work of distributing the computation to multiple machines is handled behind the scenes by Gurobi.

Specifying the Worker Pool

Once you've set up a set of one or more distributed workers, you should list at least one of their names in the WorkerPool parameter. You can provide either machine names or IP addresses, and they should be comma-separated.

You can provide the worker access password through the WorkerPassword parameter. All servers in the worker pool must have the same access password.

Requesting Distributed Algorithms

Once you've set up the worker pool through the appropriate parameters, the last step to use a distributed algorithm is to set the TuneJobs, ConcurrentJobs, or DistributedMIPJobs parameter. These parameters are used to indicate how many distinct tuning, concurrent, or distributed MIP jobs should be started on the available workers.

If some of the workers in your worker pool are running at capacity when you launch a distributed algorithm, the algorithm won't create queued jobs. Instead, it will launch as many jobs as it can (up to the requested value), and it will run with these jobs.

These distributed algorithms have been designed to be nearly indistinguishable from the single machine versions. Our hope is that, if you know how to use the single machine version, you'll find it straightforward to use the distributed version. The distributed algorithms respect all of the usual parameters. For distributed MIP, you can adjust strategies, adjust tolerances, set limits, etc. For concurrent MIP, you can allow Gurobi to choose the settings for each machine automatically or specify a set of options. For distributed tuning, you can use the usual tuning parameters, including TuneTimeLimit, TuneTrails, and TuneOutput.

There are a few things to be aware of when using distributed algorithms, though. One relates to relative machine performance. Distributed algorithms work best if all of the workers give very similar performance. For example, if one machine in your worker pool were much slower than the others in a distributed tuning run, any parameter sets tested on the slower machine would appear to be less effective than if they were run on a faster machine. Similar considerations apply for distributed MIP and distributed concurrent. We strongly recommend that you use machines with very similar performance. Note that if your machines have similarly performing cores but different numbers of cores, we suggest that you use the Threads parameter to make sure that all machines use the same number of cores.

Logging for distributed MIP is very similar to the standard MIP logging. The main differences are in the progress section. The header for the standard MIP logging looks like this:

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
```

By contrast, the distributed MIP header looks like this:

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | ParUtil Time
```

Instead of showing iterations per node, the last field in the distributed log shows parallel utilization. Specifically, it shows the fraction of the preceding time period (the time since the previous progress log line) that the workers spent actively processing MIP nodes.

Here is an example of a distributed MIP progress log:

```
    Nodes    |    Utilization     |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | ParUtil Time

H    0                            157344.61033          -      -          0s
H    0                            40707.729144          -      -          0s
H    0                            28468.534497          -      -          0s
H    0                            18150.083886          -      -          0s
H    0                            14372.871258          -      -          0s
H    0                            13725.475382          -      -          0s
     0     0 10543.7611    0   19 13725.4754 10543.7611  23.2%   99%    0s
*  266                            12988.468031 10543.7611  18.8%          0s
H 1503                            12464.099984 10630.6187  14.7%          0s
* 2350                            12367.608657 10632.7061  14.0%          1s
* 3360                            12234.641804 10641.4586  13.0%          1s
H 3870                            11801.185729 10641.4586  9.83%          1s

Ramp-up phase complete - continuing with instance 2 (best bd 10661)

 16928  2731 10660.9626    0   12 11801.1857 10660.9635  9.66%   99%    2s
 135654 57117 11226.5449   19   12 11801.1857 11042.3036  6.43%   98%    5s
 388736 135228 11693.0268   23   12 11801.1857 11182.6300  5.24%   96%   10s
 705289 196412     cutoff           11801.1857 11248.8963  4.68%   98%   15s
 1065224 232839 11604.6587   28   10 11801.1857 11330.2111  3.99%   98%   20s
 1412054 238202 11453.2202   31   12 11801.1857 11389.7119  3.49%   99%   25s
 1782362 209060     cutoff           11801.1857 11437.2670  3.08%   97%   30s
 2097018 158137 11773.6235   20   11 11801.1857 11476.1690  2.75%   92%   35s
 2468495 11516     cutoff           11801.1857 11699.9393  0.86%   78%   40s
 2481830     0     cutoff           11801.1857 11801.1857  0.00%   54%   40s
```

One thing you may find in the progress section is that node counts may not increase monotonically. Distributed MIP tries to create a single, unified view of node numbers, but with multiple machines processing nodes independently, possibly at different rates, some inconsistencies are inevitable.

Another difference is the line that indicates that the distributed ramp-up phase is complete. At this point, the distributed strategy transitions from a concurrent approach to a distributed approach. The log line indicates which worker was the winner in the concurrent approach. Distributed MIP continues by dividing the partially explored MIP search tree from this worker among all of the workers.

Another difference in the distributed log is in the summary section. The distributed MIP log includes a breakdown of how runtime was spent:

```
Runtime breakdown:
  Active:  37.85s (93%)
  Sync:     2.43s (6%)
  Comm:     0.34s (1%)
```

This is an aggregated view of the utilization data that is displayed in the progress log lines. In this example, the workers spent 93% of runtime actively working on MIP nodes, 6% waiting to synchronize with other workers, and 1% communicating data between machines.

The installation instructions for the Gurobi Remote Services can be found on Gurobi's web page [www.gurobi.com] (http://www.gurobi.com).

Gurobi Instant Cloud

An alternative to setting up your own pool of machines is to use the Gurobi Instant Cloud. You only need a GAMS/Gurobi link license when you solve your problems in the Gurobi Instant Cloud. The cost for the Gurobi license is paid on a per use basis directly to Gurobi. If you follow through the steps on the Gurobi web site, you eventually get the names of the machines Gurobi has started for you in the cloud. In order to use these machines from GAMS/Gurobi, you need to provide a Gurobi license with access instructions for the Gurobi Instant Cloud (detailed instructions for configuring the client license file).

5.17.3.8 Solution Pool

While the default goal of the Gurobi Optimizer is to find one proven optimal solution to your model, with a possible side-effect of finding other solutions along the way, the solver provides a number of parameters that allow you to change this behavior.

By default, the Gurobi MIP solver will try to find one proven optimal solution to your model. It will typically find multiple sub-optimal solutions along the way, which can be retrieved later. However, these solutions aren't produced in a systematic way. The set of solutions that are found depends on the exact path the solver takes through the MIP search. You could solve a MIP model once, obtaining a set of interesting sub-optimal solutions, and then solve the same problem again with different parameter settings, and find only the optimal solution.

To store (some of the) solutions found along the way, you can enable the Solution Pool feature by setting option solnpool. If you'd like more control over how solutions are found and retained, the Gurobi Optimizer has a number of parameters available for this. The first and simplest is PoolSolutions, which controls the size of the solution pool. Changing this parameter won't affect the number of solutions that are found - it simply determines how many of those are retained.

You can use the PoolSearchMode parameter to control the approach used to find solutions. In its default setting (0), the MIP search simply aims to find one optimal solution. Setting the parameter to 1 causes the MIP search to expend additional effort to find more solutions, but in a non-systematic way. You will get more solutions, but not necessarily the best solutions. Setting the parameter to 2 causes the MIP to do a systematic search for the n best solutions. For both non-default settings, the PoolSolutions parameter sets the target for the number of solutions to find.

If you are only interested in solutions that are within a certain gap of the best solution found, you can set the PoolGap parameter. Solutions that are not within the specified gap are discarded.

Obtaining an OPTIMAL optimization return status when using PoolSearchMode=2 indicates that the MIP solver succeeded in finding the desired number of best solutions, or it proved that the model doesn't have that many distinct feasible solutions. If the solver terminated early (e.g., due to a time limit), you *PoolObjBound* attribute (printed to the log) to evaluate the quality of the solutions that were found. This attribute gives a bound on the objective of any solution that isn't already in the solution pool. The difference between this attribute and ObjBound is that the latter gives a bound on the objective for any solution, and which is often looser than PoolObjBound. The *PoolObjBound* attribute gives a bound on the objective of undiscovered solutions. Further tree exploration won't find better solutions. You can use this bound to get a count of how many of the n best solutions you found: any solutions whose objective values are at least as good as *PoolObjBound* are among the n best.

Solution Pool Example

Let's continue with a few examples of how these parameters would be used. Imagine that you are solving a MIP model with an optimal (minimization) objective of 100. Further imagine that, using default settings, the MIP solver finds four solutions to this model with objectives 100, 110, 120, and 130.

If you set the PoolSolutions parameter to 3 and solve the model again, the MIP solver would discard the worst solution and return with 3 solutions in the solution pool. If you instead set the PoolGap parameter to value 0.2, the MIP solver would discard any solutions whose objective value is worse than 120 (which would also leave 3 solutions in the solution pool).

If you set the PoolSearchMode parameter to 2 and the PoolSolutions parameter to 10, the MIP solver would attempt to find the 10 best solutions to the model. An OPTIMAL return status would indicate that either (i) it found the 10 best solutions, or (ii) it found all feasible solutions to the model, and there were fewer than 10. If you also set the PoolGap parameter to a value of 0.1, the MIP solver would try to find 10 solutions with objective no worse than 110. While this may appear equivalent to asking for 10 solutions and simply ignoring those with objective worse than 110, the solve will typically complete significantly faster with this parameter set, since the solver does not have to expend effort looking for solutions beyond the requested gap.

Solution Pool Subtleties

There are a few subtleties associated with finding multiple solutions that we'll cover now.

Continuous Variables

One subtlety arises when considering multiple solutions for models with continuous variables. Specifically, you may have two solutions that take identical values on the integer variables but where some continuous variables differ. By choosing different points on the line between these two solutions, you actually have an infinite number of choices for feasible solutions to the problem. To avoid this issue, we define two solutions as being equivalent if they take the same values on all integer variables (and on all continuous variables that participate in SOS constraints). A solution will be discarded if it is equivalent to another solution that is already in the pool.

Optimality Gap

The interplay between the optimality gap (MIPGap or MIPGapAbs) and multiple solutions can be a bit subtle. When using the default PoolSearchMode, a non-zero optimality gap indicates that you are willing to allow the MIP solver to declare a solution optimal, even though the model may have other, better solutions. The claim the solver makes upon termination is that no other solution would improve the incumbent objective by more than the optimality gap. Terminating at this point is ultimately a pragmatic choice - we'd probably rather have the true best solution, but the cost of reducing the optimality gap to zero can often be prohibitive.

This pragmatic choice can produce a bit of confusion when finding multiple optimal solutions. Specifically, if you ask for the n best solutions, the optimality gap plays a similar role as it does in the default case, but the implications may be a bit harder to understand. Specifically, a non-zero optimality gap means that you are willing to allow the solver to declare that it has found the n best solutions, even though there may be solutions that are better than those that were returned. The claim in this case is that any solution not among the reported n best would improve on the objective for the worst among the n best by less than the optimality gap.

If you want to avoid this source of potential confusion, you should set the optimality gap to 0 when using `PoolSearchMode=2`.

Logging

If you browse the log from a MIP solve with PoolSearchMode set to a non-default value, you may see the lower bound on the objective exceed the upper bound. This can't happen with the default `PoolSearchMode` - if you are only looking for one optimal solution, the search is done as soon as the lower bound reaches the upper bound. However, if you are looking for the n best solutions, you have to prove that the model has no solution better than the n-th best. The objective for that n-th solution could be much worse than that of the incumbent. In this situation, the log file will include a line of the form:

```
Optimal solution found at node 123 - now completing solution pool...
```

Distributed MIP

One limitation that we should point out related to multiple solutions is that the distributed MIP solver has not been extended to support non-default PoolSearchMode settings. Distributed MIP will typically produce many more feasible solutions than non-distributed MIP, but there's no way to ask it to find the n best solutions.

5.17.3.9 Multiple Objectives

While typical optimization models have a single objective function, real-world optimization problems often have multiple, competing objectives. For example, in a production planning model, you may want to both maximize profits and minimize late orders, or in a workforce scheduling application, you may want to both minimize the number of shifts that are short-staffed while also respecting worker's shift preferences.

The main challenge you face when working with multiple, competing objectives is deciding how to manage the tradeoffs between them. Gurobi provides tools that simplify the task: Gurobi allows you to blend multiple objectives, to treat them *hierarchically*, or to combine the two approaches. In a blended approach, you optimize a weighted combination of the individual objectives. In a hierarchical or *lexicographic* approach, you set a priority for each objective, and optimize in priority order. When optimizing for one objective, you only consider solutions that would not degrade the objective values of higher-priority objectives. Gurobi allows you to enter and manage your objectives, to provide weights for a blended approach, or to set priorities for a hierarchical approach. Gurobi will only solve multi-objective models with strictly linear objectives. Moreover, for continous models, Gurobi will report a primal only solution (not dual information).

Following the workforce application the specifications of the objectives would be done as follows:

```
equations defObj, defNumShifts, defSumPreferences;
variables obj, numShifts, sumPreferences;

defobj..            obj =e= numShifts - 1/100*sumPreferences;
defNumShifts..      numShifts =e= ...;
defSumPreferences.. sumPreferences =e= ...;

model workforce /all/;
solve workforce minimizing obj using mip;
```

With the default setting GUROBI will solve the blended objective. Using the parameter MultObj GUROBI will use a hierarchical approach. A hierarchical or lexicographic approach assigns a priority to each objective, and optimizes for the objectives in decreasing priority order. At each step, it finds the best solution for the current objective, but only from among those that would not degrade the solution quality for higher-priority objectives. The priority is specified by the absolute value of the objective coefficient in the blended objective function (`defObj`). In the example, the `numShifts` objective with coefficient 1 has higher priority than the `sumPreferences` objective with absolute objective coefficient 1/100. The sign of the objective coefficient determines the direction of the particular objective function. So here `numShifts` will be minimized (same direction as on the `solve` statement) while `sumPreferences` will be maximized. GAMS needs to identify the various objective functions, therefore the objective variables can only appear in the blended objective functions and in the particular objective defining equation.

By default, the hierarchical approach won't allow later objectives to degrade earlier objectives. This behavior can be relaxed through a pair of attributes: ObjNRelTol and ObjNAbsTol. By setting one of these for a particular objective, you can indicate that later objectives are allowed to degrade this objective by the specified relative or absolute amount, respectively. In our earlier example, if the optimal value for `numShifts` is 100, and if we set ObjNAbsTol for this objective to 20, then the second optimization step maximizing `sumPreferences` would find the best solution for the second objective from among all solutions with objective 120 or better for `numShifts`. Note that if you modify both tolerances, later optimizations would use the looser of the two values (i.e., the one that allows the larger degradation).

5.17.4 GAMS Options

The following GAMS options are used by GAMS/Gurobi:

Option BRatio = x;

> Determines whether or not to use an advanced basis. A value of 1.0 causes GAMS to instruct Gurobi not to use an advanced basis. A value of 0.0 causes GAMS to construct a basis from whatever information is available. The default value of 0.25 will nearly always cause GAMS to pass along an advanced basis if a solve statement has previously been executed. This GAMS option is overridden by the GAMS/Gurobi option UseBasis

Option IterLim = n;

> Sets the simplex iteration limit. Simplex algorithms will terminate and pass on the current solution to GAMS. For MIP problems, if the number of the cumulative simplex iterations exceeds the limit, Gurobi will terminate. This GAMS option is overridden by the GAMS/Gurobi option IterationLimit

Option NodLim = x;

> Maximum number of nodes to process for a MIP problem. This GAMS option is overridden by the GAMS/Gurobi option NodeLimit.

Option OptCA = x;

Absolute optimality criterion for a MIP problem. The OptCA option asks Gurobi to stop when

$$|BP - BF| < \text{OptCA}$$

where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. This GAMS option is overridden by the GAMS/Gurobi option MipGapAbs.

Option OptCR = x;

Relative optimality criterion for a MIP problem. Notice that Gurobi uses a different definition than GAMS normally uses. The OptCR option asks Gurobi to stop when

$$|BP - BF| < |BF| * \text{OptCR}$$

where BF is the objective function value of the current best integer solution while BP is the best possible integer solution. The GAMS definition is:

$$|BP - BF| < |BP| * \text{OptCR}$$

This GAMS option is overridden by the GAMS/Gurobi option MipGap.

Option ResLim = x;

Sets the time limit in seconds. The algorithm will terminate and pass on the current solution to GAMS. Gurobi measures time in wall time on all platforms. Some other GAMS solvers measure time in CPU time on some Unix systems. In case resLim assumes its default value (1e+10) Gurobi will use its own default (infinity). This GAMS option is overridden by the GAMS/Gurobi option TimeLimit.

Option SysOut = On;

Will echo Gurobi messages to the GAMS listing file. This option may be useful in case of a solver failure.

***ModelName*.Cutoff = x;**

Cutoff value. When the branch and bound search starts, the parts of the tree with an objective worse than x are deleted. This can sometimes speed up the initial phase of the branch and bound algorithm. This GAMS option is overridden by the GAMS/Gurobi option CutOff.

***ModelName*.OptFile = 1;**

Instructs GAMS/Gurobi to read the option file. The name of the option file is `gurobi.opt`.

***ModelName*.PriorOpt = 1;**

Instructs GAMS/Gurobi to use the priority branching information passed by GAMS through variable suffix values *variable*`.prior`.

5.17.5 Summary of GUROBI Options

5.17.5.1 Termination options

Option	Description	Default
bariterlimit	Barrier iteration limit	infinity
bestbdstop	Best objective bound to stop	maxdouble
bestobjstop	Best objective value to stop	mindouble
cutoff	Objective cutoff	maxdouble
iterationlimit	Simplex iteration limit	infinity
memlimit	Memory limit	maxdouble
nodelimit	MIP node limit	maxdouble
softmemlimit	Soft memory limit	maxdouble
solutionlimit	MIP feasible solution limit	maxint
timelimit	Time limit	GAMS reslim
worklimit	Work limit	maxdouble

5.17.5.2 Tolerances options

Option	Description	Default
barconvtol	Barrier convergence tolerance	1e-08
barqcpconvtol	Barrier QCP convergence tolerance	1e-06
feasibilitytol	Primal feasibility tolerance	1e-06
intfeastol	Integer feasibility tolerance	1e-05
markowitztol	Threshold pivoting tolerance	0.0078125
mipgap	Relative MIP optimality gap	GAMS optcr
mipgapabs	Absolute MIP optimality gap	GAMS optca
optimalitytol	Dual feasibility tolerance	1e-06
psdtol	Positive semi-definite tolerance	1e-06

5.17.5.3 Simplex options

Option	Description	Default
lpwarmstart	Warm start usage in simplex	1
networkalg	Network simplex algorithm	-1
normadjust	Simplex pricing norm	-1
perturbvalue	Simplex perturbation magnitude	0.0002
quad	Quad precision computation in simplex	-1
sifting	Sifting within dual simplex	-1

Option	Description	Default
siftmethod	LP method used to solve sifting sub-problems	-1
simplexpricing	Simplex variable pricing strategy	-1

5.17.5.4 Barrier options

Option	Description	Default
barcorrectors	Central correction limit	-1
barhomogeneous	Barrier homogeneous algorithm	-1
barorder	Barrier ordering algorithm	-1
crossover	Barrier crossover strategy	-1
crossoverbasis	Crossover initial basis construction strategy	-1
qcpdual	Compute dual variables for QCP models	1

5.17.5.5 Scaling options

Option	Description	Default
objscale	Objective scaling	0
scaleflag	Model scaling	-1

5.17.5.6 MIP options

Option	Description	Default
branchdir	Branch direction preference	0
concurrentjobs	Enables distributed concurrent solver	0
concurrentmethod	Chooses continuous solvers to run concurrently	-1
concurrentmip	Enables concurrent MIP solver	1
degenmoves	Degenerate simplex moves	-1
distributedmipjobs	Enables the distributed MIP solver	0
dumpbcsol	Dump incumbents to GDX files during branch-and-cut	
fixoptfile	Option file for fixed problem optimization	
heuristics	Turn MIP heuristics up or down	0.05
improvestartgap	Trigger solution improvement	0
improvestartnodes	Trigger solution improvement	maxdouble
improvestarttime	Trigger solution improvement	maxdouble
.lazy	Lazy constraints value	0

Option	Description	Default
lazyconstraints	Indicator to use lazy constraints	0
minrelnodes	Minimum relaxation heuristic control	-1
mipfocus	Set the focus of the MIP solver	0
mipstart	Use mip starting values	0
mipstopexpr	Stop expression for branch and bound	
miqcpmethod	Method used to solve MIQCP models	-1
multimipstart	Use multiple (partial) mipstarts provided via gdx files	
nlpheur	Controls the NLP heuristic for non-convex quadratic models	1
nodefiledir	Directory for MIP node files	.
nodefilestart	Memory threshold for writing MIP tree nodes to disk	maxdouble
nodemethod	Method used to solve MIP node relaxations	-1
nonconvex	Control how to deal with non-convex quadratic programs	-1
norelheurtime	Limits the amount of time (in seconds) spent in the NoRel heuristic	0
norelheurwork	Limits the amount of work performed by the NoRel heuristic	0
obbt	Controls aggressiveness of optimality-based bound tightening	-1
.partition	Variable partition value	0
partitionplace	Controls when the partition heuristic runs	15
.prior	Branching priorities	1
pumppasses	Feasibility pump heuristic control	-1
rins	RINS heuristic	-1
solfiles	Location to store intermediate solution files	
solnpool	Controls export of alternate MIP solutions	
solnpoolmerge	Controls export of alternate MIP solutions for merged GDX solution file	
solnpoolnumsym	Maximum number of variable symbols when writing merged GDX solution file	10
solnpoolprefix	First dimension of variables for merged GDX solution file or file name prefix for GDX solution files	soln
solvefixed	Indicator for solving the fixed problem for a MIP to get a dual solution	1
startnodelimit	Node limit for MIP start sub-MIP	-1
submipnodes	Nodes explored by sub-MIP heuristics	500
symmetry	Symmetry detection	-1
varbranch	Branch variable selection strategy	-1
zeroobjnodes	Zero objective heuristic control	-1

5.17.5.7 Presolve options

Option	Description	Default
aggfill	Allowed fill during presolve aggregation	-1
aggregate	Presolve aggregation control	1
dualreductions	Disables dual reductions in presolve	1
precrush	Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model	0
predeprow	Presolve dependent row reduction	-1
predual	Presolve dualization	-1
premiqcpform	Format of presolved MIQCP model	-1
prepasses	Presolve pass limit	-1
preqlinearize	Presolve Q matrix linearization	-1
Presolve	Presolve level	-1
presos1bigm	Controls largest coefficient in SOS1 reformulation	-1
presos1encoding	Controls SOS1 reformulation	-1
presos2bigm	Controls largest coefficient in SOS2 reformulation	-1
presos2encoding	Controls SOS2 reformulation	-1
presparsify	Presolve sparsify reduction	-1

5.17.5.8 Tuning options

Option	Description	Default
tunecleanup	Enables a tuning cleanup phase	0
tunecriterion	Specify tuning criterion	-1
tunedynamicjobs	Enables distributed tuning using a dynamic set of workers	0
tunejobs	Enables distributed tuning using a static set of workers	0
tunemetric	Metric to aggregate results into a single measure	-1
tuneoutput	Tuning output level	2
tuneresults	Number of improved parameter sets returned	-1
tunetargetmipgap	A target gap to be reached	0
tunetargettime	A target runtime in seconds to be reached	0.005
tunetimelimit	Time limit for tuning	maxdouble
tunetrials	Perform multiple runs on each parameter set to limit the effect of random noise	0

5.17.5.9 Multiple Solutions options

Option	Description	Default
poolgap	Relative gap for solutions in pool	maxdouble
poolgapabs	Absolute gap for solutions in pool	maxdouble
poolsearchmode	Choose the approach used to find additional solutions	0
poolsolutions	Number of solutions to keep in pool	10

5.17.5.10 MIP Cuts options

Option	Description	Default
bqpcuts	BQP cut generation	-1
cliquecuts	Clique cut generation	-1
covercuts	Cover cut generation	-1
cutaggpasses	Constraint aggregation passes performed during cut generation	-1
cutpasses	Root cutting plane pass limit	-1
cuts	Global cut generation control	-1
flowcovercuts	Flow cover cut generation	-1
flowpathcuts	Flow path cut generation	-1
gomorypasses	Root Gomory cut pass limit	-1
gubcovercuts	GUB cover cut generation	-1
impliedcuts	Implied bound cut generation	-1
infproofcuts	Infeasibility proof cut generation	-1
liftprojectcuts	Lift-and-project cut generation	-1
mipsepcuts	MIP separation cut generation	-1
mircuts	MIR cut generation	-1
mixingcuts	Mixing cut generation	-1
modkcuts	Mod-k cut generation	-1
networkcuts	Network cut generation	-1
projimpliedcuts	Projected implied bound cut generation	-1
psdcuts	PSD cut generation	-1
relaxliftcuts	Relax-and-lift cut generation	-1
rltcuts	RLT cut generation	-1
strongcgcuts	Strong-CG cut generation	-1
submipcuts	Sub-MIP cut generation	-1
zerohalfcuts	Zero-half cut generation	-1

5.17.5.11 Distributed algorithms options

Option	Description	Default
workerpassword	Password for distributed worker cluster	
workerpool	Distributed worker cluster	

5.17.5.12 Other options

Option	Description	Default
disconnected	Disconnected component strategy	-1
displayinterval	Frequency at which log lines are printed	5
.dofuncpieceerror	Error allowed for PWL translation of function constraints	1e-3
.dofuncpiecelength	Piece length for PWL translation of function constraints	1e-2
.dofuncpieceratio	Control whether to under- or over-estimate function values in PWL approximation	-1
.dofuncpieces	Sets strategy for PWL function approximation	0
feasopt	Computes a minimum-cost relaxation to make an infeasible model feasible	0
feasoptmode	Mode of FeasOpt	0
.feaspref	feasibility preference	1
feasrelaxbigm	Big-M value for feasibility relaxations	1e+06
freegamsmodel	Preserves memory by dumping the GAMS model instance representation temporarily to disk	0
funcmaxval	Maximum value for x and y variables in function constraints	1e+06
funcnonlinear	Controls whether general function constraints are treated as nonlinear functions or via PWL approximation	1
funcpieceerror	Error allowed for PWL translation of function constraint	0.001
funcpiecelength	Piece length for PWL translation of function constraint	0.01
funcpieceratio	Controls whether to under- or over-estimate function values in PWL approximation	-1
funcpieces	Sets strategy for PWL function approximation	0
iis	Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible	0
iismethod	IIS method	-1
integralityfocus	Set the integrality focus	0
kappa	Display approximate condition number estimates for the optimal simplex basis	0
kappaexact	Display exact condition number estimates for the optimal simplex basis	0
method	Algorithm used to solve continuous models	-1
miptrace	Filename of MIP trace file	

Option	Description	Default
miptracenode	Node interval when a trace record is written	100
miptracetime	Time interval when a trace record is written	1
multiobjmethod	Warm-start method to solve for subsequent objectives	-1
multiobjpre	Initial presolve on multi-objective models	-1
multobj	Controls the hierarchical optimization of multiple objectives	0
names	Indicator for loading names	1
nlreform	Reform nonlinear equations to Gurobi general constraints	1
numericfocus	Set the numerical focus	0
objnabstol	Allowable absolute degradation for objective	
objnreltol	Allowable relative degradation for objective	
printoptions	List values of all options to GAMS listing file	0
qextractalg	quadratic extraction algorithm in GAMS interface	0
readparams	Read Gurobi parameter file	
rerun	Resolve without presolve in case of unbounded or infeasible	-1
rngrestart	Write GAMS readable ranging information file	
seed	Modify the random number seed	0
sensitivity	Provide sensitivity information	0
solutiontarget	Specify the solution target for LP	-1
threads	Number of parallel threads to use	GAMS threads
Tuning	Parameter Tuning	
usebasis	Use basis from GAMS	GAMS bratio
varhint	Guide heuristics and branching through variable hints	0
writeparams	Write Gurobi parameter file	
writeprob	Save the problem instance	

5.17.5.13 The GAMS/Gurobi Options File

The GAMS/Gurobi options file consists of one option or comment per line. An asterisk (∗) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by any amount of white space (blanks or tabs).

Following is an example options file *gurobi.opt*.

```
simplexpricing 3
method 0
```

It will cause Gurobi to use quick-start steepest edge pricing and will use the primal simplex algorithm.

5.17.6 GAMS/Gurobi Log File

Gurobi reports its progress by writing to the GAMS log file as the problem solves. Normally the GAMS log file is directed to the computer screen.

The log file shows statistics about the presolve and continues with an iteration log.

For the simplex algorithms, each log line starts with the iteration number, followed by the objective value, the primal and dual infeasibility values, and the elapsed wall clock time. The dual simplex uses a bigM approach for handling infeasibility, so the objective and primal infeasibility values can both be very large during phase I. The frequency at which log lines are printed is controlled by the DisplayInterval option. By default, the simplex algorithms print a log line roughly every five seconds, although log lines can be delayed when solving models with particularly expensive iterations.

The simplex screen log has the following appearance:

```
Presolve removed 977 rows and 1539 columns
Presolve changed 3 inequalities to equalities
Presolve time: 0.078000 sec.
Presolved: 1748 Rows, 5030 Columns, 32973 Nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    3.8929476e+31   1.200000e+31   1.485042e-04      0s
    5624    1.1486966e+05   0.000000e+00   0.000000e+00      2s

Solved in 5624 iterations and 1.69 seconds
Optimal objective  1.148696610e+05
```

The barrier algorithm log file starts with barrier statistics about dense columns, free variables, nonzeros in AA' and the Cholesky factor matrix, computational operations needed for the factorization, memory estimate and time estimate per iteration. Then it outputs the progress of the barrier algorithm in iterations with the primal and dual objective values, the magnitude of the primal and dual infeasibilites and the magnitude of the complementarity violation. After the barrier algorithm terminates, by default, Gurobi will perform crossover to obtain a valid basic solution. It first prints the information about pushing the dual and primal superbasic variables to the bounds and then the information about the simplex progress until the completion of the optimization.

The barrier screen log has the following appearance:

```
Presolve removed 2394 rows and 3412 columns
Presolve time: 0.09s
Presolved: 3677 Rows, 8818 Columns, 30934 Nonzeros

Ordering time: 0.20s

Barrier statistics:
 Dense cols : 10
 Free vars  : 3
 AA' NZ     : 9.353e+04
 Factor NZ  : 1.139e+06 (roughly 14 MBytes of memory)
 Factor Ops : 7.388e+08 (roughly 2 seconds per iteration)

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0   1.11502515e+13 -3.03102251e+08  7.65e+05 9.29e+07  2.68e+09     2s
   1   4.40523949e+12 -8.22101865e+09  3.10e+05 4.82e+07  1.15e+09     3s
```

```
   2   1.18016996e+12 -2.25095257e+10  7.39e+04 1.15e+07  3.37e+08     4s
   3   2.24969338e+11 -2.09167762e+10  1.01e+04 2.16e+06  5.51e+07     5s
   4   4.63336675e+10 -1.44308755e+10  8.13e+02 4.30e+05  9.09e+06     6s
   5   1.25266057e+10 -4.06364070e+09  1.52e+02 8.13e+04  2.21e+06     7s
   6   1.53128732e+09 -1.27023188e+09  9.52e+00 1.61e+04  3.23e+05     9s
   7   5.70973983e+08 -8.11694302e+08  2.10e+00 5.99e+03  1.53e+05    10s
   8   2.91659869e+08 -4.77256823e+08  5.89e-01 5.96e-08  8.36e+04    11s
   9   1.22358325e+08 -1.30263121e+08  6.09e-02 7.36e-07  2.73e+04    12s
  10   6.47115867e+07 -4.50505785e+07  1.96e-02 1.43e-06  1.18e+04    13s
  ......
  26   1.12663966e+07  1.12663950e+07  1.85e-07 2.82e-06  1.74e-04     2s
  27   1.12663961e+07  1.12663960e+07  3.87e-08 2.02e-07  8.46e-06     2s

Barrier solved model in 27 iterations and 1.86 seconds
Optimal objective 1.12663961e+07

Crossover log...

    1592 DPushes remaining with DInf 0.0000000e+00                 2s
       0 DPushes remaining with DInf 2.8167333e-06                 2s

     180 PPushes remaining with PInf 0.0000000e+00                 2s
       0 PPushes remaining with PInf 0.0000000e+00                 2s

  Push phase complete: Pinf 0.0000000e+00, Dinf 2.8167333e-06      2s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
    1776    1.1266396e+07   0.000000e+00   0.000000e+00      2s

Solved in 2043 iterations and 2.00 seconds
Optimal objective  1.126639605e+07
```

For MIP problems, the Gurobi solver prints regular status information during the branch and bound search. The first two output columns in each log line show the number of nodes that have been explored so far in the search tree, followed by the number of nodes that remain unexplored. The next three columns provide information on the most recently explored node in the tree. The solver prints the relaxation objective value for this node, followed by its depth in the search tree, followed by the number of integer variables with fractional values in the node relaxation solution. The next three columns provide information on the progress of the global MIP bounds. They show the objective value for the best known integer feasible solution, the best bound on the value of the optimal solution, and the gap between these lower and upper bounds. Finally, the last two columns provide information on the amount of work performed so far. The first column gives the average number of simplex iterations per explored node, and the next column gives the elapsed wall clock time since the optimization began.

At the default value for option DisplayInterval, the MIP solver prints one log line roughly every five seconds. Note, however, that log lines are often delayed in the MIP solver due to particularly expensive nodes or heuristics.

```
Presolve removed 12 rows and 11 columns
Presolve tightened 70 bounds and modified 235 coefficients
Presolve time: 0.02s
Presolved: 114 Rows, 116 Columns, 424 Nonzeros
Objective GCD is 1

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
```

```
H    0     0                          -0.0000          -     -      -    0s
Root relaxation: 208 iterations, 0.00 seconds
     0     0    29.6862    0   64    -0.0000    29.6862     -      -    0s
H    0     0                           8.0000    29.6862   271%     -    0s
H    0     0                          17.0000    29.6862  74.6%     -    0s
     0     2    27.4079    0   60    17.0000    27.4079  61.2%     -    0s
H   27    17                          18.0000    26.0300  44.6%  51.6    0s
*   87    26              45          20.0000    26.0300  30.2%  28.4    0s
*  353    71              29          21.0000    25.0000  19.0%  19.3    0s
  1268   225    24.0000   28   43    21.0000    24.0000  14.3%  32.3    5s
  2215   464    22.0000   43   30    21.0000    24.0000  14.3%  33.2   10s

Cutting planes:
  Gomory: 175
  Cover: 25
  Implied bound: 87
  MIR: 150

Explored 2550 nodes (84600 simplex iterations) in 11.67 seconds
Thread count was 1 (of  4 available processors)

Optimal solution found (tolerance 1.00e-01)
Best objective 2.1000000000e+01, best bound 2.3000000000e+01, gap 9.5238%
```

5.17.7 Detailed Descriptions of GUROBI Options

aggfill *(integer)*: Allowed fill during presolve aggregation ↩

Controls the amount of fill allowed during presolve aggregation. Larger values generally lead to presolved models with fewer rows and columns, but with more constraint matrix non-zeros.

The default value chooses automatically, and usually works well.

Default: `-1`

aggregate *(integer)*: Presolve aggregation control ↩

Default: `1`

barconvtol *(real)*: Barrier convergence tolerance ↩

The barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance.

Default: `1e-08`

barcorrectors *(integer)*: Central correction limit ↩

The default value is chosen automatically, depending on problem characteristics.

Default: `-1`

barhomogeneous *(integer)*: Barrier homogeneous algorithm ↩

Determines whether to use the homogeneous barrier algorithm. At the default setting (-1), it is only used when barrier solves a node relaxation for a MIP model. Setting the parameter to 0 turns it off, and setting it to 1 forces it on. The homogeneous algorithm is useful for recognizing infeasibility or unboundedness. It is a bit slower than the default algorithm.

Default: `-1`

value	meaning
-1	Auto
0	Homogeneous Barrier off
1	Force Homogeneous Barrier on

bariterlimit *(integer)*: Barrier iteration limit ↩

Default: `infinity`

barorder *(integer)*: Barrier ordering algorithm ↩

Default: `-1`

value	meaning
-1	Auto
0	Approximate Minimum Degree ordering
1	Nested Dissection ordering

barqcpconvtol *(real)*: Barrier QCP convergence tolerance ↩

When solving a QCP model, the barrier solver terminates when the relative difference between the primal and dual objective values is less than the specified tolerance. Tightening this tolerance may lead to a more accurate solution, but it may also lead to a failure to converge.

Default: `1e-06`

bestbdstop *(real)*: Best objective bound to stop ↩

Terminates as soon as the engine determines that the best bound on the objective value is at least as good as the specified value.

Default: `maxdouble`

bestobjstop *(real)*: Best objective value to stop ↩

Terminate as soon as the engine finds a feasible solution whose objective value is at least as good as the specified value.

Default: `mindouble`

bqpcuts *(integer)*: BQP cut generation ↩

Default: `-1`

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

branchdir *(integer)*: Branch direction preference ↩

This option allows more control over how the branch-and-cut tree is explored. Specifically, when a node in the MIP search is completed and two child nodes, corresponding to the down branch and the up branch are created, this parameter allows you to determine whether the MIP solver will explore the down branch first, the up branch first, or whether it will choose the next node based on a heuristic determination of which sub-tree appears more promising.

Default: `0`

value	meaning
`-1`	Always explore the down branch first
`0`	Automatic
`1`	Always explore the up branch first

cliquecuts *(integer)*: Clique cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Conservative
`2`	Aggressive

concurrentjobs *(integer)*: Enables distributed concurrent solver ↩

Enables distributed concurrent optimization, which can be used to solve LP or MIP models on multiple machines. A value of n causes the solver to create n independent models, using different parameter settings for each. Each of these models is sent to a distributed worker for processing. Optimization terminates when the first solve completes. Use the WorkerPool parameter to provide a list of available distributed workers.

By default, Gurobi chooses the parameter settings used for each independent solve automatically. The intent of concurrent MIP solving is to introduce additional diversity into the MIP search. By bringing the resources of multiple machines to bear on a single model, this approach can sometimes solve models much faster than a single machine.

Default: `0`

concurrentmethod *(integer)*: Chooses continuous solvers to run concurrently ↩

This parameter is only evaluated when solving an LP with a concurrent solver (Method = 3 or 4). It controls which methods are run concurrently by the concurrent solver.

Default: `-1`

value	meaning
`-1`	Auto
`0`	barrier, dual, primal simplex
`1`	barrier and dual simplex
`2`	barrier and primal simplex
`3`	dual and primal simplex

concurrentmip *(integer)*: Enables concurrent MIP solver ↩

This parameter enables the concurrent MIP solver. When the parameter is set to value n, the MIP solver performs n independent MIP solves in parallel, with different parameter settings for each. Optimization terminates when the first solve completes. Gurobi chooses the parameter settings used for each independent solve automatically. The intent of concurrent MIP solving is to introduce additional diversity into the MIP search. This approach can sometimes solve models much faster than applying all available threads to a single MIP solve, especially on very large parallel machines.

The concurrent MIP solver divides available threads evenly among the independent solves. For example, if you have 6 threads available and you set ConcurrentMIP to 2, the concurrent MIP solver will allocate 3 threads to each independent solve. Note that the number of independent solves launched will not exceed the number of available threads.

The concurrent MIP solver produces a slightly different log from the standard MIP solver. The log only provides periodic summary information. Each concurrent MIP log line shows the objective for the best feasible solution found by any of the independent solves to that point, the best objective bound proved by any of the independent solves, and the relative gap between these two values. Gurobi also includes node counts from one of the independent solves, as well as elapsed times, to give some indication of forward progress.

Default: `1`

covercuts *(integer)*: Cover cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Conservative
`2`	Aggressive

crossover *(integer)*: Barrier crossover strategy ↩

Use value 0 to disable crossover; the solver will return an interior solution. Other options control whether the crossover algorithm tries to push primal or dual variables to bounds first, and then which simplex algorithm is used once variable pushing is complete. Options 1 and 2 push dual variables first, then primal variables. Option 1 finishes with primal, while option 2 finishes with dual. Options 3 and 4 push primal variables first, then dual variables. Option 3 finishes with primal, while option 4 finishes with dual The default value of -1 chooses automatically.

Default: `-1`

crossoverbasis *(integer)*: Crossover initial basis construction strategy ↩

Default: `-1`

value	meaning
`-1`	Auto
`0`	Chooses an initial basis quickly
`1`	Can take much longer, but often produces a more numerically stable start basis

cutaggpasses *(integer)*: Constraint aggregation passes performed during cut generation ↩

A non-negative value indicates the maximum number of constraint aggregation passes performed during cut generation. See the description of the global Cuts parameter for further information.

Default: `-1`

cutoff *(real)*: Objective cutoff ↩

Optimization will terminate if the engine determines that the optimal objective value for the model is worse than the specified cutoff. This option overwrites the GAMS cutoff option.

Default: `maxdouble`

cutpasses *(integer)*: Root cutting plane pass limit ↩

Default: `-1`

cuts *(integer)*: Global cut generation control ↩

The parameters, `Cuts`, CliqueCuts, CoverCuts, FlowCoverCuts, FlowPathCuts, GUBCoverCuts, ImpliedCuts, InfProofCuts, MIPSepCuts, MIRCuts, ModKCuts, NetworkCuts, GomoryPasses, StrongCGCuts, SubMIPCuts, CutAggPasses and ZeroHalfCuts, affect the generation of MIP cutting planes. In all cases except GomoryPasses and CutAggPasses, a value of -1 corresponds to an automatic setting, which allows the solver to determine the appropriate level of aggressiveness in the cut generation. Unless otherwise noted, settings of 0, 1, and 2 correspond to no cut generation, conservative cut generation, or aggressive cut generation, respectively. The `Cuts` parameter provides global cut control, affecting the generation of all cuts. This parameter also has a setting of 3, which corresponds to very aggressive cut generation. The other parameters override the global `Cuts` parameter (so setting `Cuts` to 2 and CliqueCuts to 0 would generate all cut types aggressively, except clique cuts which would not be generated at all. Setting `Cuts` to 0 and GomoryPasses to 10 would not generate any cuts except Gomory cuts for 10 passes).

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Conservative
`2`	Aggressive
`3`	Very aggressive

degenmoves *(integer)*: Degenerate simplex moves ↩

Limits degenerate simplex moves. These moves are performed to improve the integrality of the current relaxation solution. By default, the algorithm chooses the number of moves to perform automatically.

Changing the value of this parameter can help performance in cases where an excessive amount of time is spent after the initial root relaxation has been solved but before the cut generation process or the root heuristics have started.

Default: `-1`

disconnected *(integer)*: Disconnected component strategy ↩

A MIP model can sometimes be made up of multiple, completely independent sub-models. This parameter controls how aggressively we try to exploit this structure. A value of 0 ignores this structure entirely, while larger values try more aggressive approaches. The default value of -1 chooses automatically. This only affects mixed integer programming (MIP) models.

Default: `-1`

value	meaning
-1	Auto
0	Ignores structure entirely
1	Conservative
2	Aggressive

displayinterval *(integer)*: Frequency at which log lines are printed ↩

Default: `5`

distributedmipjobs *(integer)*: Enables the distributed MIP solver ↩

Enables distributed MIP. A value of n causes the MIP solver to divide the work of solving a MIP model among n machines. Use the WorkerPool parameter to provide the list of available machines.

Default: `0`

.dofuncpieceerror *(real)*: Error allowed for PWL translation of function constraints ↩

Default: `1e-3`

.dofuncpiecelength *(real)*: Piece length for PWL translation of function constraints ↩

The default length behavior for piecewise-linear approximation of a function constraint is controlled by funcPieceLength. This dot option `.doFuncPieceError` allows to overwrite the default behavior by constraint. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: `1e-2`

.dofuncpieceratio *(real)*: Control whether to under- or over-estimate function values in PWL approximation ↩

The default ratio behavior for piecewise-linear approximation of a function constraint is controlled by funcPieceRatio. This dot option `.doFuncPieceError` allows to overwrite the default behavior by constraint. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: `-1`

.dofuncpieces *(integer)*: Sets strategy for PWL function approximation ↩

The default strategy for performing a piecewise-linear approximation of a function constraint is set by funcPieces. This dot option `.doFuncPieces` allows to overwrite the default strategy by constraint. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

dofuncpieceerror The default error behavior for piecewise-linear approximation of a function constraint is controlled by funcPieceError. This dot option `.doFuncPieceError` allows to overwrite the default behavior by constraint. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: `0`

value	meaning
-2	Bounds the relative error of the approximation; the error bound is provided in the FuncPieceError parameter
-1	Bounds the absolute error of the approximation; the error bound is provided in the FuncPieceError parameter
0	Automatic based on relative error approach
1	Uses a fixed width for each piece; the actual width is provided in the FuncPiece-Length parameter
>=2	Sets the number of pieces; pieces are equal width

dualreductions *(boolean)*: Disables dual reductions in presolve ↩

Default: 1

dumpbcsol *(string)*: Dump incumbents to GDX files during branch-and-cut ↩

The content of this string option is used as a file stem for GDX point files. The file name gets completed by the solution number. So if this option has beed set to `mysol` then GDX files containing the new solution with the names `mysol0.gdx`, `mysol1.gdx`, ... will be created. The number of GDX files created depends on the number of solutions Gurobi finds during branch-and-cut.

feasibilitytol *(real)*: Primal feasibility tolerance ↩

All constrains must be satisfied to a tolerance of `FeasibilityTol`.

Range: [1e-09, 0.01]

Default: 1e-06

feasopt *(boolean)*: Computes a minimum-cost relaxation to make an infeasible model feasible ↩

With `Feasopt` turned on, a minimum-cost relaxation of the right hand side values of constraints or bounds on variables is computed in order to make an infeasible model feasible. It marks the relaxed right hand side values and bounds in the solution listing.

Several options are available for the metric used to determine what constitutes a minimum-cost relaxation which can be set by option FeasOptMode.

Feasible relaxations are available for all problem types.

Default: 0

value	meaning
0	Turns Feasible Relaxation off
1	Turns Feasible Relaxation on

feasoptmode *(integer)*: Mode of FeasOpt ↩

The parameter `FeasOptMode` allows different strategies in finding feasible relaxation in one or two phases. In its first phase, it attempts to minimize its relaxation of the infeasible model. That is, it attempts to find a feasible solution that requires minimal change. In its second phase, it finds an optimal solution (using the original objective) among those that require

only as much relaxation as it found necessary in the first phase. Values of the parameter `FeasOptMode` indicate two aspects: (1) whether to stop in phase one or continue to phase two and (2) how to measure the minimality of the relaxation (as a *sum* of required relaxations; as the *number* of constraints and bounds required to be relaxed; as a *sum of the squares* of required relaxations).

Default: `0`

value	meaning
`0`	Minimize sum of relaxations Minimize the sum of all required relaxations in first phase only
`1`	Minimize sum of relaxations and optimize Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations
`2`	Minimize number of relaxations Minimize the number of constraints and bounds requiring relaxation in first phase only
`3`	Minimize number of relaxations and optimize Minimize the number of constraints and bounds requiring relaxation in first phase and execute second phase to find optimum among minimal relaxations
`4`	Minimize sum of squares of relaxations Minimize the sum of squares of required relaxations in first phase only
`5`	Minimize sum of squares of relaxations and optimize Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations

.feaspref *(real)*: feasibility preference ↩

You can express the costs associated with relaxing a bound or right hand side value during a FeasOpt run through the `.feaspref` option. The syntax for dot options is explained in the Introduction chapter of the Solver Manual. The input value denotes the users willingness to relax a constraint or bound. More precisely, the reciprocal of the specified value is used to weight the relaxation of that constraint or bound. The user may specify a preference value less than or equal to 0 (zero), which denotes that the corresponding constraint or bound must not be relaxed.

Default: `1`

feasrelaxbigm *(real)*: Big-M value for feasibility relaxations ↩

When relaxing a constraint in a feasibility relaxation, it is sometimes necessary to introduce a big-M value. This parameter determines the default magnitude of that value.

Default: `1e+06`

fixoptfile *(string)*: Option file for fixed problem optimization ↩

flowcovercuts *(integer)*: Flow cover cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

flowpathcuts *(integer)*: Flow path cut generation ↩

See the description of the global Cuts parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

freegamsmodel *(boolean)*: Preserves memory by dumping the GAMS model instance representation temporarily to disk ↩

In order to provide the maximum amount of memory to the solver this option dumps the internal representation of the model instance temporarily to disk and frees memory. This option only works with SolveLink=0 and only for models without quadratic constraints.

Default: 0

funcmaxval *(real)*: Maximum value for x and y variables in function constraints ↩

Very large values in piecewise-linear approximations can cause numerical issues. This parameter limits the bounds on the variables that participate in function constraints. Specifically, if x or y participate in a function constraint, any bound larger than funcMaxVal (in absolute value) will be truncated.

Default: 1e+06

funcnonlinear *(boolean)*: Controls whether general function constraints are treated as nonlinear functions or via PWL approximation ↩

Default: 1

value	meaning
0	Piecewise-linear approximations
1	Nonlinear functions

funcpieceerror *(real)*: Error allowed for PWL translation of function constraint ↩

If the funcPieces parameter is set to value -1 or -2, this parameter provides the maximum allowed error (absolute for -1, relative for -2) in the piecewise-linear approximation.

Default: 0.001

funcpiecelength *(real)*: Piece length for PWL translation of function constraint ↩

If the funcPieces parameter is set to value 1, this parameter gives the length of each piece of the piecewise-linear approximation.

Default: `0.01`

funcpieceratio *(real)*: Controls whether to under- or over-estimate function values in PWL approximation ↩

This option controls whether the piecewise-linear approximation of a function constraint is an underestimate of the function, an overestimate, or somewhere in between. A value of 0 will always underestimate, while a value of 1 will always overestimate. A value in between will interpolate between the underestimate and the overestimate. A special value of -1 chooses points that are on the original function.

Default: `-1`

funcpieces *(integer)*: Sets strategy for PWL function approximation ↩

Default: `0`

value	meaning
`-2`	Bounds the relative error of the approximation; the error bound is provided in the FuncPieceError parameter
`-1`	Bounds the absolute error of the approximation; the error bound is provided in the FuncPieceError parameter
`0`	Automatic PWL approximation
`1`	Uses a fixed width for each piece; the actual width is provided in the FuncPiece-Length parameter
`>=2`	Sets the number of pieces; pieces are equal width

gomorypasses *(integer)*: Root Gomory cut pass limit ↩

A non-negative value indicates the maximum number of Gomory cut passes performed. See the description of the global Cuts parameter for further information.

Default: `-1`

gubcovercuts *(integer)*: GUB cover cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Conservative
`2`	Aggressive

heuristics *(real)*: Turn MIP heuristics up or down ↩

Larger values produce more and better feasible solutions, at a cost of slower progress in the best bound.

Range: [0, 1]

Default: 0.05

iis *(integer)*: Run the Irreducible Inconsistent Subsystem (IIS) finder if the problem is infeasible ↩

Default: 0

value	meaning
0	No conflict analysis
1	Conflict analysis after solve if infeasible
2	Conflict analysis without previous solve

iismethod *(integer)*: IIS method ↩

Chooses the IIS method to use. Method 0 is often faster, while method 1 can produce a smaller IIS. Method 2 ignores the bound constraints. Method 3 will return the IIS for the LP relaxation of a MIP model if the relaxation is infeasible, even though the result may not be minimal when integrality constraints are included. The default value of -1 chooses automatically.

Default: -1

impliedcuts *(integer)*: Implied bound cut generation ↩

See the description of the global Cuts parameter for further information.

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

improvestartgap *(real)*: Trigger solution improvement ↩

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify an optimality gap at which the MIP solver will switch to this strategy. For example, setting this parameter to 0.1 will cause the MIP solver to switch once the relative optimality gap is smaller than 0.1.

Default: 0

improvestartnodes *(real)*: Trigger solution improvement ↩

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify the node count at which the MIP solver switches to a solution improvement strategy. For example, setting this parameter to 10 will cause the MIP solver to switch strategies once the node count is larger than 10.

Default: maxdouble

improvestarttime *(real)*: Trigger solution improvement ↩

The MIP solver can change parameter settings in the middle of the search in order to adopt a strategy that gives up on moving the best bound and instead devotes all of its effort towards finding better feasible solutions. This parameter allows you to specify a time limit when the MIP solver will switch to this strategy. For example, setting this parameter to 10 will cause the MIP solver to switch 10 seconds after starting the optimization.

Default: `maxdouble`

infproofcuts *(integer)*: Infeasibility proof cut generation ↩

Controls infeasibility proof cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

Default: `-1`

integralityfocus *(boolean)*: Set the integrality focus ↩

One unfortunate reality in MIP is that integer variables don't always take exact integral values. While this typically doesn't create significant problems, in some situations the side-effects can be quite undesirable. The best-known example is probably a trickle flow, where a continuous variable that is meant to be zero when an associated binary variable is zero instead takes a non-trivial value. More precisely, given a constraint `y =l= Mb`, where y is a non-negative continuous variable, b is a binary variable, and M is a constant that captures the largest possible value of y, the constraint is intended to enforce the relationship that y must be zero if b is zero. With the default integer feasibility tolerance, the binary variable is allowed to take a value as large as *1e-5* while still being considered as taking value zero. If the M value is large, then the M b upper bound on the y variable can be substantial.

Reducing the value of the intFeasTol parameter can mitigate the effects of such trickle flows, but often at a significant cost, and often with limited success. The *integralityFocus* parameter provides a better alternative. Setting this parameter to 1 requests that the solver work harder to try to avoid solutions that exploit integrality tolerances. More precisely, the solver tries to find solutions that are still (nearly) feasible if all integer variables are rounded to exact integral values. We should say that the solver won't always succeed in finding such solutions, and that this setting introduces a modest performance penalty, but the setting will significantly reduce the frequency and magnitude of such violations.

Default: `0`

value	meaning
`0`	Disable the integrality focus
`1`	Enable the integrality focus

intfeastol *(real)*: Integer feasibility tolerance ↩

An integrality restriction on a variable is considered satisfied when the variable's value is less than `IntFeasTol` from the nearest integer value.

Range: [`1e-09`, `0.1`]

Default: `1e-05`

iterationlimit *(real)*: Simplex iteration limit ↩

Default: `infinity`

kappa *(boolean)*: Display approximate condition number estimates for the optimal simplex basis ↩

Default: `0`

value	meaning
`0`	Do not compute and display approximate condition number
`1`	Compute and display approximate condition number

kappaexact *(boolean)*: Display exact condition number estimates for the optimal simplex basis ↩

Default: `0`

value	meaning
`0`	Do not compute and display exact condition number
`1`	Compute and display exact condition number

.lazy *(integer)*: Lazy constraints value ↩

Determines whether a linear constraint is treated as a lazy constraint. At the beginning of the MIP solution process, any constraint whose Lazy attribute is set to 1, 2, or 3 (the default value is 0) is removed from the model and placed in the lazy constraint pool. Lazy constraints remain inactive until a feasible solution is found, at which point the solution is checked against the lazy constraint pool. If the solution violates any lazy constraint, the solution is discarded and one or more of the violated lazy constraints are pulled into the active model.

Larger values for this attribute cause the constraint to be pulled into the model more aggressively. With a value of 1, the constraint can be used to cut off a feasible solution, but it won't necessarily be pulled in if another lazy constraint also cuts off the solution. With a value of 2, all lazy constraints that are violated by a feasible solution will be pulled into the model. With a value of 3, lazy constraints that cut off the relaxation solution are also pulled in.

Any constraint whose Lazy attribute is set to -1 is treated as a user cut; it is removed from the model and placed in the user cut pool. User cuts may be added to the model at any node in the branch-and-cut search tree to cut off relaxation solutions.

The main difference between user cuts and lazy constraints is that the former are not allowed to cut off integer-feasible solutions. In other words, they are redundant for the MIP model, and the solver is free to decide whether or not to use them to cut off relaxation solutions. The hope is that adding them speeds up the overall solution process. Lazy constraints have no such restrictions. They are essential to the model, and the solver is forced to apply them whenever a solution would otherwise not satisfy them.

Only affects MIP models. Lazy constraints are only active if option LazyConstraints is enabled and are specified through the option `.lazy`. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: `0`

lazyconstraints *(boolean)*: Indicator to use lazy constraints ↩

Default: `0`

liftprojectcuts *(integer)*: Lift-and-project cut generation ↩

Controls lift-and-project cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

Default: `-1`

lpwarmstart *(integer)*: Warm start usage in simplex ↩

Controls whether and how Gurobi uses warm start information for an LP optimization. The non default setting of 2 is particularly useful for communicating advanced start information while retaining the performance benefits of presolve.

As a general rule, setting this parameter to 0 ignores any start information and solves the model from scratch. Setting it to 1 (the default) uses the provided warm start information to solve the original, unpresolved problem, regardless of whether presolve is enabled. Setting it to 2 uses the start information to solve the presolved problem, assuming that presolve is enabled. This involves mapping the solution of the original problem into an equivalent (or sometimes nearly equivalent) crushed solution of the presolved problem. If presolve is disabled, then setting 2 still prioritizes start vectors, while setting 1 prioritizes basis statuses. Taken together, the LPWarmStart parameter setting, the LP algorithm specified by Gurobi's Method parameter, and the available advanced start information determine whether Gurobi will use basis statuses only, basis statuses augmented with information from start vectors, or a basis obtained by applying the crossover method to the provided primal and dual start vectors to jump start the optimization.

When Gurobi's Method parameter requests the barrier solver, primal and dual start vectors are prioritized over basis statuses (but only if you provide both). These start vectors are fed to the crossover procedure. This is the same crossover that is used to compute a basic solution from the interior solution produced by the core barrier algorithm, but in this case crossover is started from arbitrary start vectors. If you set the LPWarmStart parameter to 1, crossover will be invoked on the original model using the provided vectors. Any provided basis information will not be used in this case. If you set LPWarmStart to 2, crossover will be invoked on the presolved model using crushed start vectors. If you set the parameter to 2 and provide a basis but no start vectors, the basis will be used to compute the corresponding primal and dual solutions on the original model. Those solutions will then be crushed and used as primal and dual start vectors for the crossover, which will then construct a basis for the presolved model. Note that for all of these settings and start combinations, no barrier algorithm iterations are performed.

The simplex algorithms provide more warm-starting options, with a parameter value of 1, simplex will start from a provided basis, if available. Otherwise, it uses a provided start vector to refine the crash basis it computes.

With a value of 2, simplex will use the crushed start vector on the presolved model to refine the crash basis. This is true regardless of whether the start is derived from start vectors or a starting basis from the original model. The difference is that if you provide an advanced basis, the basis will be used to compute the corresponding primal and dual solutions on the original model from which the primal or dual start on the presolved model will be derived.

Note: Only affects linear programming (LP) models.

Default: `1`

markowitztol *(real)*: Threshold pivoting tolerance ↩

Used to limit numerical error in the simplex algorithm. A larger value may avoid numerical problems in rare situations, but it will also harm performance.

Range: [`0.0001`, `0.999`]

Default: `0.0078125`

memlimit *(real)*: Memory limit ↩

Limits the total amount of memory (in GB, i.e., 10^9 bytes) available to Gurobi. If more is needed, Gurobi will fail with an OUT_OF_MEMORY error.

Default: `maxdouble`

method *(integer)*: Algorithm used to solve continuous models ↩

Synonyms: lpmethod rootmethod

Algorithm used to solve continuous models or the root node of a MIP model. Options are: -1=automatic, 0=primal simplex, 1=dual simplex, 2=barrier, 3=concurrent, 4=deterministic concurrent, 5=deterministic concurrent simplex.

In the current release, the default Automatic (-1) setting will typically choose non-deterministic concurrent (Method=3) for an LP, barrier (Method=2) for a QP or QCP, and dual (Method=1) for the MIP root node. Only the simplex and barrier algorithms are available for continuous QP models. Only primal and dual simplex are available for solving the root of an MIQP model. Only barrier is available for continuous QCP models.

Concurrent optimizers run multiple solvers on multiple threads simultaneously, and choose the one that finishes first. Method=3 and Method=4 will run dual simplex, barrier, and sometimes primal simplex (depending on the number of available threads). Method=5 will run both primal and dual simplex. The deterministic options (Method=4 and Method=5) give the exact same result each time, while Method=3 is often faster but can produce different optimal bases when run multiple times.

The default setting is rarely significantly slower than the best possible setting, so you generally won't see a big gain from changing this parameter. There are classes of models where one particular algorithm is consistently fastest, though, so you may want to experiment with different options when confronted with a particularly difficult model.

Note that if memory is tight on an LP model, you should consider using the dual simplex method (Method=1). The concurrent optimizer, which is typically chosen when using the default setting, consumes a lot more memory than dual simplex alone.

Default: `-1`

value	meaning
`-1`	Automatic
`0`	Primal simplex
`1`	Dual simplex
`2`	Barrier
`3`	Concurrent
`4`	Deterministic concurrent
`5`	Both primal and dual simplex

minrelnodes *(integer)*: Minimum relaxation heuristic control ↩

Number of nodes to explore in the minimum relaxation heuristic. Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a feasible solution.

This heuristic is quite expensive, and generally produces poor quality solutions. You should generally only use it if other means, including exploration of the tree with default settings, fail to produce a feasible solution.

The default value automatically chooses whether to apply the heuristic. It will only rarely choose to do so.

Default: `-1`

mipfocus *(integer)*: Set the focus of the MIP solver ↩

Default: `0`

value	meaning
0	Balance between finding good feasible solutions and proving optimality
1	Focus towards finding feasible solutions
2	Focus towards proving optimality
3	Focus on moving the best objective bound

mipgap *(real)*: Relative MIP optimality gap ↩

The MIP engine will terminate (with an optimal result) when the gap between the lower and upper objective bound is less than `MipGap` times the upper bound.

Range: [0, ∞]

Default: `GAMS optcr`

mipgapabs *(real)*: Absolute MIP optimality gap ↩

The MIP solver will terminate (with an optimal result) when the gap between the lower and upper objective bound is less than `MIPGapAbs`.

Range: [0, ∞]

Default: `GAMS optca`

mipsepcuts *(integer)*: MIP separation cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

mipstart *(boolean)*: Use mip starting values ↩

Default: `0`

value	meaning
0	Do not use the values
1	Use the values

mipstopexpr *(string)*: Stop expression for branch and bound ↩

If the provided logical expression is true, the branch-and-bound is aborted. Supported values are: resusd, nodusd, objest, objval. Supported opertators are: +, -, *, /, ^, %, !=, ==, <, <=, >, >=, !, &&, ||, (,), abs, ceil, exp, floor, log, log10, pow, sqrt. Example:
```
nodusd >= 1000 && abs(objest - objval) / abs(objval) < 0.1
```

If multiple stop expressions are given in an option file, the algorithm stops if any of them is true (|| concatenation).

miptrace *(string)*: Filename of MIP trace file ↩

More info is available in chapter Solve trace.

miptracenode *(integer)*: Node interval when a trace record is written ↩

Default: `100`

miptracetime *(real)*: Time interval when a trace record is written ↩

Default: `1`

miqcpmethod *(integer)*: Method used to solve MIQCP models ↩

Controls the method used to solve MIQCP models. Value 1 uses a linearized, outer-approximation approach, while value 0 solves continuous QCP relaxations at each node. The default setting (-1) chooses automatically.

Default: `-1`

value	meaning
-1	Auto
0	Continuous QCP relaxations at each node
1	Linearized, outer-approximation approach

mircuts *(integer)*: MIR cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

mixingcuts *(integer)*: Mixing cut generation ↩

Controls Mixing cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

Default: `-1`

modkcuts *(integer)*: Mod-k cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

multimipstart *(string)*: Use multiple (partial) mipstarts provided via gdx files ↩

Specifies (multiple) GDX files with values for the variables. Each file is treated as one intial guess for the MIP start. These MIP starts are added in addition to the initial guess provided by the level attribute. A MIP start GDX file can be created, for example, by using the command line option savepoint.

This option further allows to add partial MIP starts where some variable level records are not given. GUROBI will then try to construct a full MIP start out of it. In order to not provide a certain variable level record, simply not include that record in the GDX file or set the level value to NA or UNDEF.

This option requires mipstart enabled.

multiobjmethod *(integer)*: Warm-start method to solve for subsequent objectives ↩

When solving a continuous multi-objective model using a hierarchical approach, the model is solved once for each objective. The algorithm used to solve for the highest priority objective is controlled by the Method parameter. This parameter determines the algorithm used to solve for subsequent objectives. As with the Method parameters, values of 0 and 1 use primal and dual simplex, respectively. A value of 2 indicates that warm-start information from previous solves should be discarded, and the model should be solved from scratch (using the algorithm indicated by the Method parameter). The default setting of -1 usually chooses primal simplex.

Default: `-1`

multiobjpre *(integer)*: Initial presolve on multi-objective models ↩

Controls the initial presolve level used for multi-objective models. Value 0 disables the initial presolve, value 1 applies presolve conservatively, and value 2 applies presolve aggressively. The default -1 value usually applies presolve conservatively. Aggressive presolve may increase the chance of the objective values being slightly different than those for other options.

Default: `-1`

multobj *(boolean)*: Controls the hierarchical optimization of multiple objectives ↩

Default: `0`

names *(boolean)*: Indicator for loading names ↩

Default: `1`

value	meaning
0	Do not load GAMS names into Gurobi model
1	Load GAMS names into Gurobi model

networkalg *(integer)*: Network simplex algorithm ↩

Default: `-1`

networkcuts *(integer)*: Network cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Conservative
`2`	Aggressive

nlpheur *(boolean)*: Controls the NLP heuristic for non-convex quadratic models ↩

The NLP heuristic uses a non-linear barrier solver to find feasible solutions to non-convex quadratic models. It can often find solutions much more quickly than the alternative, but in some cases it can consume significant runtime without producing a solution.

Note: Only affects non-convex quadratic models.

Default: `1`

nlreform *(boolean)*: Reform nonlinear equations to Gurobi general constraints ↩

Default: `1`

nodefiledir *(string)*: Directory for MIP node files ↩

Determines the directory into which nodes are written when node memory usage exceeds the specified NodefileStart value.

Default: `.`

nodefilestart *(real)*: Memory threshold for writing MIP tree nodes to disk ↩

Controls the point at which MIP tree nodes are written to disk. Whenever node storage exceeds the specified value (in GBytes), nodes are written to disk.

Default: `maxdouble`

nodelimit *(real)*: MIP node limit ↩

Default: `maxdouble`

nodemethod *(integer)*: Method used to solve MIP node relaxations ↩

Algorithm used for MIP node relaxations. Note that barrier is not an option for MIQP node relaxations.

Default: `-1`

value	meaning
`-1`	Automatic
`0`	Primal simplex
`1`	Dual simplex
`2`	Barrier

nonconvex *(integer)*: Control how to deal with non-convex quadratic programs ↩

Sets the strategy for handling non-convex quadratic objectives or non-convex quadratic constraints. With setting 0, an error is reported if the original user model contains non-convex quadratic constructs (unless Q matrix linearization, as controlled by the PreQLinearize parameter, removes the non-convexity). With setting 1, an error is reported if non-convex quadratic constructs could not be discarded or linearized during presolve. With setting 2, non-convex quadratic problems are solved by translating them into bilinear form and applying spatial branching. The default -1 setting is currently almost equivalent to 2, except that it takes less care to avoid presolve reductions that might transform a convex constraint into one that can no longer be detected to be convex, and thus can sometimes perform more presolve reductions.

Default: `-1`

norelheurtime *(real)*: Limits the amount of time (in seconds) spent in the NoRel heuristic ↩

Limits the amount of time (in seconds) spent in the NoRel heuristic. This heuristic searches for high-quality feasible solutions before solving the root relaxation. It can be quite useful on models where the root relaxation is particularly expensive. Note that this parameter will introduce non-determinism - different runs may take different paths. Use norelheurwork parameter for deterministic results.

Default: `0`

norelheurwork *(real)*: Limits the amount of work performed by the NoRel heuristic ↩

Limits the amount of work spent in the NoRel heuristic. This heuristic searches for high-quality feasible solutions before solving the root relaxation. It can be quite useful on models where the root relaxation is particularly expensive. The work metric used in this parameter is tough to define precisely. A single unit corresponds to roughly a second, but this will depend on the machine, the core count, and in some cases the model. You may need to experiment to find a good setting for your model.

Default: `0`

normadjust *(integer)*: Simplex pricing norm ↩

Chooses from among multiple pricing norm variants. The default value of -1 chooses automatically.

Default: `-1`

numericfocus *(integer)*: Set the numerical focus ↩

The `NumericFocus` parameter controls the degree to which the code attempts to detect and manage numerical issues. The default setting makes an automatic choice, with a slight preference for speed. Settings 1-3 increasingly shift the focus towards being more careful in numerical computations. With higher values, the code will spend more time checking the numerical accuracy of intermediate results, and it will employ more expensive techniques in order to avoid potential numerical issues.

Default: `0`

obbt *(integer)*: Controls aggressiveness of optimality-based bound tightening ↩

Default: `-1`

objnabstol *(string)*: Allowable absolute degradation for objective ↩

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization (MultObj). The syntax for this parameter is `ObjNAbsTol ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. If it achieves objective value z when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most `ObjNAbsTol`.

objnreltol *(string)*: Allowable relative degradation for objective ↩

This parameter is used to set the allowable degradation for an objective when doing hierarchical multi-objective optimization (MultObj). The syntax for this parameter is `ObjNRelTol ObjVarName value`.

Hierarchical multi-objective optimization will optimize for the different objectives in the model one at a time, in priority order. If it achieves objective value z when it optimizes for this objective, then subsequent steps are allowed to degrade this value by at most `ObjNRelTol`$*|z|$.

objscale *(real)*: Objective scaling ↩

Divides the model objective by the specified value to avoid numerical errors that may result from very large objective coefficients. The default value of 0 decides on the scaling automatically. A value less than zero uses the maximum coefficient to the specified power as the scaling (so `ObjScale=-0.5` would scale by the square root of the largest objective coefficient).

Range: [`-1`, ∞]

Default: `0`

optimalitytol *(real)*: Dual feasibility tolerance ↩

Reduced costs must all be larger than `OptimalityTol` in the improving direction in order for a model to be declared optimal.

Range: [`1e-09`, `0.01`]

Default: `1e-06`

.partition *(integer)*: Variable partition value ↩

The MIP solver can perform a solution improvement heuristic using user-provided partition information. The provided partition number can be positive, which indicates that the variable should be included when the correspondingly numbered sub-MIP is solved, 0 which indicates that the variable should be included in every sub-MIP, or -1 which indicates that the variable should not be included in any sub-MIP. Variables that are not included in the sub-MIP are fixed to their values in the current incumbent solution.

To give an example, imagine you are solving a model with 400 variables and you set the partition attribute to -1 for variables 0-99, 0 for variables 100-199, 1 for variables 200-299, and 2 for variables 300-399. The heuristic would solve two sub-MIP models: sub-MIP 1 would fix variables 0-99 and 300-399 to their values in the incumbent and solve for the rest, while sub-MIP 2 would fix variables 0-99 and 200-299.

The parameter PartitionPlace controls the use of the heursitic. The parition numbers are specified through the option `.partition`. The syntax for dot options is explained in the Introduction chapter of the Solver Manual.

Default: `0`

partitionplace *(integer)*: Controls when the partition heuristic runs ↩

This option works in combination with the Partition number for variables. Setting this option and providing some partitions enables the partitioning heuristic, which uses large-neighborhood search to try to improve the current incumbent solution.

This parameter determines where that heuristic runs. Options are:

- Before the root relaxation is solved (16)
- At the start of the root cut loop (8)
- At the end of the root cut loop (4)
- At the nodes of the branch-and-cut search (2)
- When the branch-and-cut search terminates (1)

The parameter value is a bit vector, where each bit turns the heuristic on or off at that place. The numerical values next to the options listed above indicate which bit controls the corresponding option. Thus, for example, to enable the heuristic at the beginning and end of the root cut loop (and nowhere else), you would set the 8 bit and the 4 bit to 1, which would correspond to a parameter value of 12.

The recommended value is 15 which indicates that every option except the first one listed above is enabled.

Default: `15`

perturbvalue *(real)*: Simplex perturbation magnitude ↩

Range: [`0`, ∞]

Default: `0.0002`

poolgap *(real)*: Relative gap for solutions in pool ↩

Determines how large a gap to tolerate in stored solutions. When this parameter is set to a non-default value, solutions whose objective values exceed that of the best known solution by more than the specified (relative) gap are discarded. For example, if the MIP solver has found a solution at objective 100, then a setting of `PoolGap=0.2` would discard solutions with objective worse than 120 (assuming a minimization objective).

Default: `maxdouble`

poolgapabs *(real)*: Absolute gap for solutions in pool ↩

Determines how large a (absolute) gap to tolerate in stored solutions. When this parameter is set to a non-default value, solutions whose objective values exceed that of the best known solution by more than the specified (absolute) gap are discarded. For example, if the MIP solver has found a solution at objective 100, then a setting of `PoolGapAbs=20` would discard solutions with objective worse than 120 (assuming a minimization objective).

Default: `maxdouble`

poolsearchmode *(integer)*: Choose the approach used to find additional solutions ↩

With the default setting (`PoolSearchMode=0`), the MIP solver tries to find an optimal solution to the model. It keeps other solutions found along the way, but those are incidental. By setting this parameter to a non-default value, the MIP search will continue after the optimal solution has been found in order to find additional, high-quality solutions. With a setting of 2, it will find the n best solutions, where n is determined by the value of the PoolSolutions parameter. With a setting of 1, it will try to find additional solutions, but with no guarantees about the quality of those solutions. The cost of the solve will increase with increasing values of this parameter.

Once optimization is complete, the *PoolObjBound* attribute (printed to the log) can be used to evaluate the quality of the solutions that were found. For example, a value of `PoolObjBound=100` indicates that there are no other solutions with objective better 100, and thus that any known solutions with objective better than 100 are better than any as-yet undiscovered solutions.

Default: `0`

poolsolutions *(integer)*: Number of solutions to keep in pool ↩

Determines how many MIP solutions are stored. For the default value of PoolSearchMode, these are just the solutions that are found along the way in the process of exploring the MIP search tree. For other values of PoolSearchMode, this parameter sets a target for how many solutions to find, so larger values will impact performance.

Default: `10`

precrush *(boolean)*: Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model ↩

Allows presolve to translate constraints on the original model to equivalent constraints on the presolved model. This parameter is turned on when you use BCH with Gurobi.

Default: `0`

predeprow *(integer)*: Presolve dependent row reduction ↩

Controls the presolve dependent row reduction, which eliminates linearly dependent constraints from the constraint matrix. The default setting (-1) applies the reduction to continuous models but not to MIP models. Setting 0 turns the reduction off for all models. Setting 1 turns it on for all models.

Default: `-1`

predual *(integer)*: Presolve dualization ↩

Depending on the structure of the model, solving the dual can reduce overall solution time. The default setting uses a heuristic to decide. Setting 0 forbids presolve from forming the dual, while setting 1 forces it to take the dual. Setting 2 employs a more expensive heuristic that forms both the presolved primal and dual models (on two threads), and heuristically chooses one of them.

Default: `-1`

premiqcpform *(integer)*: Format of presolved MIQCP model ↩

Option 0 leaves the model in MIQCP form, so the branch-and-cut algorithm will operate on a model with arbitrary quadratic constraints. Option 1 always transforms the model into MISOCP form; quadratic constraints are transformed into second-order cone constraints. Option 2 always transforms the model into disaggregated MISOCP form; quadratic constraints are transformed into rotated cone constraints, where each rotated cone contains two terms and involves only three variables.

Default: `-1`

value	meaning
-1	Auto
0	Always leaves the model in MIQCP form
1	Always transforms the model into MISOCP form
2	Always transforms the model into disaggregated MISOCP form

prepasses *(integer)*: Presolve pass limit ↩

Limits the number of passes performed by presolve. The default setting (-1) chooses the number of passes automatically.

Default: -1

preqlinearize *(integer)*: Presolve Q matrix linearization ↩

Options 1 and 2 attempt to linearize quadratic constraints or a quadratic objective, potentially transforming an MIQP or MIQCP model into an MILP. Option 1 focuses on getting a strong LP relaxation. Option 2 aims for a compact relaxation. Option 0 always leaves Q matrices unmodified. The default setting (-1) chooses automatically.

Default: -1

value	meaning
-1	Auto
0	Linearization off
1	Force Linearization and get strong LP relaxation
2	Force Linearization and get compact relaxation

Presolve *(integer)*: Presolve level ↩

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

presos1bigm *(real)*: Controls largest coefficient in SOS1 reformulation ↩

Controls the automatic reformulation of SOS1 constraints into binary form. SOS1 constraints are often handled more efficiently using a binary representation. The reformulation often requires big-M values to be introduced as coefficients. This parameter specifies the largest big-M that can be introduced by presolve when performing this reformulation. Larger values increase the chances that an SOS1 constraint will be reformulated, but very large values (e.g., 1e8) can lead to numerical issues.

The default value of -1 chooses a threshold automatically. You should set the parameter to 0 to shut off SOS1 reformulation entirely, or a large value to force reformulation.

Range: [-1, 1e+10]

Default: -1

presos1encoding *(integer)*: Controls SOS1 reformulation ↩

Controls the automatic reformulation of SOS1 constraints. Such constraints can be handled directly by the MIP branch-and-cut algorithm, but they are often handled more efficiently by reformulating them using binary or integer variables. There are several diffent ways to perform this reformulation; they differ in their size and strength. Smaller reformulations add fewer variables and constraints to the model. Stronger reformulations reduce the number of branch-and-cut nodes required to solve the resulting model.

Options 0 and 1 of this parameter encode an SOS1 constraint using a formulation whose size is linear in the number of SOS members. Option 0 uses a so-called multiple choice model. It usually produces an LP relaxation that is easier to solve. Option 1 uses an incremental model. It often gives a stronger representation, reducing the amount of branching required to solve harder problems.

Options 2 and 3 of this parameter encode the SOS1 using a formulation of logarithmic size. They both only apply when all the variables in the SOS1 are non-negative. Option 3 additionally requires that the sum of the variables in the SOS1 is equal to 1. Logarithmic formulations are often advantageous when the SOS1 constraint has a large number of members. Option 2 focuses on a formulation whose LP relaxation is easier to solve, while option 3 has better branching behaviour.

The default value of -1 chooses a reformulation for each SOS1 constraint automatically.

Note that the reformulation of SOS1 constraints is also influenced by the PreSOS1BigM parameter. To shut off the reformulation entirely you should set that parameter to 0.

Default: `-1`

presos2bigm *(real)*: Controls largest coefficient in SOS2 reformulation ↩

Controls the automatic reformulation of SOS2 constraints into binary form. SOS2 constraints are often handled more efficiently using a binary representation. The reformulation often requires big-M values to be introduced as coefficients. This parameter specifies the largest big-M that can be introduced by presolve when performing this reformulation. Larger values increase the chances that an SOS2 constraint will be reformulated, but very large values (e.g., 1e8) can lead to numerical issues.

The default value of 0 disables the reformulation. You can set the parameter to -1 to choose an automatic approach, or a large value to force reformulation.

Range: [`-1`, `1e+10`]

Default: `-1`

presos2encoding *(integer)*: Controls SOS2 reformulation ↩

Controls the automatic reformulation of SOS2 constraints. Such constraints can be handled directly by the MIP branch-and-cut algorithm, but they are often handled more efficiently by reformulating them using binary or integer variables. There are several diffent ways to perform this reformulation; they differ in their size and strength. Smaller reformulations add fewer variables and constraints to the model. Stronger reformulations reduce the number of branch-and-cut nodes required to solve the resulting model.

Options 0 and 1 of this parameter encode an SOS2 constraint using a formulation whose size is linear in the the number of SOS members. Option 0 uses a so-called multiple choice model. It usually produces an LP relaxation that is easier to solve. Option 1 uses an incremental model. It often gives a stronger representation, reducing the amount of branching required to solve harder problems.

Options 2 and 3 of this parameter encode the SOS2 using a formulation of logarithmic size. They both only apply when all the variables in the SOS2 are non-negative. Option 3 additionally requires that the sum of the variables in the SOS2 is equal to 1. Logarithmic formulations are often advantageous when the SOS2 constraint has a large number of members. Option 2 focuses on a formulation whose LP relaxation is easier to solve, while option 3 has better branching behaviour.

The default value of -1 chooses a reformulation for each SOS2 constraint automatically.

Note that the reformulation of SOS2 constraints is also influenced by the PreSOS2BigM parameter. To shut off the reformulation entirely you should set that parameter to 0.

Default: `-1`

presparsify *(integer)*: Presolve sparsify reduction ↩

This reduction can sometimes significantly reduce the number of nonzero values in the presolved model.

Default: `-1`

value	meaning
`-1`	Auto
`0`	Disable the presolve sparsify reduction
`1`	Enable the presolve sparsify reduction for MIPs
`2`	Enable the presolve sparsify reduction for all model types

printoptions *(boolean)*: List values of all options to GAMS listing file ↩

Default: `0`

value	meaning
`0`	Do not list option values to GAMS listing file
`1`	List option values to GAMS listing file

.prior *(real)*: Branching priorities ↩

GAMS allows to specify priorities for discrete variables only. Gurobi can detect that continuous variables are implied discrete variables and can utilize priorities. Such priorities can be specified through a GAMS/Gurobi solver option file. The syntax for dot options is explained in the Introduction chapter of the Solver Manual. The priorities are only passed on to Gurobi if the model attribute `priorOpt` is turned on.

Default: `1`

projimpliedcuts *(integer)*: Projected implied bound cut generation ↩

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Moderate
`2`	Aggressive

psdcuts *(integer)*: PSD cut generation ↩

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Moderate
`2`	Aggressive

psdtol *(real)*: Positive semi-definite tolerance ↩

Positive semi-definite tolerance (for QP/MIQP). Sets a limit on the amount of diagonal perturbation that the optimizer is allowed to automatically perform on the Q matrix in order to correct minor PSD violations. If a larger perturbation is required, the optimizer will terminate stating the problem is not PSD.

Range: [`0`, ∞]

Default: `1e-06`

pumppasses *(integer)*: Feasibility pump heuristic control ↩

Note that this heuristic is only applied at the end of the MIP root.

This heuristic is quite expensive, and generally produces poor quality solutions. You should generally only use it if other means, including exploration of the tree with default settings, fail to produce a feasible solution.

Default: `-1`

qcpdual *(boolean)*: Compute dual variables for QCP models ↩

Determines whether dual variable values are computed for QCP models. Computing them can add significant time to the optimization, so you should turn this parameter to 0 if you do not need them.

Default: `1`

value	meaning
`0`	Do not compute dual for QCP problem
`1`	Compute dual for QCP problem

qextractalg *(integer)*: quadratic extraction algorithm in GAMS interface ↩

Default: `0`

value	meaning
`0`	Automatic
`1`	ThreePass: Uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage.
`2`	DoubleForward: Uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.

value	meaning
3	Concurrent: Uses ThreePass and DoubleForward in parallel. As soon as one finishes, the other one stops.

quad *(integer)*: Quad precision computation in simplex ↩

Enables or disables quad precision computation in simplex. The -1 default setting allows the algorithm to decide.

Default: `-1`

readparams *(string)*: Read Gurobi parameter file ↩

relaxliftcuts *(integer)*: Relax-and-lift cut generation ↩

Default: `-1`

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

rerun *(integer)*: Resolve without presolve in case of unbounded or infeasible ↩

In case Gurobi reports *Model was proven to be either infeasible or unbounded*, this option decides about a resolve without presolve which will determine the exact model status. If the option is set to *auto* and the model fits into demo limits, the problems is resolved.

Default: `-1`

value	meaning
-1	No
0	Auto
1	Yes

rins *(integer)*: RINS heuristic ↩

Default value (-1) chooses automatically. A value of 0 shuts off RINS. A positive value n applies RINS at every n-th node of the MIP search tree.

Default: `-1`

rltcuts *(integer)*: RLT cut generation ↩

Default: `-1`

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

rngrestart *(string)*: Write GAMS readable ranging information file ↩

If the extension specified is `gdx`, a GDX file is exported, and a GAMS file otherwise.

scaleflag *(integer)*: Model scaling ↩

Controls model scaling. By default, the rows and columns of the model are scaled in order to improve the numerical properties of the constraint matrix. The scaling is removed before the final solution is returned. Scaling typically reduces solution times, but it may lead to larger constraint violations in the original, unscaled model. Turning off scaling `ScaleFlag=0` can sometimes produce smaller constraint violations. Choosing a more aggressive scaling option `ScaleFlag=2` can sometimes improve performance for particularly numerically difficult models.

Default: `-1`

seed *(integer)*: Modify the random number seed ↩

Modifies the random number seed. This acts as a small perturbation to the solver, and typically leads to different solution paths.

Default: `0`

sensitivity *(boolean)*: Provide sensitivity information ↩

Default: `0`

value	meaning
`0`	Do not provide sensitivity information
`1`	Provide sensitivity information

sifting *(integer)*: Sifting within dual simplex ↩

Enables or disables sifting within dual simplex. Sifting is often useful for LP models where the number of variables is many times larger than the number of constraints. With a *Moderate* setting, sifting will be applied to LP models and to the root node for MIP models. With an *Aggressive* setting, sifting will be also applied to the nodes of a MIP. Note that this parameter has no effect if you aren't using dual simplex. Note also that sifting will be skipped in cases where it is obviously a worse choice, even when sifting has been selected.

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Moderate
`2`	Agressive

siftmethod *(integer)*: LP method used to solve sifting sub-problems ↩

Note that this parameter only has an effect when you are using dual simplex and sifting has been selected (either by the automatic method, or through the Sifting parameter).

Default: `-1`

value	meaning
-1	Auto
0	Primal Simplex
1	Dual Simplex
2	Barrier

simplexpricing *(integer)*: Simplex variable pricing strategy ↩

Default: -1

value	meaning
-1	Auto
0	Partial Pricing
1	Steepest Edge
2	Devex
3	Quick-Start Steepest Edge

softmemlimit *(real)*: Soft memory limit ↩

Default: maxdouble

solfiles *(string)*: Location to store intermediate solution files ↩

During the MIP solution process, multiple incumbent solutions are typically found on the path to finding a proven optimal solution. Setting this parameter to a non-empty string causes these solutions to be written to files (in .sol format) as they are found. The MIP solver will append _n.sol to the value of the parameter to form the name of the file that contains solution number n. For example, setting the parameter to value solutions/mymodel will create files mymodel_0.sol, mymodel_1.sol, etc., in directory solutions.

solnpool *(string)*: Controls export of alternate MIP solutions ↩

The GDX file specified by this option will contain a set call index that contains the names of GDX files with the individual solutions. For details see example model **dumpsol** in the GAMS Test Library. The option PoolSolutions, PoolSearchModel, and PoolGap control the search for alternative solutions. Please also refer to the secion Solution Pool.

solnpoolmerge *(string)*: Controls export of alternate MIP solutions for merged GDX solution file ↩

Similar to SolnPool this option stores multiple alternative solutions to a MIP problem, but in a single GDX file. The GDX file specified by this option will contain all variables with an additional first index (determined through SolnPoolPrefix) as parameters. The option PoolSolutions, PoolSearchModel, and PoolGap control the search for alternative solutions. Please also refer to the secion Solution Pool.

solnpoolnumsym *(integer)*: Maximum number of variable symbols when writing merged GDX solution file ↩

Default: 10

solnpoolprefix *(string)*: First dimension of variables for merged GDX solution file or file name prefix for GDX solution files ↩

Default: soln

solutionlimit *(integer)*: MIP feasible solution limit ↩

Default: maxint

solutiontarget *(integer)*: Specify the solution target for LP ↩

Default: -1

value	meaning
-1	Auto
0	primal and dual optimal, and basic
1	primal and dual optimal

solvefixed *(boolean)*: Indicator for solving the fixed problem for a MIP to get a dual solution ↩

Default: 1

value	meaning
0	Do not solve the fixed problem
1	Solve the fixed problem

startnodelimit *(integer)*: Node limit for MIP start sub-MIP ↩

This parameter limits the number of branch-and-bound nodes explored when completing a partial MIP start. The default value of -1 uses the value of the SubMIPNodes parameter. A value of -2 means to only check full MIP starts for feasibility and to ignore partial MIP starts. A value of -3 shuts off MIP start processing entirely. Non-negative values are node limits.

Default: -1

strongcgcuts *(integer)*: Strong-CG cut generation ↩

Controls Strong Chvátal-Gomory (Strong-CG) cut generation. Use 0 to disable these cuts, 1 for moderate cut generation, or 2 for aggressive cut generation. The default -1 value chooses automatically. Overrides the Cuts parameter.

Default: -1

submipcuts *(integer)*: Sub-MIP cut generation ↩

See the description of the global Cuts parameter for further information.

Default: -1

submipnodes *(integer)*: Nodes explored by sub-MIP heuristics ↩

Limits the number of nodes explored by the heuristics, like RINS. Exploring more nodes can produce better solutions, but it generally takes longer.

Default: 500

symmetry *(integer)*: Symmetry detection ↩

Default: -1

value	meaning
-1	Auto
0	Off
1	Conservative
2	Aggressive

threads *(integer)*: Number of parallel threads to use ↩

Default number of parallel threads allowed for any solution method. Non-positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.

Default: `GAMS threads`

timelimit *(real)*: Time limit ↩

Default: `GAMS reslim`

tunecleanup *(real)*: Enables a tuning cleanup phase ↩

Enables a cleanup phase at the end of tuning. The parameter indicates the percentage of total tuning time to devote to this phase, with a goal of reducing the number of parameter changes required to achieve the best tuning result.

Default: `0`

tunecriterion *(integer)*: Specify tuning criterion ↩

Modifies the tuning criterion for the tuning tool. The primary tuning criterion is always to minimize the runtime required to find a proven optimal solution. However, for MIP models that don't solve to optimality within the specified time limit, a secondary criterion is needed. Set this parameter to 1 to use the optimality gap as the secondary criterion. Choose a value of 2 to use the objective of the best feasible solution found. Choose a value of 3 to use the best objective bound. Choose 0 to ignore the secondary criterion and focus entirely on minimizing the time to find a proven optimal solution. The default value of -1 chooses automatically.

Default: `-1`

tunedynamicjobs *(integer)*: Enables distributed tuning using a dynamic set of workers ↩

Enables distributed parallel tuning, which can significantly increase the performance of the tuning tool. A value of n causes the tuning tool to use a dynamic set of up to n workers in parallel. These workers are used for a limited amount of time and afterwards potentially released so that they are available for other remote jobs. A value of -1 allows the solver to use an unlimited number of workers. Note that this parameter can be combined with TuneJobs to get a static set of workers and a dynamic set of workers for distributed tuning. You can use the WorkerPool parameter to provide a distributed worker cluster.

Note that distributed tuning is most effective when the worker machines have similar performance. Distributed tuning doesn't attempt to normalize performance by server, so it can incorrectly attribute a boost in performance to a parameter change when the associated setting is tried on a worker that is significantly faster than the others.

Default: `0`

tunejobs *(integer)*: Enables distributed tuning using a static set of workers ↩

Enables distributed parallel tuning, which can significantly increase the performance of the tuning tool. A value of n causes the tuning tool to distribute tuning work among n parallel jobs. These jobs are distributed among a set of workers. Use the WorkerPool parameter to provide a list of available workers.

Note that distributed tuning is most effective when the workers have similar performance. Distributed tuning doesn't attempt to normalize performance by worker, so it can incorrectly attribute a boost in performance to a parameter change when the associated setting is tried on a worker that is significantly faster than the others.

Default: `0`

tunemetric *(integer)*: Metric to aggregate results into a single measure ↩

A single tuning run typically produces multiple timing results for each candidate parameter set, either as a result of performing multiple trials, or tuning multiple models, or both. This parameter controls how these results are aggregated into a single measure. The default setting (-1) chooses the aggregation automatically; setting 0 computes the average of all individual results; setting 1 takes the maximum.

Default: `-1`

tuneoutput *(integer)*: Tuning output level ↩

Default: `2`

value	meaning
`0`	No output
`1`	Summary output only when a new best parameter set is found
`2`	Summary output for each parameter set that is tried
`3`	Summary output, plus detailed solver output, for each parameter set tried

tuneresults *(integer)*: Number of improved parameter sets returned ↩

The tuning tool often finds multiple parameter sets that improve over the baseline settings. This parameter controls how many of these sets should be retained when tuning is complete. A non-negative value indicates how many sets should be retained. The default value (-1) retains the efficient frontier of parameter sets. That is, it retains the best set for one changed parameter, the best for two changed parameters, etc. Sets that aren't on the efficient frontier are discarded. If you interested in all the sets, use value -2 for the parameter.

Note that the first set in the results is always the set of parameters which was used for the first solve, the baseline settings. This set serves as the base for any improvement. So if you are interested in the best tuned set of parameters you need to request at least 2 tune results. The first one (with index 0) will be the baseline setting and the second one (with index 1) will be the best set found during tuning.

Default: `-1`

tunetargetmipgap *(real)*: A target gap to be reached ↩

A target gap to be reached. As soon as the tuner has found parameter settings that allow Gurobi to reach the target gap for the given model(s), it stops trying to improve parameter settings further. Instead, the tuner switches into the cleanup phase (see TuneCleanup parameter).

Default: `0`

tunetargettime *(real)*: A target runtime in seconds to be reached ↩

A target runtime in seconds to be reached. As soon as the tuner has found parameter settings that allow Gurobi to solve the model(s) within the target runtime, it stops trying to improve parameter settings further. Instead, the tuner switches into the cleanup phase (see TuneCleanup parameter).

Default: `0.005`

tunetimelimit *(real)*: Time limit for tuning ↩

Limits total tuning runtime (in seconds). The default setting (-1) chooses a time limit automatically.

Default: `maxdouble`

tunetrials *(integer)*: Perform multiple runs on each parameter set to limit the effect of random noise ↩

Performance on a MIP model can sometimes experience significant variations due to random effects. As a result, the tuning tool may return parameter sets that improve on the baseline only due to randomness. This parameter allows you to perform multiple solves for each parameter set, using different Seed values for each, in order to reduce the influence of randomness on the results. The default value of 0 indicates an automatic choice that depends on model characteristics.

Note: Only affects mixed integer programming (MIP) models

Default: `0`

Tuning *(string)*: Parameter Tuning ↩

Invokes the Gurobi parameter tuning tool. The mandatory value following the keyword specifies a GAMS/Gurobi option file. All options found in this option file will be used but not modified during the tuning. A sequence of file names specifying existing problem files may follow the option file name. The files can be in MPS, REW, LP, RLP, and ILP format created by the WriteProb option. Gurobi will tune the parameters either for the problem provided by GAMS (no additional problem files specified) or for the suite of problems listed after the GAMS/Gurobi option file name without considering the problem provided by GAMS. The result of such a run is the updated GAMS/Gurobi option file with a tuned set of parameters. In case the option TuneResults is larger than 1, GAMS/Gurobi will create a sequence of GAMS/Gurobi option files. The solver and model status returned to GAMS will be `NORMAL COMPLETION` and `NO SOLUTION`. Tuning is incompatible with advanced features like FeasOpt of GAMS/Gurobi.

usebasis *(integer)*: Use basis from GAMS ↩

If `UseBasis` is not specified, GAMS (via option `BRatio`) decides if the starting basis or a primal/dual solution is given to Gurobi. If `UseBasis` is explicitly set in an option file then the basis or a primal/dual solution is passed to Gurobi independent of the GAMS option `BRatio`. Please note, if Gurobi uses a starting basis presolve will be skipped.

Default: `GAMS bratio`

value	meaning
0	No basis
1	Supply basis if basis is full otherwise provide primal dual solution
2	Supply basis iff basis is full
3	Supply primal dual solution

varbranch *(integer)*: Branch variable selection strategy ↩

Default: `-1`

value	meaning
-1	Auto
0	Pseudo Reduced Cost Branching

value	meaning
1	Pseudo Shadow Price Branching
2	Maximum Infeasibility Branching
3	Strong Branching

varhint *(boolean)*: Guide heuristics and branching through variable hints ↩

If you know that a variable is likely to take a particular value in high quality solutions of a MIP model, you can provide this information as a hint. If `VarHint` option is active, GAMS/Gurobi will pass variable levels rounded to the nearest integer as hints to Gurobi if their level is within `TryInt` of an integer. The closer the level is to the rounded integer the higher your level of confidence in this hint. Internally this is recalculated into a Gurobi variable hint priority: $\lceil \frac{1}{\max(10^{-6}, \lvert x.l - [x.l] \rvert)} \rceil$

The Gurobi MIP solver will use these variable hints in a number of different ways. Hints will affect the heuristics that Gurobi uses to find feasible solutions, and the branching decisions that Gurobi makes to explore the MIP search tree. In general, high quality hints should produce high quality MIP solutions faster. In contrast, low quality hints will lead to some wasted effort, but shouldn't lead to dramatic performance degradations.

Variables hints and MIP starts are similar in concept, but they behave in very different ways. If you specify a MIP start, the Gurobi MIP solver will try to build a single feasible solution from the provided set of variable values. If you know a solution, you should use a MIP start to provide it to the solver. In contrast, variable hints provide guidance to the MIP solver that affects the entire solution process. If you have a general sense of the likely values for variables, you should provide them through variable hints.

Default: `0`

workerpassword *(string)*: Password for distributed worker cluster ↩

When using a distributed algorithm (the distributed concurrent MIP solver or distributed tuning), this parameter allows you to specify the password for the workers listed in the WorkerPool parameter.

workerpool *(string)*: Distributed worker cluster ↩

When using a distributed algorithm (distributed MIP, distributed concurrent, or distributed tuning), this parameter allows you to specify a Remote Services cluster that will provide distributed workers. You should also specify the access password for that cluster, if there is one, in the WorkerPassword parameter.

You can provide a comma-separated list of machines for added robustness. If the first node in the list is unavailable, the client will attempt to contact the second node, etc.

To give an example, if you have a Remote Services cluster that uses port 61000 on a pair of machines named server1 and server2, you could set WorkerPool to `server1:61000` `server1:61000,server2:61000`.

worklimit *(real)*: Work limit ↩

Limits the total work expended (in work units).

In contrast to the TimeLimit, work limits are deterministic. This means that on the same hardware and with the same parameter and attribute settings, a work limit will stop the optimization of a given model at the exact same point every time. One work unit corresponds very roughly to one second on a single thread, but this greatly depends on the hardware on which Gurobi is running and the model that is being solved.

Note that optimization may not stop immediately upon hitting the work limit. It will stop when the optimization is next in a deterministic state, and it will then perform the required additional computations of the attributes associated with the terminated optimization. As a result, the Work attribute may be larger than the specified WorkLimit upon completion, and repeating the optimization with a WorkLimit set to the Work attribute of the stopped optimization may result in additional computations and a larger attribute value.

Default: `maxdouble`

writeparams *(string)*: Write Gurobi parameter file ↩

writeprob *(string)*: Save the problem instance ↩

zerohalfcuts *(integer)*: Zero-half cut generation ↩

See the description of the global Cuts parameter for further information.

Default: `-1`

value	meaning
`-1`	Auto
`0`	Off
`1`	Conservative
`2`	Aggressive

zeroobjnodes *(integer)*: Zero objective heuristic control ↩

Note that this heuristic is only applied at the end of the MIP root, and only when no other root heuristic finds a feasible solution.

This heuristic is quite expensive, and generally produces poor quality solutions. You should generally only use it if other means, including exploration of the tree with default settings, fail to produce a feasible solution.

Default: `-1`

5.17.8 Setting up a GAMS/Gurobi-Link license

The GAMS/Gurobi-Link requires two licenses:

- A GAMS license with components GAMS/GUROBI-Link and GAMS/BASE (to generate models beyond the demo limits)

- A Gurobi license which you need to get directly from Gurobi

An attempt to use the GAMS/Gurobi solver with a GAMS/Gurobi-Link license but without a properly set up Gurobi license will result in a licensing error with a message describing the problem. For example, the following message is sent to the log when attempting to solve a model with dimensions beyond the community limits:

```
--- Executing GUROBI (Solvelink=2): elapsed 0:00:00.047[LST:85]

Gurobi           32.2.0 rc62c018 Released Aug 26, 2020 WEI x86 64bit/MS Window

Gurobi link license.
*** Cannot initialize Gurobi environment.
*** Could be a missing or invalid license. (status=10009|10009)
```

Starting with GAMS distribution 24.7, even demo sized models require a license from Gurobi. An attempt to solve a demo sized model without a Gurobi license installed results in:

```
--- Executing GUROBI (Solvelink=2): elapsed 0:00:00.016[LST:208]

Gurobi           32.2.0 rc62c018 Released Aug 26, 2020 WEI x86 64bit/MS Window

*** This solver runs with a demo license. No commercial use.
GAMS/Gurobi demo/community requires a Gurobi license from Gurobi Optimization.
*** Cannot initialize Gurobi environment.
*** Could be a missing or invalid license. (status=10009|10009)
```

To make the GAMS/Gurobi-Link work you do **not** need to download or install the Gurobi software but only your Gurobi license. GAMS will use it's own Gurobi DLL/shared library, so the **Gurobi license has to be valid for the Gurobi version GAMS uses**. You can download your Gurobi license from www.gurobi.com. Log in to your Gurobi account and go to `Download & Licenses` –> `Your Gurobi licenses`.

Now click on the license you want to download and click on `Get License Details`. On the license detail page, copy the `grbgetkey` command on the bottom.

Paste the `grbgetkey` command to a command/terminal prompt. Your GAMS system directory contains the `grbgetkey` program, so to make sure it is found you should open the terminal via GAMS Studio or add your GAMS system directory to the path variable. After running the `grbgetkey` command, you will be asked where you would like to save your Gurobi license.

```
$ ./grbgetkey [...]
info  : grbgetkey version 9.0.3, build v9.0.3rc0
info  : Contacting Gurobi key server...
info  : Key for license ID [...] was successfully retrieved
info  : License expires at the end of the day on 2021-08-31
info  : Saving license key...

In which directory would you like to store the Gurobi license key file?
[hit Enter to store it in /opt/gurobi]: /home/nb/gurobilic

info  : License [...] written to file /home/nb/gurobilic/gurobi.lic
info  : You may have saved the license key to a non-default location
info  : You need to set the environment variable GRB_LICENSE_FILE before you can use this license key
info  : GRB_LICENSE_FILE=/home/nb/gurobilic/gurobi.lic
```

Once you have saved your `gurobi.lic` file, you need to make GAMS/Gurobi aware of that license via environment variable `GRB_LICENSE_FILE`. The `environmentVariables` section in the GAMS configuration file (available as of GAMS version 31.1.0) is a convenient way to set the `GRB_LICENSE_FILE` environment variable. This can either be done by manually opening and editing the `gamsconfig.yaml` file which can be found in one of the standard locations or via the corresponding GAMS Configuration Editor in GAMS Studio. As a result, the configuration file should contain an entry for environment variable `GRB_LICENSE_FILE` that points to the Gurobi license, e.g.

```
[...]
environmentVariables:
  [...]
  - GRB_LICENSE_FILE:
      value: /home/nb/gurobilic/gurobi.lic
[...]
```

As an alternative you could also set `GRB_LICENSE_FILE` via the usual OS-specific ways to set environment variables.

To test whether the license setup has been successful, you can solve a model from the GAMS Model library, e.g. **[INDUS89]**, where you should get the following output.

```
$ gamslib indus89
Copy ASCII : indus89.gms
$ gams indus89 lp gurobi
--- Job indus89 Start 09/01/20 12:41:41 32.2.0 rc62c018 LEX-LEG x86 64bit/Linux
[...]
Gurobi           32.2.0 rc62c018 Released Aug 26, 2020 LEG x86 64bit/Linux

Gurobi link license.
Gurobi library version 9.0.3
[...]
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    7.1618935e+31   2.041808e+32   7.161894e+01      0s
    2990    1.1487366e+05   0.000000e+00   0.000000e+00      0s

Solved in 2990 iterations and 0.32 seconds
Optimal objective  1.148736556e+05

User-callback calls 3040, time in user-callback 0.00 sec
LP status(2): Model was solved to optimality (subject to tolerances).
--- Reading solution for model wsisn
*** Status: Normal completion
--- Job indus89.gms Stop 09/01/20 12:29:49 elapsed 0:00:00.835
```

Note

To install a Gurobi license key as described above, the `grbgetkey` program needs to be able to communicate with the Gurobi website. If it fails to work, the `grbprobe` program which can also be found in your GAMS system directory may be used to retrieve the required hardware information which then needs to be manually submitted to the Gurobi Website (detailed instructions can be found here).

5.18 Gather-Update-Solve-Scatter (GUSS)

5.18.1 Introduction

The purpose of this chapter is to detail an extension of the GAMS modeling system that allows collections of models (parameterized exogenously by a set of samples or indices) to be described, instantiated, and solved efficiently.

As a specific example, we consider the parametric optimization problem $\mathcal{P}(s)$ defined by:

$$\min_{x \in X(s)} f(x;s) \text{ s.t. } g(x;s) \leq 0 \tag{5.11}$$

where $s \in S = \{1, \ldots, K\}$. Note that each scenario s represents a different problem for which the optimization variable is x. The form of the constraint set as given above is simply for concreteness; equality constraints and range and bound constraints are trivial extensions of the above framework. Clearly the problems $\mathcal{P}(s)$ are interlinked. We intend to show how such problems can be easily specified within GAMS, and detail one type of algorithmic extension that can exploit the nature of the linkage. Other extensions of GAMS allow solves to be executed in parallel or by using grid computing resources. Note that in our description we will use the terms indexed, parameterized, and scenario somewhat interchangeably. An extended version of this chapter containing several examples is available as a paper at http://www.gams.com/modlib/adddocs/gusspaper.pdf.

5.18.2 Design Methodology

One of the most important functions of GAMS is to build a model instance from the collection of equations (i.e. an optimization model defined by the GAMS keyword `MODEL`) and corresponding data (consisting of the content of GAMS (sub)sets and parameters). Such a model instance is constructed or generated when the GAMS execution system executes a `SOLVE` statement. The generated model instance is passed to a solver which searches for a solution of this model instance and returns status information, statistics, and a (primal and dual) solution of the model instance. After the solver terminates, GAMS brings the solution back into the GAMS database, i.e. it updates the level (`.L`) and marginal (`.M`) fields of variable and equation symbols used in the model instance. Hence, the `SOLVE` statement can be interpreted as a complex operator against the GAMS database. The model instance generated by a `SOLVE` statement only lives during the execution of this one statement, and hence has no representation within the GAMS language. Moreover, its structure does fit the relational data model of GAMS. A model instance consists of vectors of bounds and right hand sides, a sparse matrix representation of the Jacobian, a representation of the non-linear expressions that allow the efficient calculation of gradient vectors and Hessian matrices, and so on.

This chapter is concerned with solving collections of models that have similar structure but modified data. As an example, consider a linear program of the form:

$$\min c^T x \text{ s.t. } Ax \geq b,\ \ell \leq x \leq u.$$

The data in this problem is (A, b, c, ℓ, u). Omitting some details, the following code could be used within GAMS to solve a collection of such linear programs in which each member of the collection has a different A matrix and lower bound ℓ:

```
Set       i / ... /, j / ... /;
Parameter A(i,j), b(i);
Variable  x(j), z, ...;
Equation  e(i), ...;
e(i).. sum(j, A(i,j)*x(j)) =g= b(i);
...
```

```
model mymodel /all/;

Set s / s1*s10 /
Parameter
    A_s(s,i,j) 'Scenario data'
    xlo_s(s,j) 'Scenario lower bound for variable x'
    xl_s(s,j)  'Scenario solution for x.l'
    em_s(s,i)  'Scenario solution for e.m';
Loop(s,
  A(i,j) = A_s(s,i,j);
  x.lo(j)= xlo_s(s,j);
  solve mymodel min z using lp;
  xl_s(s,j) = x.l(j);
  em_s(s,i) = e.m(i);
);
```

Summarizing, we solve one particular model (`mymodel`) in a loop over `s` with an unchanged model rim (i.e. the same individual variables and equations) but with different model data and different bounds for the variables. The change in model data for a subsequent solve statement does not depend on the previous model solutions in the loop.

The purpose of this new Gather-Update-Solve-Scatter (GUSS) manager is to provide syntax at the GAMS modeling level that makes an instance of a problem that provides limited access to treat that instance as an object, and allows the modeler to update portions of it iteratively. Specifically, we provide syntax that gives a list of data changes to an instance, and allows these changes to be applied sequentially to the instance (which is then solved without returning to GAMS). Thus, we can simulate a limited set of actions to be applied to the model instance object and retrieve portions of the solution of these changed instances back in the modeling environment. Such changes can be done to any model type in GAMS, including nonlinear problems and mixed integer models. However, the only changes we allow are to named parameters appearing in the equations and lower and upper bounds used in the model definition.

Thus, in the above example GUSS allows us to replace lines 15-21 by

```
Set dict / s.   scenario. ''
           A.   param.    A_s
           x.   lower.    xlo_s
           x.   level.    xl_s
           e.   marginal. em_s   /;
solve mymodel min z using lp scenario dict;
```

The three dimensional `set dict` (you can freely choose the name of this symbol) contains mapping information between symbols in the model (in the first position) and symbols that supply required update data or store solution information (in the third position), and the type of update/storing (in the second position). An exception to this rule is the tuple with label `scenario` in the second position. This tuple determines the symbol (in the first position) that is used as the scenario index. This scenario symbol can be a multidimensional set. A tuple in this set represents a single scenario. The remaining tuples in the `set dict` can be grouped into input and output tuples. Input tuples determine the modifications of the model instance prior to solving, while output tuples determine which part of the solution gets saved away. The following keywords can be used in the second position of the set `dict`:

Type	Keywords	Description
Input:	`param`	Supplies scenario data for a parameter used in the model
	`lower`	Supplies scenario lower bounds for a variable
	`upper`	Supplies scenario upper bounds for a variable
	`fixed`	Supplies scenario fixed bounds for a variable
Output:	`level`	Stores the levels of a scenario solution of variable or equation
	`marginal`	Stores the marginals of a scenario solution of variable or equation

Sets in the model cannot be updated. GUSS works as follows: GAMS generates the model instance for the original data. As with regular `SOLVE` statements, all the model data (e.g. parameter `A`) needs to be defined at this time. The model instance with the original data is also called the *base case*. The solution of the base case is reported back to GAMS in the regular way and is accessible via the regular `.L` and `.M` fields after the `SOLVE` statement. After solving the base case, the update data for the first scenario is applied to the model. The tuples with `lower`, `upper`, `fixed` update the bounds of the variables, whereas the tuples with `param` update the parameters in the model.

The scenario index `s` needs to be the first index in the parameters mapped in the `set dict`. The update of the model parameters goes far beyond updating the coefficients of the constraint matrix/objective function or the right hand side of an equation, as one can do with some other systems. GAMS stores all the necessary expressions of the constraints with the model instance, so the change in the constraint matrix coefficient is the result of an expression evaluation. For example, consider a term in the calculation of the cost for shipping a variable amount of goods `x(i,j)` between cities `i` and `j`. The expression for shipping cost is `d(i,j)*f*x(i,j)`, i.e. the distance between the cities times a freight rate `f` times the variable amount of goods. In order to find out the sensitivity of the solution with respect to the freight rate `f`, one can solve the same model with different values for `f`. In a matrix representation of the model one would need to calculate the coefficient of `x(i,j)` which is `d(i,j)*f`, but with GUSS it is sufficient to supply different values for `f` that potentially result in many modified coefficients on the matrix level. GUSS evaluates the shipping cost term and communicates the resulting matrix coefficient to the solver reliably behind the scenes.

After the variable bound and the model parameter updates have been applied and the resulting updates to the model instance data structures (e.g. constraint matrix) has been determined, the modified model instance is passed to the solver. Some solvers (e.g. Cplex, Gurobi, SoPlex, and Xpress) allow modifying a model instance. In these cases GUSS only communicates the changes from the previous model instance to the solver. This reduces the amount of data communicated to the solver and also, in the case of an LP model, allows the solver to restart from an advanced basis and its factorization. In the case of an NLP model, this provides initial values. After the solver determines the solution of a model instance, GUSS stores the part of the solution requested by the output tuples of `dict` to some GAMS parameters and continues with the next scenario. GUSS emphasizes on speed and only works with solver that allow to communicate the model instance through memory, Hence, the following solvers cannot be used as subsolvers of GUSS: ALPHAECP, BARON, CONVERT, DECISC, DECISM, DICOPT, EXAMINER, GAMSCHK, JAMS, KESTREL, LOGMIP, MILES, MPSGE, NLPEC, PATHNLP, SBB.

5.18.3 GUSS Options

The execution of GUSS can be parameterized using some options. Options are not passed through a solver option file but via another tuple in the `dict` set. The keyword in the second position of this tuple is `opt`. A one dimensional parameter is expected in the first position (or the label `&prime;&prime;`). This parameter may contain some of the following labels with values:

Options	Description
`OptfileInit`:	Option file number for the first solve (default from GAMS OptFile setting)
`Optfile`:	Option file number for subsequent solves (default 0)
`LogOption`:	Determines amount of log output: `0` - Moderate log (default) `1` - Minimal log `2` - Detailed log
`NoHotStart`:	Disable hot start capability in solver that supports hot starts (default 0)
`NoMatchLimit`:	Limit of unmatched scenario records (default 0)
`RestartType`::	Determines restart point for the scenarios `0` - Restart from last solution (default) `1` - Restart from solution of base case `2` - Restart from input point

Options	Description
SkipBaseCase:	Switch for solving the base case (0 solves the base case)
ReportLastScen:	Switch for reporting the solution of the last scenario rather than solution of the base case (default 0)
SolveEmpty:	Limit of solved empty scenarios, afterwards empty scenarios will be skipped (default 0)
UpdateType:	Scenario update mechanism: 0 - Set everything to zero and apply changes (default) 1 - Reestablish base case and apply changes 2 - Build on top of last scenario and apply changes

For the example model above the UpdateType setting would mean:

```
UpdateType=0:  loop(s, A(i,j) = A_s(s,i,j))
UpdateType=1:  loop(s, A(i,j) = A_base(i,j);
                       A(i,j) $= A_s(s,i,j))
UpdateType=2:  loop(s, A(i,j) $= A_s(s,i,j))
```

The option SkipBaseCase=1 allows the user to skip the base case. This means only the scenarios are solved and there is no solution reported back to GAMS in the traditional way. The third position in the opt-tuple can contain a parameter for storing the scenario solution attribute information, e.g. model and solve status, or needs to have the label ′′. The labels to store solution status information must be known to GAMS, so one needs to declare a set with such labels. A convenient way to enter these attributes is via System.GUSSModelAttributes:

```
Set ma 'GUSS Model Attributes' / System.GUSSModelAttributes /; display ma;


----      1 SET ma  GUSS Model Attributes

ModelStat,    SolveStat,    NumInfes ,    SumInfes ,    IterUsd
ResUsd   ,    ObjVal   ,    NodUsd   ,    ObjEst   ,    DomUsd
RObj     ,    MaxInfes ,    MeanInfes
```

The following example shows how to use some of the GUSS options and the use of a parameter to store some solution status information:

```
Set h solution headers / System.GUSSModelAttributes /;
Parameter
   o / SkipBaseCase 1, UpdateType 1, Optfile 1 /
   r_s(s,h) Solution status report;
Set dict / s.   scenario. ''
           o.   opt.      r_s
           a.   param.    a_s
           x.   lower.    xlo_s
           x.   level.    xl_s
           e.   marginal. em_s   /;
solve mymodel min z using lp scenario dict;
```

Please note that the domain set of the solution status report attributes (here h) must only contain model attributes known to GUSS. If this domain (unless the domain in $*$) contains a label unknown to GUSS, a compilation error is triggered.

5.18.4 Implementation Details

This section describes some technical details that may provide useful insight in case of unexpected behavior.

Because GUSS changes all model parameters mentioned in the `dict` set to variables, a linear model can produce some non-linear instructions (e.g. `d(i,j)*f*x(i,j)` becomes a non-linear expression since `f` becomes a variable in the model instance given to GUSS). This also explains why some models compile without complaint, but if the model is used in the context of GUSS, the compile time check of the model will fail because a parameter that is turned into a variable can no longer be used in that way. For example, suppose the model contains a constraint `e(i).. sum(j$A(i,j), ...)`. If `A(i,j)` is a parameter in the regular model, the compiler will not complain, but if `A` becomes a parameter that shows up in the first position of a `param` tuple in the `dict` set, the GAMS compiler will turn `A` into a variable and complain that an endogenous variable cannot be used in a `$`-condition.

The sparsity pattern of a model can be greatly affected by GUSS. In a regular model instance GAMS will only generate and pass on non-zero matrix elements of a constraint `e(i).. sum(j, A(i,j)*x(j)) ...`, so the sparsity of `A` determines the sparsity of the generated model instance. GUSS allows to use this constraint with different values for `A` hence GUSS cannot exclude any of the pairs `(i,j)` and generate a dense matrix. The user can enforce some sparsity by explicitly restricting the `(i,j)` pairs: `e(i).. sum(ij(i,j), A(i,j)*x(j)) ...`

Attention

> While GUSS is available for many model types quadratic models require special attention. Linear solvers that have been extended to cover (convex) quadratic models, e.g. Cplex, Gurobi, Mosek, Xpress, do not work properly if the modifying parameter affects the left hand side of linear equations or any modifications of quadratic equations. Bound updates as well as changes of the right hand side of linear constraints are okay (see the example of the quadratic support vector machine below). Unfortunately, detecting if a quadratic model is okay or not for a given solver is at the moment difficult to detect, so use caution with quadratic models in combination with such (linear) solvers.

The actual change of the GAMS language required for the implementation of GUSS is minimal. The only true change is the extension of the `SOLVE` statement with the term `SCENARIO dict`. Existing language elements have been used to store symbol mapping information, options, and model result statistics. Some parts of the GUSS presentation look somewhat unnatural, e.g. since `dict` is a three dimensional `set` the specification the scenario `set` using keyword `scenario` requires a third dummy label `&prime;&prime;`. However, this approach gives maximum flexibility for future extension, allows reliable consistency checks at compile and execution time, and allows the user to delay the commitment for significant and permanent syntax changes of a developing method to handle model instances at a GAMS language level.

5.18.5 Applications

5.18.5.1 Cross Validation in GAMS via GUSS

Cross validation is a statistical/machine learning technique that aims to evaluate the generalizability of a classifier (or other decision) process. It does this by setting aside a portion of the data for testing, and uses the remaining data entries to produce the classifier. The testing data is subsequently used to evaluate how well the classifier works. Cross validation performs this whole process a number of times in order to estimate the true power of the classifier.

Ten-fold cross validation is a special case, where the original data is split into ten pieces, and cross validation is performed using each of these ten pieces as the testing set. Thus, the training process is performed ten times, each of which uses the data obtained by deleting the testing set from the whole dataset. We show below how to carry this out using the Gather-Update-Solve-Scatter (GUSS) facility in GAMS.

A paper with the title "GUSS: Solving Collections of Data Related Models within GAMS" that contains two additional application examples for GUSS is available here.

GUSS formulation in GAMS

The following example compares the two formulations for a feature-selection model under cross-validation using data files a_data.inc and b_data.inc . The actual source code for both of these GAMS formulations is available here.

Original GAMS formulation (without the GAMS/DEA interface):

```
$title Ten-fold cross validation example
$eolcom !

$setglobal num_folds 10

set a 'set for category 1'         /1*1505/
    b 'set for category 2'         /1*957/
    o 'observations'               /1*14/
    p 'folds to perform'           /1*%num_folds%/
    f 'maximum features to select' /1*10/

* Read in the data from the data files
parameter a_data(a, o) /
$offlisting
$include "a_data.inc"
$onlisting
/;

parameter b_data(b, o) /
$offlisting
$include "b_data.inc"
$onlisting
/;

set a_test(p,a), b_test(p,b) 'testing sets'
    a_trai(a),   b_trai(b)   'training sets';

* Define problem
scalar w_tol /1/
       features /6/;

positive variables a_err(a), sla(a)
                   b_err(b), slb(b);

variables c,
          weight(o),
          gamma;

binary variable y(o);


equations w_def1(o),
          w_def2(o),
          y_def,
          c_def,
          a_def(a),
          b_def(b);

w_def1(o)..
```

```
        weight(o) =l= w_tol*y(o);

w_def2(o)..
        weight(o) =g= -w_tol*y(o);

y_def..
        sum(o, y(o)) =e= features;

c_def..
        c =e= sum(a, a_err(a)) + sum(b, b_err(b));

a_def(a)..
        -sum(o, a_data(a, o)*weight(o)) + gamma + 1 =l= a_err(a) + sla(a);

b_def(b)..
         sum(o, b_data(b, o)*weight(o)) - gamma + 1 =l= b_err(b) + slb(b);


model train /all/;

$batinclude gentestset.inc "p,a" "p,b"

set headers 'report' / modelstat, solvestat, objval /;
parameter rep(p,headers);
train.optfile = 0;
option limrow=0, limcol=0, solprint=silent, mip=xpress,
       solvelink=%solveLink.loadLibrary%, optcr=0, optca=0;

$echo loadmipsol=1 > xpress.opt

loop(p,
  a_err.up(a) = inf; a_err.up(a)$a_test(p,a) = 0;
  b_err.up(b) = inf; b_err.fx(b)$b_test(p,b) = 0;
  sla.fx(a) = 0; sla.up(a)$a_test(p,a) = inf;
  slb.fx(b) = 0; slb.up(b)$b_test(p,b) = inf;
  solve train using mip minimizing c;
  train.optfile = 1; ! use mipstart for the second run
  rep(p,'modelstat') = train.modelstat;
  rep(p,'solvestat') = train.solvestat;
  rep(p,'objval') = train.objval;
);
display rep;
```

Options file for the original formulation: xpress.opt

```
 loadmipsol=1
```

The batinclude file gentestset.inc . gives instructions for generating the testing sets. It produces a_test and b_test that detail which equations are left out on solve p.

The actual model is set up to include all the data points in the equations a_def and b_def. To delete the equations that correspond to the test set, we introduce nonnegative slack variables into all the equations. We then set the upper bounds of the slack variables to zero in equations corresponding to the training set, and to infinity in equations corresponding to the testing set. At the same time we fix the error measures a_err and b_err belonging to the testing set by setting their upper bounds to zero. Thus the testing set equations are always satisfiable by choice of the slack variables alone - essentially they are discarded from

the model as required. An alternative formulation could "include" the data equations that you need in each scenario, but the update from one scenario to the next in the defining data is much larger.

Cross validation formulated using GUSS: This model essentially mimics what the standard model does, but the implementation of the solver loop behind the scenes is much more efficient, and the consquences are that are clear to see if you execute both model runs. The changes are in the last 40 lines of the GAMS code.

```
$title Ten-fold cross validation example
$eolcom !

$setglobal num_folds 10

set a 'set for category 1'         /1*1505/
    b 'set for category 2'         /1*957/
    o 'observations'               /1*14/
    p 'folds to perform'           /1*%num_folds%/
    f 'maximum features to select' /1*10/

* Read in the data from the data files
parameter a_data(a, o) /
$offlisting
$include "a_data.inc"
$onlisting
/;

parameter b_data(b, o) /
$offlisting
$include "b_data.inc"
$onlisting
/;

set a_test(p,a), b_test(p,b) 'testing sets'
    a_trai(a),   b_trai(b)   'training sets';

* Define problem
scalar w_tol /1/
       features /6/;

positive variables a_err(a), sla(a)
                   b_err(b), slb(b);

variables c,
          weight(o),
          gamma;

binary variable y(o);


equations w_def1(o),
          w_def2(o),
          y_def,
          c_def,
          a_def(a),
          b_def(b);

w_def1(o)..
        weight(o) =l= w_tol*y(o);
```

```
w_def2(o)..
        weight(o) =g= -w_tol*y(o);

y_def..
        sum(o, y(o)) =e= features;

c_def..
        c =e= sum(a, a_err(a)) + sum(b, b_err(b));

a_def(a)..
        -sum(o, a_data(a, o)*weight(o)) + gamma + 1 =l= a_err(a) + sla(a);

b_def(b)..
         sum(o, b_data(b, o)*weight(o)) - gamma + 1 =l= b_err(b) + slb(b);


model train /all/;
train.optfile = 1;

$batinclude gentestset.inc "p,a" "p,b"

parameter wval(p,o), gval(p);

set headers 'report' / modelstat, solvestat, objval /;
parameter
    scenrep(p,headers)
    scopt(*) / SkipBaseCase 1, Optfile 1, LogOption 2 /;

set dict / p.     scenario.''
           scopt. opt.     scenrep
           a_err. upper.   aupper
           b_err. upper.   bupper
           sla.   upper.   afree
           slb.   upper.   bfree
           weight.level.   wval
           gamma. level.   gval /

$echo loadmipsol=1 > xpress.opt

Parameter aupper(p,a), bupper(p,b), afree(p,a), bfree(p,b);

aupper(p,a)$(not a_test(p,a)) = inf;
bupper(p,b)$(not b_test(p,b)) = inf;

afree(p,a)$a_test(p,a) = inf;
bfree(p,b)$b_test(p,b) = inf;

option mip=xpress, optcr=0, optca=0;
solve train using mip minimizing c scenario dict;
display scenrep, gval;
```

Firstly, parameters `aupper`, `bupper`, `afree` and `bfree` are used to set the bounds on the error and slack variables in the testing set equations respectively. The setting of the upper bounds are governed by the syntax shown in the controlling set dict. Furthermore, the output of the classifier (`w,gamma`) for each fold of the cross validation uses the dict set to place results into the parameters `wval` and `gval` respectively. Finally, the GUSS options are used to guarantee that the subsequent solves are instructed to process solver

options (`Optfile 1`) which instruct the solver to use the previous solution to start the branch-and-cut process (`loadmipsol=1`).

The complete data and model files for this example are found in (galaxy zip archive). The data and model for a second instance based on the Wisconsin Diagnostic Breast Cancer Database is downloadable as (wdbc zip archive).

Quadratic Programs

GUSS is not limited to linear programs, but can be used more generally. Simple (indexed) quadratic models can be solved using GUSS. The following example illustrates the use of GUSS for quadratic programs. In this example, a support vector machine is used to determine a linear classifier that separates data into two categories. We use the following model:

$$\begin{array}{ll} \text{Min}_{w,g,z} & (1/2)\|w\|2^2 + Ce^T z \\ \text{subject to} & D(Aw - g) + z \geq 1 \\ & z \geq 0 \end{array} \tag{5.12}$$

Here, A is a matrix containing the training data (patients by features) and D is a diagonal matrix with values $+1$ or -1 (each denoting one of the two classes). C is a parameter weighting the importance of maximizing the margin between the classes $(2/\|w\|_2)$ versus minimizing the misclassification error (z). The solution w and g are used to define a separating hyperplane $\{x|w^T x = g\}$ to classify (unseen) data points.

As given, the standard linear support vector machine is not a slice model per se. It becomes a slice model under cross-validation training, where it is solved multiple times on different pieces of data. In this case, only the data A and D vary between solves, appropriately fitting the definition of a slice model.

The data for this example comes from the Wisconsin Diagnosis Breast Cancer Database, and is available here . The data was converted to the GAMS file wdbc.gms , which defines A and D. The actual source code for the following GAMS formulation is available here.

The GUSS formulation for quadratic svm:

```
$title Ten-fold cross validation example using GUSS
$eolcom !

$setglobal num_folds 10

set p /1*%num_folds%/;  ! folds to perform

! Read in data
$include "wdbc.gms"

set test(p,i);          ! testing set

! Define problem
parameter C /1/;
positive variables z(i);
variables obj, w(k), gamma, slack(i);

equations obj_def, sep_def(i);

obj_def..  obj =e= 1/2*sum(k, sqr(w(k))) + C*sum(i, z(i));
sep_def(i)..
```

```
          D(i)*(sum(k, A(i,k)*w(k)) - gamma) + z(i) + slack(i) =g= 1;

model train /all/;

! Generate testing sets (to be deleted in each problem)
loop(p,
$batinclude gentestset2.inc "p,i"
);

set headers report / modelstat, solvestat, objval /;
parameter
    scenrep(p,headers)
    scopt / SkipBaseCase 1, LogOption 2 /;

set dict / p.    scenario.''
           scopt.opt.     scenrep
           z.    upper.   iupper
           slack.upper.   ifree /;

Parameter iupper(p,i), ifree(p,i);
iupper(p,i)$(not test(p,i)) = inf;
ifree(p,i)$test(p,i)        = inf;

option qcp=conopt, optcr=0, optca=0;
solve train using qcp minimizing obj scenario dict;
display scenrep;
```

Because the problem is quadratic, we must use a quadratic program solver. The variable values for weight and gamma could be saved for later testing using the same method as detailed above for the linear case.

The batinclude file gentestset2.inc is very similar to gentestset.inc from the earlier cross-validation examples. In gentestset2.inc, though, only one set is being dealt with rather than two. The complete source GAMS code for this formulation is available in this zip archive.

5.18.5.2 DEA Modeling in GAMS via GUSS

Data Envelopment Analysis (DEA) models can be solved most efficiently in GAMS using the Gather-Update-Solve-Scatter (GUSS) facility. This is the preferred method since release 23.7 of GAMS and hence the GAMS/DEA solver is no longer available.

A paper with the title "GUSS: Solving Collections of Data Related Models within GAMS" that contains two additional application examples for GUSS is available here.

Introduction

The basic (CCR) DEA model is a collection of models indexed by k and defined by

$$\begin{array}{lll} \max_{u,v} & u^T Y_{*,k} & \text{(objective slice)} \\ \text{subject to} & v^T X_{*,k} = 1 & \text{(slice constraint)} \\ & u^T Y \le v^T X & \text{(core constraint)} \\ & u, v \ge 0 & \text{(core constraint)} \end{array} \tag{5.13}$$

where X, Y are data matrices.

Without using GUSS in GAMS, a model would be defined and solved in a loop over k, requiring the model to be generated multiple times with different instances for each value of k. GUSS is an alternative (and more efficient) way to define the individual programs and pass them to any underlying GAMS solver. In this way, individual programs are not re-generated, but are instead defined as data modifications of each other. This reduces overall model generation time. Further, previous solutions can be used as starting points in later solves to speed up overall processing time.

Some DEA examples compare the two formulations. The actual source code for both of these formulations is available here.

DEA Examples

Original GAMS formulation (without GUSS): In all these models the model setup and data are given at the top of the file and the code of interest is in the last 10 or so lines. In this setting we loop over the set k and change the data in the objective function and the first constraint of the model explicitly before each solve. We only output a minimal summary of the solution.

```
$title Data  Envelopment Analysis - DEA

$ontext
Data Envelopment Analysis (DEA) is a technique for measuring the relative
performance of organizational units where presence of multiple inputs and
outputs makes comparison difficult.

            efficiency = weighted sum of output / weighted sum of input

Find weights that maximize the efficiency for one unit while ensuring
that no other units has an efficiency < 1 using these weights. A primal
and dual formulation is presented.


Dyson, Thanassoulis, and Boussofiane, A DEA Tutorial.
Warwick Business School

$offtext

Sets  i     'units' / Depot1*Depot20 /
      j     'inputs and outputs' / stock, wages, issues, receipts, reqs /
      ji(j) 'inputs'             / stock, wages                         /
      jo(j)            'outputs' /               issues, receipts, reqs /;
alias(i,k);


Table data(i,j)
         stock  wages  issues  receipts  reqs
Depot1     3      5      40       55      30
Depot2     2.5    4.5    45       50      40
Depot3     4      6      55       45      30
Depot4     6      7      48       20      60
Depot5     2.3    3.5    28       50      25
Depot6     4      6.5    48       20      65
Depot7     7     10      80       65      57
Depot8     4.4    6.4    25       48      30
Depot9     3      5      45       64      42
Depot10    5      7      70       65      48
Depot11    5      7      45       65      40
```

```
Depot12    2      4      45       40      44
Depot13    5      7      65       25      35
Depot14    4      4      38       18      64
Depot15    2      3      20       50      15
Depot16    3      6      38       20      60
Depot17    7     11      68       64      54
Depot18    4      6      25       38      20
Depot19    3      4      45       67      32
Depot20    3      6      57       60      40
;

Parameter slice(j) 'slice of data'
          eff_k(i) 'efficiency report';

Positive variables v(ji) 'input weights'
                   u(jo) 'output weights';

Variable eff 'efficiency';


Equations defe    'efficiency definition - weighted output'
          denom   'weighted input'
          lime(i) 'output / input < 1';

defe..    eff =e= sum(jo, u(jo)*slice(jo));

denom..   sum(ji, v(ji)*slice(ji)) =e= 1;

lime(i).. sum(jo, u(jo)*data(i,jo)) =l= sum(ji, v(ji)*data(i,ji));

model dea /defe, denom, lime /;

set headers / modelstat, solvestat, objval /;
parameter rep(k,headers) 'solution report summary';
option limrow=0, limcol=0, solprint=silent,
       solvelink=%solveLink.loadLibrary%;
loop(k,
  slice(j) = data(k,j);
  solve dea using lp max eff;
  rep(k,'modelstat') = dea.modelstat;
  rep(k,'solvestat') = dea.solvestat;
  rep(k,'objval'   ) = dea.objval;
);
display rep;
```

The DEA problem formulated using GUSS: In this setting, the solve statement includes an extra keyword scenario that points to a new set called `dict`. The contents of this set are directives to GUSS that state the scenario index is k, the parameter slice is populated from the parameter data and the values of the variable `eff` are stored into the parameter `eff_k` for each scenario solved. More details follow below.

```
$title Data  Envelopment Analysis - DEA

Sets  i     'units' / Depot1*Depot20 /
      j     'inputs and outputs '/ stock, wages, issues, receipts, reqs /
      ji(j) 'inputs'             / stock, wages                         /
      jo(j)            'outputs' /               issues, receipts, reqs /;
alias(i,k);
```

```
Table data(i,j)
         stock  wages  issues  receipts  reqs
Depot1     3      5      40       55      30
Depot2     2.5    4.5    45       50      40
Depot3     4      6      55       45      30
Depot4     6      7      48       20      60
Depot5     2.3    3.5    28       50      25
Depot6     4      6.5    48       20      65
Depot7     7     10      80       65      57
Depot8     4.4    6.4    25       48      30
Depot9     3      5      45       64      42
Depot10    5      7      70       65      48
Depot11    5      7      45       65      40
Depot12    2      4      45       40      44
Depot13    5      7      65       25      35
Depot14    4      4      38       18      64
Depot15    2      3      20       50      15
Depot16    3      6      38       20      60
Depot17    7     11      68       64      54
Depot18    4      6      25       38      20
Depot19    3      4      45       67      32
Depot20    3      6      57       60      40
;

Parameter slice(j) 'slice of data'
          eff_k(i) 'efficiency report';

Positive variables v(ji) 'input weights'
                   u(jo) 'output weights';

Variable eff 'efficiency';


Equations defe    'efficiency definition - weighted output'
          denom   'weighted input'
          lime(i) 'output / input < 1';

defe..    eff =e= sum(jo, u(jo)*slice(jo));

denom..   sum(ji, v(ji)*slice(ji)) =e= 1;

lime(i).. sum(jo, u(jo)*data(i,jo)) =l= sum(ji, v(ji)*data(i,ji));

model dea /defe, denom, lime /;

set headers 'report' / modelstat, solvestat, objval /;
parameter scenrep(k,headers) 'solution report summary'
          scopt / SkipBaseCase 1 /;

set dict / k      .scenario.''
           slice  .param.   data
           eff    .level.   eff_k
           scopt  .opt.     scenrep /;

slice(j) = 0; option lp=cplex;
solve dea using lp max eff scenario dict;
```

```
display scenrep,eff_k;
```

In the GUSS version we indicate the collection of models to be solved using the set `dict`. The first element of `dict` determines the set to be used for the scenario (collection) index, in this case k. The second element of dict then details that in each scenario k, the parameter slice is instantiated using a slice of the parameter data. Essentially, this corresponds to the GAMS statement:

slice(j) = data(k,j)

Note the scenario index k must appear as the first index of the parameter data. The third element of `dict` allows the modeler to collect information from each solve and store it into a GAMS parameter. Essentially, the third element of `dict` corresponds to the GAMS statement:

eff_k(k) = eff.l

that gets executed immediately after the solve of scenario k.

More complex scenario models can also be formulated using GUSS, including multiple equations being updated. This is shown by the dual of the basic DEA model, given by

$$\begin{array}{lcl} \min_{z,\lambda} & z & \text{(objective)} \\ \text{subject to} & X^*\lambda \leq zX_{*,k} & \text{(slice constraint)} \\ & Y^*\lambda \geq Y_{*,k} & \text{(slice constraint)} \\ & \lambda \geq 0 & \text{(core constraint)} \end{array} \tag{5.14}$$

The next example compares the two formulations for this model. The actual source code for both of these formulations is available here.

Original GAMS formulation (without GUSS):

```
$title Data Envelopment Analysis - DEA (traditional)

sets  i  'units' / Depot1*Depot20 /
      j     'inputs and outputs' / stock, wages, issues, receipts, reqs /
      ji(j) 'inputs'             / stock, wages                         /
      jo(j)            'outputs' /               issues, receipts, reqs /;
alias(k,i);

Table data(i,j)
         stock  wages  issues  receipts  reqs
Depot1     3      5      40       55      30
Depot2     2.5    4.5    45       50      40
Depot3     4      6      55       45      30
Depot4     6      7      48       20      60
Depot5     2.3    3.5    28       50      25
Depot6     4      6.5    48       20      65
Depot7     7     10      80       65      57
Depot8     4.4    6.4    25       48      30
Depot9     3      5      45       64      42
Depot10    5      7      70       65      48
Depot11    5      7      45       65      40
Depot12    2      4      45       40      44
Depot13    5      7      65       25      35
Depot14    4      4      38       18      64
Depot15    2      3      20       50      15
Depot16    3      6      38       20      60
```

```
Depot17    7     11      68       64      54
Depot18    4      6      25       38      20
Depot19    3      4      45       67      32
Depot20    3      6      57       60      40
;

parameter slice(j) 'slice of data'
          eff_k(i) 'efficiency report';

Variables z      'efficiency'
          lam(i) 'dual weights';

positive variables lam;

Equations dii(ji) 'input duals'
          dio(jo) 'output dual';

*  dual model

dii(ji).. sum(i, lam(i)*data(i,ji)) =l= z*slice(ji);

dio(jo).. sum(i, lam(i)*data(i,jo)) =g=     slice(jo);

model deadc dual with CRS / dii, dio /;

parameter rep 'summary report';
option limrow=0, limcol=0, solprint=silent, lp=cplex,
       solvelink=%solveLink.loadLibrary%;

loop(k,
   slice(j) = data(k,j);
   solve deadc using lp minimizing z ;
   rep(k,'modelstat') = deadc.modelstat;
   rep(k,'solvestat') = deadc.modelstat;
   rep(k,'objval') = deadc.objval;
);

display rep;
```

Dual (CRS) DEA model formulated using GUSS: The key modeling statements occur in the last 10 lines below.

```
$title Data Envelopment Analysis - DEA (dual,GUSS)

sets  i  'units' / Depot1*Depot20 /
      j     'inputs and outputs' / stock, wages, issues, receipts, reqs /
      ji(j) 'inputs'             / stock, wages                         /
      jo(j)            'outputs' /               issues, receipts, reqs /;
alias(k,i);

Table data(i,j)
         stock  wages  issues  receipts  reqs
Depot1     3      5      40       55      30
Depot2     2.5    4.5    45       50      40
Depot3     4      6      55       45      30
Depot4     6      7      48       20      60
Depot5     2.3    3.5    28       50      25
```

```
Depot6     4      6.5    48       20       65
Depot7     7     10      80       65       57
Depot8     4.4    6.4    25       48       30
Depot9     3      5      45       64       42
Depot10    5      7      70       65       48
Depot11    5      7      45       65       40
Depot12    2      4      45       40       44
Depot13    5      7      65       25       35
Depot14    4      4      38       18       64
Depot15    2      3      20       50       15
Depot16    3      6      38       20       60
Depot17    7     11      68       64       54
Depot18    4      6      25       38       20
Depot19    3      4      45       67       32
Depot20    3      6      57       60       40
;

parameter slice(j) 'slice of data'
          eff_k(i) 'efficiency report';

Variables z      'efficiency'
          lam(i) 'dual weights';

positive variables lam;

Equations dii(ji) 'input duals'
          dio(jo) 'output dual';

*  dual model

dii(ji).. sum(i, lam(i)*data(i,ji)) =l= z*slice(ji);

dio(jo).. sum(i, lam(i)*data(i,jo)) =g=     slice(jo);

model deadc dual with CRS / dii, dio /;

set headers 'report' / modelstat, solvestat, objval /;
parameter scenrep(k,headers) 'solution report summary'
          scopt / SkipBaseCase 1 /;

set dict / k.     scenario.''
           scopt. opt.     scenrep
           slice. param.   data
           z.     level.   eff_k   /;

slice(j) = 0; option lp=cplex;
solve deadc using lp min z scenario dict;
display scenrep,eff_k;
```

Extensions of these models to formulations with weighted outputs or variable returns to scale are easy to formulate with the scenario solver within GAMS. This extended model can be downloaded here.

The **DEA model** in the model library is similar to the extended model, but does not make use of GUSS.

5.19 HiGHS

HiGHS is an optimization package for solving continuous and mixed-integer linear programming problems

(LPs and MIPs) using simplex, interior-point, and branch-and-cut algorithms. HiGHS is developed by the Edinburgh Research Group in Optimization.

For more detailed information on the implemented simplex method, we refer to [98].

5.19.1 Usage

The following statement can be used inside your GAMS program to specify using HiGHS

```
Option MIP = HIGHS;     { or LP or RMIP }
```

The above statement should appear before the Solve statement. If HiGHS was specified as the default solver during GAMS installation, the above statement is not necessary.

5.19.1.1 Specification of HiGHS Options

GAMS/HiGHS supports the GAMS parameters reslim, iterlim, nodlim. optca, optcr, cutoff, and threads.

Options can be specified by a HiGHS options file. A HiGHS options file consists of one option or comment per line. A pound sign (#) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs).

A small example for a highs.opt file is:

```
solver = ipm
ipm_optimality_tolerance = 1e-6
run_crossover = false
```

It causes HiGHS to use an interior point solver for an LP solve, increases the optimality tolerance to 10^{-6}, and turns off crossover to a basis solution.

5.19.2 List of HiGHS Options

In the following, we give a detailed list of all HiGHS options.

Option	Description	Default
dual_feasibility_tolerance	Dual feasibility tolerance Range: [1e-10, ∞]	1e-07
infinite_bound	Limit on |constraint bound|: values greater than or equal to this will be treated as infinite Range: [1e+15, ∞]	1e+20
infinite_cost	Limit on |cost coefficient|: values greater than or equal to this will be treated as infinite Range: [1e+15, ∞]	1e+20
ipm_iteration_limit	Iteration limit for IPM solver Range: {0, ..., ∞}	GAMS iterlim

Option	Description	Default
ipm_optimality_tolerance	IPM optimality tolerance Range: [1e-12, ∞]	1e-08
large_matrix_value	Upper limit on |matrix entries|: values greater than or equal to this will be treated as infinite Range: [1, ∞]	1e+15
mip_abs_gap	Tolerance on absolute gap of MIP, |ub-lb|, to determine whether optimality has been reached for a MIP instance Range: [0, ∞]	GAMS optca
mip_detect_symmetry	Whether MIP symmetry should be detected Range: boolean	1
mip_feasibility_tolerance	MIP feasibility tolerance Range: [1e-10, ∞]	1e-06
mip_heuristic_effort	Effort spent for MIP heuristics Range: [0, 1]	0.05
mip_lp_age_limit	Maximal age of dynamic LP rows before they are removed from the LP relaxation in the MIP solver Range: {0, ..., 32767}	10
mip_max_improving_sols	Limit on the number of improving solutions found to stop the MIP solver prematurely Range: {1, ..., ∞}	∞
mip_max_leaves	MIP solver max number of leave nodes Range: {0, ..., ∞}	∞
mip_max_nodes	MIP solver max number of nodes Range: {0, ..., ∞}	GAMS nodlim, if > 0, ∞ otherwise
mip_max_stall_nodes	MIP solver max number of nodes where estimate is above cutoff bound Range: {0, ..., ∞}	∞
mip_min_cliquetable_entries_for_parallelism	Minimal number of entries in the MIP solver cliquetable before neighbourhood queries of the conflict graph use parallel processing Range: {0, ..., ∞}	100000
mip_min_logging_interval	MIP minimum logging interval Range: [0, ∞]	5
mip_pool_age_limit	Maximal age of rows in the MIP solver cutpool before they are deleted Range: {0, ..., 1000}	30
mip_pool_soft_limit	Soft limit on the number of rows in the MIP solver cutpool for dynamic age adjustment Range: {1, ..., ∞}	10000

Option	Description	Default
mip_pscost_minreliable	Minimal number of observations before MIP solver pseudo costs are considered reliable Range: {0, ..., ∞}	8
mip_rel_gap	Tolerance on relative gap, |ub-lb|/|ub|, to determine whether optimality has been reached for a MIP instance Range: [0, ∞]	GAMS optcr
mipstart	Whether to pass initial level values as starting point to MIP solver If the solution is not feasible, HiGHS will solve the LP obtained from fixing all discrete variables to their initial level values. Range: boolean	0
objective_bound	Objective bound for termination of the dual simplex solver Range: real	GAMS cutoff
objective_target	Objective target for termination of the MIP solver Range: real	-∞
output_flag	Enables or disables solver output Range: boolean	0, if GAMS logoption = 0, otherwise 1
parallel	Parallel option: "off", "choose" or "on" Range: string	choose
presolve	Presolve option: "off", "choose" or "on" Range: string	choose
primal_feasibility_tolerance	Primal feasibility tolerance Range: [1e-10, ∞]	1e-07
random_seed	Random seed used in HiGHS Range: {0, ..., ∞}	0
run_crossover	Run IPM crossover: "off", "choose" or "on" Range: string	on
sensitivity	Whether to run sensitivity analysis after solving an LP with a simplex method Range: boolean	0
simplex_dual_edge_weight_strategy	Strategy for simplex dual edge weights: Choose / Dantzig / Devex / Steepest Edge (-1/0/1/2) Range: {-1, ..., 2}	-1
simplex_iteration_limit	Iteration limit for simplex solver when solving LPs, but not subproblems in the MIP solver Range: {0, ..., ∞}	GAMS iterlim
simplex_max_concurrency	Maximum level of concurrency in parallel simplex Range: {1, ..., 8}	8

Option	Description	Default
simplex_primal_edge_weight_strategy	Strategy for simplex primal edge weights: Choose / Dantzig / Devex / Steepest Edge (-1/0/1/2) Range: {-1, ..., 2}	-1
simplex_scale_strategy	Simplex scaling strategy: off / choose / equilibration / forced equilibration / max value 0 / max value 1 (0/1/2/3/4/5) Range: {0, ..., 5}	1
simplex_strategy	Strategy for simplex solver 0 => Choose; 1 => Dual (serial); 2 => Dual (PAMI); 3 => Dual (SIP); 4 => Primal Range: {0, ..., 4}	1
simplex_update_limit	Limit on the number of simplex UPDATE operations Range: {0, ..., ∞}	5000
small_matrix_value	Lower limit on |matrix entries|: values less than or equal to this will be treated as zero Range: [1e-12, ∞]	1e-09
solution_file	Solution file Range: string	<inputname>.sol
solver	LP algorithm to run: "simplex", "choose", or "ipm"; ignored for MIP Range: string	choose
solvetrace	Name of file for writing solving progress information during MIP solve Range: string	
solvetracenodefreq	Frequency in number of nodes for writing to solve trace file Range: {0, ..., ∞}	100
solvetracetimefreq	Frequency in seconds for writing to solve trace file Range: [0, ∞]	5
threads	Number of threads used by HiGHS (0: automatic) Range: {0, ..., ∞}	GAMS threads
time_limit	Time limit (seconds) Range: [0, ∞]	GAMS reslim
write_model_file	Write model file Range: string	<inputname>.lp
write_model_to_file	Write the model to a file Range: boolean	0

Option	Description	Default
write_solution_style	Style of solution file (raw = computer-readable, pretty = human-readable): -1 => HiGHS old raw (deprecated); 0 => HiGHS raw; 1 => HiGHS pretty; 2 => Glpsol raw; 3 => Glpsol pretty; 4 => HiGHS sparse raw Range: {-1, ..., 4}	0
write_solution_to_file	Write the primal and dual solution to a file Range: boolean	0
Options for expert users		
allow_unbounded_or_infeasible	whether to spend extra effort to distinguish unboundedness and infeasibility if necessary Range: boolean	0
allowed_cost_scale_factor	Largest power-of-two factor permitted when scaling the costs Range: {0, ..., 20}	0
allowed_matrix_scale_factor	Largest power-of-two factor permitted when scaling the constraint matrix Range: {0, ..., 30}	20
cost_scale_factor	Scaling factor for costs Range: {-20, ..., 20}	0
dual_simplex_cost_perturbation_multiplier	Dual simplex cost perturbation multiplier: 0 => no perturbation Range: [0, ∞]	1
dual_simplex_pivot_growth_tolerance	Dual simplex pivot growth tolerance Range: [1e-12, ∞]	1e-09
dual_steepest_edge_weight_error_tolerance	Tolerance on dual steepest edge weight errors Range: [0, ∞]	∞
dual_steepest_edge_weight_log_error_threshold	Threshold on dual steepest edge weight errors for Devex switch Range: [1, ∞]	10
factor_pivot_threshold	Matrix factorization pivot threshold Range: [0.0008, 0.5]	0.1
factor_pivot_tolerance	Matrix factorization pivot tolerance Range: [0, 1]	1e-10
highs_analysis_level	Analysis level in HiGHS Range: {0, ..., 63}	0
highs_debug_level	Debugging level in HiGHS Range: {0, ..., 3}	0
icrash	Run iCrash Range: boolean	0
icrash_approx_iter	iCrash approximate minimization iterations Range: {0, ..., 100}	50

Option	Description	Default
icrash_breakpoints	Exact subproblem solution for iCrash Range: boolean	0
icrash_dualize	Dualize strategy for iCrash Range: boolean	0
icrash_exact	Exact subproblem solution for iCrash Range: boolean	0
icrash_iterations	iCrash iterations Range: {0, ..., 200}	30
icrash_starting_weight	iCrash starting weight Range: [1e-10, 1e+50]	0.001
icrash_strategy	Strategy for iCrash Range: string	ICA
ipx_dualize_strategy	Strategy for dualizing before IPX Range: {-1, ..., 3}	2
less_infeasible_DSE_check	Check whether LP is candidate for LiDSE Range: boolean	1
less_infeasible_DSE_choose_row	Use LiDSE if LP has right properties Range: boolean	1
log_dev_level	Output development messages: 0 => none; 1 => info; 2 => verbose Range: {0, ..., 3}	0
lp_presolve_requires_basis_postsolve	Prevents LP presolve steps for which postsolve cannot maintain a basis Range: boolean	1
max_dual_simplex_cleanup_level	Max level of dual simplex cleanup Range: {0, ..., ∞}	1
max_dual_simplex_phase1_cleanup_level	Max level of dual simplex phase 1 cleanup Range: {0, ..., ∞}	2
mip_report_level	MIP solver reporting level Range: {0, ..., 2}	1
no_unnecessary_rebuild_refactor	No unnecessary refactorization on simplex rebuild Range: boolean	1
presolve_pivot_threshold	Matrix factorization pivot threshold for substitutions in presolve Range: [0.0008, 0.5]	0.01
presolve_reduction_limit	Limit on number of presolve reductions -1 => no limit Range: {-1, ..., ∞}	-1
presolve_rule_logging	Log effectiveness of presolve rules for LP Range: boolean	0

Option	Description	Default
presolve_rule_off	Bit mask of presolve rules that are not allowed Range: {0, ..., ∞}	0
presolve_substitution_maxfillin	Maximal fillin allowed for substitutions in presolve Range: {0, ..., ∞}	10
primal_simplex_bound_perturbation_multiplier	Primal simplex bound perturbation multiplier: 0 => no perturbation Range: [0, ∞]	1
rebuild_refactor_solution_error_tolerance	Tolerance on solution error when considering refactorization on simplex rebuild Range: real	1e-08
simplex_crash_strategy	Strategy for simplex crash: off / LTSSF / Bixby (0/1/2) Range: {0, ..., 9}	0
simplex_dualize_strategy	Strategy for dualizing before simplex Range: {-1, ..., 1}	-1
simplex_initial_condition_check	Perform initial basis condition check in simplex Range: boolean	1
simplex_initial_condition_tolerance	Tolerance on initial basis condition in simplex Range: [1, ∞]	1e+14
simplex_min_concurrency	Minimum level of concurrency in parallel simplex Range: {1, ..., 8}	1
simplex_permute_strategy	Strategy for permuting before simplex Range: {-1, ..., 1}	-1
simplex_price_strategy	Strategy for PRICE in simplex Range: {0, ..., 3}	3
simplex_unscaled_solution_strategy	Strategy for solving unscaled LP in simplex Range: {0, ..., 2}	1
start_crossover_tolerance	Tolerance to be satisfied before IPM crossover will start Range: [1e-12, ∞]	1e-08
use_implied_bounds_from_presolve	Use relaxed implied bounds from presolve Range: boolean	0
use_original_HFactor_logic	Use original HFactor logic for sparse vs hyper-sparse TRANs Range: boolean	1

5.20 IPOPT and IPOPTH

COIN-OR IPOPT (**I**nterior **P**oint **Opt**imizer) is an open-source solver for large-scale nonlinear programming (NLP). The code has been written primarily by Andreas Wächter.

IPOPT implements an interior point line search filter method for nonlinear programming models which functions can be nonconvex, but should be twice continuously differentiable. For more information on the algorithm we refer to [144] [196] [195] [197] [194] and the IPOPT documentation. Most of the IPOPT documentation in the section was taken from the IPOPT manual [104] .

5.20.1 Available linear solvers

The performance and robustness of IPOPT on larger models heavily relies on the used solver for sparse symmetric indefinite linear systems.

GAMS/IPOPT includes the sparse solver MUMPS [8] [9] (currently the default), and MKL PARDISO [166] [167]. The latter is not available for systems on ARM64 CPUs. In the commerically licensed GAMS/IPOPTH version, also the Harwell Subroutine Library (HSL) solvers MA27, MA57, HSL_MA86, and HSL_MA97 are available and MA27 is used by default.

MUMPS, MA57, HSL_MA86, and HSL_MA97 use METIS for matrix ordering [103], see also the METIS manual . METIS is copyrighted by the regents of the University of Minnesota.

IPOPT and IPOPTH can exploit parallelization of the linear solvers MKL Pardiso, HSL MA86, and HSL MA97 and the linear algebra routines (see next section).

The linear solver is chosen by the linear_solver option. Benchmarks have shown that MA57 and HSL_MA97 are often able to outperform MA27 on larger instances. Further, PARDISO often allows for performance that is better than MUMPS and similar to the HSL solvers. If IPOPT fails to solve an instance with PARDISO, it's worth to try changing the options pardisomkl_order and pardisomkl_max_iterative_refinement_steps.

It is also possible to use the linear solver PARDISO from the PARDISO Solver Project or the HSL routines with GAMS/IPOPT if a user provides libraries that can be loaded at runtime. PARDISO from the PARDISO Solver Project can provide performance that exceeds the one of PARDISO from Intel MKL. To build the HSL routines, COIN-OR project ThirdParty-HSL may be useful. See also options linear_solver, linear_system_scaling, nlp_scaling_method, pardisolib, and hsllib. Note that it is your responsibility to ensure that you are entitled to download and use these routines!

5.20.2 The linear algebra library

On systems for AMD and Intel CPUs, the IPOPT library distributed by GAMS and most of the linear solvers used by IPOPT use the Intel oneAPI Math Kernel Library (MKL), which provides a fast and parallel implementation of linear algebra routines (BLAS/LAPACK) and the linear solver PARDISO. MKL chooses an internal code path that provides the best possible performance for the used CPU type. As a consequence, results can be different when changing from one CPU to another. By setting an environment variable, the code path to use can be set by the user. See the Intel MKL documentation regarding Conditional Numerical Reproducibility for more details.

Intel MKL has been optimized for use with Intel CPUs. On CPUs from other vendors, MKL may not use an internal code path that could provide a better performance on that CPU. For example, it may not use AVX2 instructions on an AMD CPU that provides AVX2. However, Intel recently started to add optimized code for AMD's Zen CPUs.

To gain more insight into the use of MKL in GAMS/IPOPT, one may set the environment variable MKL_VERBOSE to 1. This will print out information about the MKL library used, functions being called, time spend there, etc.

On the GAMS system for macOS on ARM64 CPUs, the Apple Accelerate framework is used as linear algebra library.

5.20.3 Usage

The following statement can be used inside your GAMS program to specify using IPOPT

```
Option NLP = IPOPT;      { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP, CNS }
```

The above statement should appear before the Solve statement. If IPOPT was specified as the default solver during GAMS installation, the above statement is not necessary.

To use IPOPTH, the statement should be

```
Option NLP = IPOPTH;     { or LP, RMIP, DNLP, RMINLP, QCP, RMIQCP, CNS }
```

5.20.3.1 Specification of Options

IPOPT has many options that can be adjusted for the algorithm (see Section List of IPOPT Options). Options are all identified by a string name, and their values can be of one of three types: Number (real), Integer, or String. Number options are used for things like tolerances, integer options are used for things like maximum number of iterations, and string options are used for setting algorithm details, like the NLP scaling method. Options can be set by creating a `ipopt.opt` file in the directory you are executing IPOPT.

The `ipopt.opt` file is read line by line and each line should contain the option name, followed by whitespace, and then the value. Comments can be included with the `#` symbol. For example, the following is a valid `ipopt.opt` file:

```
# This is a comment

# Turn off the NLP scaling
nlp_scaling_method none

# Change the initial barrier parameter
mu_init 1e-2

# Set the max number of iterations
max_iter 500
```

GAMS/IPOPT understands currently the following GAMS parameters: reslim (time limit), iterlim (iteration limit), domlim (domain violation limit). Further the option threads can be used to control the number of threads used in the linear algebra routines and the linear solver. Setting `threads=0` currently does not enable multithreaded linear algebra.

5.20.3.2 Warmstarting IPOPT

As an interior point solver, it is difficult to warm start IPOPT. By default, only the level values of the variables are passed as starting point to IPOPT. Setting the IPOPT option warm_start_init_point to `yes` enables that also dual values for variables and constraints are passed to IPOPT.

However, the expected behavior that IPOPT finishes within one iteration if optimal primal and dual values are passed is not reached this way, yet. This is, because IPOPT by default moves any initial value that is close to a bound into the interior. The amount on how much the initial point is moved can be controlled by various `bound_push` and `bound_frac` options. To make IPOPT accept an optimal primal/dual solution within one iteration, it should be sufficient to set the following options:

```
warm_start_init_point       yes
warm_start_bound_push       1e-9
warm_start_bound_frac       1e-9
warm_start_slack_bound_frac 1e-9
warm_start_slack_bound_push 1e-9
warm_start_mult_bound_push  1e-9
```

Further, it can be useful to specify that IPOPT can use a less central path in its first iterations by reducing the value of option mu_init. This option is only used if option mu_strategy is set to "monotone", so the option file entries would be

```
mu_strategy monotone
mu_init     0.0001
```

Finally, IPOPT by default checks whether it should scale the problem. The computed scaling depends on the starting point, which can be undesired when warmstarting. Thus, it may be useful to turn off scaling via option nlp_scaling_method:

```
nlp_scaling_method none
```

If a modified but structually equivalent problem instance is solved, e.g., via GUSS, option warm_start_init_point is automatically set to "yes" for every solve following the first one. If this is not desired, warm_start_init_point should explicitly be set to "no" in an IPOPT options file.

5.20.4 Output

This section describes the standard IPOPT console output. The output is designed to provide a quick summary of each iteration as IPOPT solves the problem.

Before IPOPT starts to solve the problem, it displays the problem statistics (number of nonzero-elements in the matrices, number of variables, etc.). Note that if you have fixed variables (both upper and lower bounds are equal), IPOPT may remove these variables from the problem internally and not include them in the problem statistics.

Following the problem statistics, IPOPT will begin to solve the problem and you will see output resembling the following,

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1.6109693e+01 1.12e+01 5.28e-01   0.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1  1.8029749e+01 9.90e-01 6.62e+01   0.1 2.05e+00    -  2.14e-01 1.00e+00f  1
   2  1.8719906e+01 1.25e-02 9.04e+00  -2.2 5.94e-02   2.0 8.04e-01 1.00e+00h  1
```

and the columns of output are defined as

item

The current iteration count. This includes regular iterations and iterations while in restoration phase. If the algorithm is in the restoration phase, the letter `r` will be appended to the iteration number.

objective

The unscaled objective value at the current point. During the restoration phase, this value remains the unscaled objective value for the original problem.

inf_pr

The unscaled constraint violation at the current point. This quantity is the infinity-norm (max) of the (unscaled) constraint violation. During the restoration phase, this value remains the constraint violation of the original problem at the current point. The option inf_pr_output can be used to switch to the printing of a different quantity. During the restoration phase, this value is the primal infeasibility of the original problem at the current point.

inf_du

The scaled dual infeasibility at the current point. This quantity measure the infinity-norm (max) of the internal dual infeasibility (Eq. (4a) in [197]), including inequality constraints reformulated using slack variables and problem scaling. During the restoration phase, this is the value of the dual infeasibility for the restoration phase problem.

lg(mu)

$\log_{10}$ of the value of the barrier parameter μ.

||d||

The infinity norm (max) of the primal step (for the original variables `x` and the internal slack variables `s`). During the restoration phase, this value includes the values of additional variables, `p` and `n` in Eq. (10) of [197] .

lg(rg)

$\log_{10}$ of the value of the regularization term for the Hessian of the Lagrangian in the augmented system (δ_w in Eq. (26) of [197]). A dash (`-`) indicates that no regularization was done.

alpha_du

The stepsize for the dual variables (α_k^z in Eq. (14c) of [197]).

alpha_pr

The stepsize for the primal variables (α_k in Eq. (14a) of [197]). The number is usually followed by a character for additional diagnostic information regarding the step acceptance criterion:

- `f`: f-type iteration in the filter method w/o second order correction
- `F`: f-type iteration in the filter method w/ second order correction
- `h`: h-type iteration in the filter method w/o second order correction
- `H`: h-type iteration in the filter method w/ second order correction
- `k`: penalty value unchanged in merit function method w/o second order correction

- `K`: penalty value unchanged in merit function method w/ second order correction
- `n`: penalty value updated in merit function method w/o second order correction
- `N`: penalty value updated in merit function method w/ second order correction
- `R`: Restoration phase just started
- `w`: in watchdog procedure
- `s`: step accepted in soft restoration phase
- `t`/`T`: tiny step accepted without line search
- `r`: some previous iterate restored

ls

The number of backtracking line search steps (does not include second-order correction steps).

Note that the step acceptance mechanisms in IPOPT consider the barrier objective function (Eq. (3a) in [197]) which is usually different from the value reported in the `objective` column. Similarly, for the purposes of the step acceptance, the constraint violation is measured for the internal problem formulation, which includes slack variables for inequality constraints and potentially scaling of the constraint functions. This value, too, is usually different from the value reported in `inf_pr`. As a consequence, a new iterate might have worse values both for the objective function and the constraint violation as reported in the iteration output, seemingly contradicting globalization procedure.

When the algorithm terminates, IPOPT will output a message to the screen. The following is a list of the possible output messages and a brief description.

Optimal Solution Found.

This message indicates that IPOPT found a (locally) optimal point within the desired tolerances.

Solved To Acceptable Level.

This indicates that the algorithm did not converge to the "desired" tolerances, but that it was able to obtain a point satisfying the "acceptable" tolerance level as specified by `acceptable-`$*$ options. This may happen if the desired tolerances are too small for the current problem.

Feasible point for square problem found.

This message is printed if the problem is "square" (i.e., it has as many equality constraints as free variables) and IPOPT found a point that is feasible w.r.t. constr_viol_tol. It may, however, not be feasible w.r.t. tol.

Converged to a point of local infeasibility. Problem may be infeasible.

The restoration phase converged to a point that is a minimizer for the constraint violation (in the ℓ_1-norm), but is not feasible for the original problem. This indicates that the problem may be infeasible (or at least that the algorithm is stuck at a locally infeasible point). The returned point (the minimizer of the constraint violation) might help you to find which constraint is causing the problem. If you believe that the NLP is feasible, it might help to start the optimization from a different point.

Search Direction is becoming Too Small.

This indicates that IPOPT is calculating very small step sizes and making very little progress. This could happen if the problem has been solved to the best numerical accuracy possible given the current scaling.

Iterates divering; problem might be unbounded.

This message is printed if the max-norm of the iterates becomes larger than the value of the option diverging_iterates_tol. This can happen if the problem is unbounded below and the iterates are diverging.

Stopping optimization at current point as requested by user.

This message is printed if either an interrupt signal was received (e.g., Ctrl+C was pressed) or the domain violation limit is reached.

Maximum Number of Iterations Exceeded.

This indicates that IPOPT has exceeded the maximum number of iterations as specified by the IPOPT option max_iter or the GAMS option iterlim.

Maximum wallclock time exceeded.

This indicates that IPOPT has exceeded the maximum number of wallclock seconds as specified by the IPOPT option max_wall_time or the GAMS option reslim.

Maximum CPU time exceeded.

This indicates that IPOPT has exceeded the maximum number of CPU seconds as specified by the IPOPT option max_cpu_time.

Restoration Failed!

This indicates that the restoration phase failed to find a feasible point that was acceptable to the filter line search for the original problem. This could happen if the problem is highly degenerate, does not satisfy the constraint qualification, or if an external or extrinsic function in GAMS provides incorrect derivative information.

Error in step computation!

This messages is printed if IPOPT is unable to compute a step towards a new iterate and the current iterate is not acceptable for the specified tolerances.

A possible reason is that a search direction could not be computed despite several attempts to modify the iteration matrix. Usually, the value of the regularization parameter then becomes too large.

Another reason is that the feasibility restoration phase could not be activated because the current iterate is not infeasible. Reasons for this again include that the problem is highly degenerate, badly scaled, or does not satisfy the constraint qualification. Before IPOPT 3.14, this resulted in a Restoration_Failed status code with message "Restoration phase is called at almost feasible point...".

Problem has too few degrees of freedom.

This indicates that your problem, as specified, has too few degrees of freedom. This can happen if you have too many equality constraints, or if you fix too many variables (IPOPT removes fixed variables by default, see also the option fixed_variable_treatment).

Not enough memory.

An error occurred while trying to allocate memory. The problem may be too large for your current memory and swap configuration. Sometimes it can help to choose a different linear solver.

INTERNAL ERROR: Unknown SolverReturn value - Notify IPOPT Authors.

An unknown internal error has occurred. Please notify the authors of the GAMS/IPOPT link or IPOPT (refer to support@gams.com).

5.20.4.1 Diagnostic Tags for IPOPT

To print additional diagnostic tags for each iteration of IPOPT, set the options print_info_string to `yes`. With this, a tag will appear at the end of an iteration line with the following diagnostic meaning that are useful to flag difficulties for a particular IPOPT run. The following is a list of possible strings:

- `!`: Tighten resto tolerance if only slightly infeasible, see Sec. 3.3 in [197]
- `A`: Current iteration is acceptable (alternate termination)
- `a`: Perturbation for PD Singularity can't be done, assume singular, see Sec. 3.1 in [197]
- `C`: Second Order Correction taken, see Sec. 2.4 in [197]
- `Dh`: Hessian degenerate based on multiple iterations, see Sec. 3.1 in [197]
- `Dhj`: Hessian/Jacobian degenerate based on multiple iterations, see Sec. 3.1 in [197]
- `Dj`: Jacobian degenerate based on multiple iterations, see Sec. 3.1 in [197]
- `dx`: δ_x perturbation too large, see Sec. 3.1 in [197]
- `e`: Cutting back α due to evaluation error (in backtracking line search)
- `F-`: Filter should be reset, but maximal resets exceeded, see Sec. 2.3 in [197]
- `F+`: Resetting filter due to last few rejections of filter, see Sec. 2.3 in [197]
- `L`: Degenerate Jacobian, δ_c already perturbed, see Sec. 3.1 in [197]
- `l`: Degenerate Jacobian, δ_c perturbed, see Sec. 3.1 in [197]
- `M`: Magic step taken for slack variables (in backtracking line search)
- `Nh`: Hessian not yet degenerate, see Sec. 3.1 in [197]
- `Nhj`: Hessian/Jacobian not yet degenerate, see Sec. 3.1 in [197]
- `Nj`: Jacobian not yet degenerate, see Sec. 3.1 in [197]
- `NW`: Warm start initialization failed (in Warm Start Initialization)
- `q`: PD system possibly singular, attempt to improve solution quality, see Sec. 3.1 in [197]
- `R`: Solution of restoration phase, see Sec. 3.3 in [197]
- `S`: PD system possibly singular, accept current solution, see Sec. 3.1 in [197]
- `s`: PD system singular, see Sec. 3.1 in [197]
- `s`: Square Problem. Set multipliers to zero (default initialization routine)
- `Tmax`: Trial θ is larger than $\theta_{\max}$ (filter parameter, Eq. (21) in [197])
- `W`: Watchdog line search procedure successful, see Sec. 3.2 in [197]
- `w`: Watchdog line search procedure unsuccessful, stopped, see Sec. 3.2 in [197]
- `Wb`: Undoing most recent SR1 update, see Sec. 5.4.1 in [23]
- `We`: Skip Limited-Memory Update in restoration phase, see Sec. 5.4.1 in [23]
- `Wp`: Safeguard $B^0 = \sigma I$ for Limited-Memory Update, see Sec. 5.4.1 in [23]
- `Wr`: Resetting Limited-Memory Update, see Sec. 5.4.1 in [23]
- `Ws`: Skip Limited-Memory Update since $s^T y$ is not positive, see Sec. 5.4.1 in [23]
- `WS`: Skip Limited-Memory Update since Δx is too small, see Sec. 5.4.1 in [23]
- `y`: Dual infeasibility, use least square multiplier update (during IPOPT algorithm)
- `z`: Apply correction to bound multiplier if too large (during IPOPT algorithm)

5.20.5 List of IPOPT Options

5.20.5.1 Termination

Option	Description	Default
acceptable_compl_inf_tol	"Acceptance" threshold for the complementarity conditions.	0.01
acceptable_constr_viol_tol	"Acceptance" threshold for the constraint violation.	0.01
acceptable_dual_inf_tol	"Acceptance" threshold for the dual infeasibility.	1e+10
acceptable_iter	Number of "acceptable" iterates before triggering termination.	0
acceptable_obj_change_tol	"Acceptance" stopping criterion based on objective function change.	1e+20
acceptable_tol	"Acceptable" convergence tolerance (relative).	1e-06
compl_inf_tol	Desired threshold for the complementarity conditions.	0.0001
constr_viol_tol	Desired threshold for the constraint and variable bound violation.	1e-06
diverging_iterates_tol	Threshold for maximal value of primal iterates.	1e+20
dual_inf_tol	Desired threshold for the dual infeasibility.	1
max_cpu_time	Maximum number of CPU seconds.	1e+20
max_iter	Maximum number of iterations.	GAMS iterlim
max_wall_time	Maximum number of walltime clock seconds.	GAMS reslim
mu_target	Desired value of complementarity.	0
tol	Desired convergence tolerance (relative).	1e-08
Options for expert users		
s_max	Scaling threshold for the NLP error.	100

5.20.5.2 Output

Option	Description	Default
inf_pr_output	Determines what value is printed in the "inf_pr" output column.	original
print_eval_error	Switch to enable printing information about function evaluation errors into the GAMS listing file.	yes
print_frequency_iter	Determines at which iteration frequency the summarizing iteration output line should be printed.	1
print_frequency_time	Determines at which time frequency the summarizing iteration output line should be printed.	0
print_info_string	Enables printing of additional info string at end of iteration output.	no
print_level	Output verbosity level.	5
print_options_mode	format in which to print options documentation	text
print_timing_statistics	Switch to print timing statistics.	no

Option	Description	Default
report_mininfeas_solution	Switch to report intermediate solution with minimal constraint violation to GAMS if the final solution is not feasible.	no
Options for expert users		
print_advanced_options	whether to print also advanced options	no

5.20.5.3 NLP

Option	Description	Default
bound_relax_factor	Factor for initial relaxation of the bounds.	1e-10
check_derivatives_for_naninf	Indicates whether it is desired to check for Nan/Inf in derivative matrices	no
fixed_variable_treatment	Determines how fixed variables should be handled.	make_parameter
honor_original_bounds	Indicates whether final points should be projected into original bounds.	no
Options for expert users		
dependency_detection_with_rhs	Indicates if the right hand sides of the constraints should be considered in addition to gradients during dependency detection	no
dependency_detector	Indicates which linear solver should be used to detect linearly dependent equality constraints.	none
kappa_d	Weight for linear damping term (to handle one-sided bounds).	1e-05

5.20.5.4 NLP Scaling

Option	Description	Default
nlp_scaling_max_gradient	Maximum gradient after NLP scaling.	100
nlp_scaling_method	Select the technique used for scaling the NLP.	gradient-based if GAMS scaleopt is not set, otherwise none
nlp_scaling_min_value	Minimum value of gradient-based scaling values.	1e-08
Options for expert users		
nlp_scaling_constr_target_gradient	Target value for constraint function gradient size.	0
nlp_scaling_obj_target_gradient	Target value for objective function gradient size.	0

5.20.5.5 Initialization

Option	Description	Default
bound_frac	Desired minimum relative distance from the initial point to bound.	0.01
bound_mult_init_method	Initialization method for bound multipliers	constant
bound_mult_init_val	Initial value for the bound multipliers.	1

Option	Description	Default
bound_push	Desired minimum absolute distance from the initial point to bound.	0.01
constr_mult_init_max	Maximum allowed least-square guess of constraint multipliers.	1000
least_square_init_duals	Least square initialization of all dual variables	no
least_square_init_primal	Least square initialization of the primal variables	no
slack_bound_frac	Desired minimum relative distance from the initial slack to bound.	0.01
slack_bound_push	Desired minimum absolute distance from the initial slack to bound.	0.01

5.20.5.6 Warm Start

Option	Description	Default
warm_start_bound_frac	same as bound_frac for the regular initializer	0.001
warm_start_bound_push	same as bound_push for the regular initializer	0.001
warm_start_init_point	Warm-start for initial point	yes, if run on modified model instance (e.g., from GUSS), otherwise no
warm_start_mult_bound_push	same as mult_bound_push for the regular initializer	0.001
warm_start_mult_init_max	Maximum initial value for the equality multipliers.	1e+06
warm_start_slack_bound_frac	same as slack_bound_frac for the regular initializer	0.001
warm_start_slack_bound_push	same as slack_bound_push for the regular initializer	0.001
Options for expert users		
warm_start_target_mu		0

5.20.5.7 Miscellaneous

Option	Description	Default
timing_statistics	Indicates whether to measure time spend in components of Ipopt and NLP evaluation	no
Options for expert users		
replace_bounds	Whether all variable bounds should be replaced by inequality constraints	no

5.20.5.8 Barrier Parameter Update

Option	Description	Default
adaptive_mu_globalization	Globalization strategy for the adaptive mu selection mode.	obj-constr-filter
barrier_tol_factor	Factor for mu in barrier stop test.	10

Option	Description	Default
fixed_mu_oracle	Oracle for the barrier parameter when switching to fixed mode.	average_compl
mu_init	Initial value for the barrier parameter.	0.1
mu_linear_decrease_factor	Determines linear decrease rate of barrier parameter.	0.2
mu_max	Maximum value for barrier parameter.	100000
mu_max_fact	Factor for initialization of maximum value for barrier parameter.	1000
mu_min	Minimum value for barrier parameter.	1e-11
mu_oracle	Oracle for a new barrier parameter in the adaptive strategy.	quality-function
mu_strategy	Update strategy for barrier parameter.	adaptive
mu_superlinear_decrease_power	Determines superlinear decrease rate of barrier parameter.	1.5
quality_function_max_section_steps	Maximum number of search steps during direct search procedure determining the optimal centering parameter.	8
Options for expert users		
adaptive_mu_kkt_norm_type	Norm used for the KKT error in the adaptive mu globalization strategies.	2-norm-squared
adaptive_mu_kkterror_red_fact	Sufficient decrease factor for "kkt-error" globalization strategy.	0.9999
adaptive_mu_kkterror_red_iters	Maximum number of iterations requiring sufficient progress.	4
adaptive_mu_monotone_init_factor	Determines the initial value of the barrier parameter when switching to the monotone mode.	0.8
adaptive_mu_restore_previous_iterate	Indicates if the previous accepted iterate should be restored if the monotone mode is entered.	no
filter_margin_fact	Factor determining width of margin for obj-constr-filter adaptive globalization strategy.	1e-05
filter_max_margin	Maximum width of margin in obj-constr-filter adaptive globalization strategy.	1
mu_allow_fast_monotone_decrease	Allow skipping of barrier problem if barrier test is already met.	yes
quality_function_balancing_term	The balancing term included in the quality function for centrality.	none
quality_function_centrality	The penalty term for centrality that is included in quality function.	none
quality_function_norm_type	Norm used for components of the quality function.	2-norm-squared
quality_function_section_qf_tol	Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space).	0
quality_function_section_sigma_tol	Tolerance for the section search procedure determining the optimal centering parameter (in sigma space).	0.01

Option	Description	Default
sigma_max	Maximum value of the centering parameter.	100
sigma_min	Minimum value of the centering parameter.	1e-06
tau_min	Lower bound on fraction-to-the-boundary parameter tau.	0.99

5.20.5.9 Line Search

Option	Description	Default
accept_every_trial_step	Always accept the first trial step.	no
alpha_for_y	Method to determine the step size for constraint multipliers (alpha_y) .	primal
alpha_for_y_tol	Tolerance for switching to full equality multiplier steps.	10
max_soc	Maximum number of second order correction trial steps at each iteration.	4
recalc_y	Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates.	no
recalc_y_feas_tol	Feasibility threshold for recomputation of multipliers.	1e-06
soc_method	Ways to apply second order correction	0
watchdog_shortened_iter_trigger	Number of shortened iterations that trigger the watchdog.	10
watchdog_trial_iter_max	Maximum number of watchdog iterations.	3
Options for expert users		
accept_after_max_steps	Accept a trial point after maximal this number of steps even if it does not satisfy line search conditions.	-1
alpha_min_frac	Safety factor for the minimal step size (before switching to restoration phase).	0.05
alpha_red_factor	Fractional reduction of the trial step size in the backtracking line search.	0.5
constraint_violation_norm_type	Norm to be used for the constraint violation in the line search.	1-norm
corrector_compl_avrg_red_fact	Complementarity tolerance factor for accepting corrector step.	1
corrector_type	The type of corrector steps that should be taken.	none
delta	Multiplier for constraint violation in the switching rule.	1
eta_phi	Relaxation factor in the Armijo condition.	1e-08
filter_reset_trigger	Number of iterations that trigger the filter reset.	5
gamma_phi	Relaxation factor in the filter margin for the barrier function.	1e-08
gamma_theta	Relaxation factor in the filter margin for the constraint violation.	1e-05
kappa_sigma	Factor limiting the deviation of dual variables from primal estimates.	1e+10
kappa_soc	Factor in the sufficient reduction rule for second order correction.	0.99

Option	Description	Default
line_search_method	Globalization method used in backtracking line search	filter
max_filter_resets	Maximal allowed number of filter resets	5
nu_inc	Increment of the penalty parameter.	0.0001
nu_init	Initial value of the penalty parameter.	1e-06
obj_max_inc	Determines the upper bound on the acceptable increase of barrier objective function.	5
rho	Value in penalty parameter update formula.	0.1
s_phi	Exponent for linear barrier function model in the switching rule.	2.3
s_theta	Exponent for current constraint violation in the switching rule.	1.1
skip_corr_if_neg_curv	Whether to skip the corrector step in negative curvature iteration.	yes
skip_corr_in_monotone_mode	Whether to skip the corrector step during monotone barrier parameter mode.	yes
slack_move	Correction size for very small slacks.	1.81899e-12
theta_max_fact	Determines upper bound for constraint violation in the filter.	10000
theta_min_fact	Determines constraint violation threshold in the switching rule.	0.0001
tiny_step_tol	Tolerance for detecting numerically insignificant steps.	2.22045e-15
tiny_step_y_tol	Tolerance for quitting because of numerically insignificant steps.	0.01

5.20.5.10 Linear Solver

Option	Description	Default
hsllib	Name of library containing HSL routines for load at runtime	libhsl.so (Linux), libhsl.dylib (macOS), libhsl.dll (Windows)
linear_scaling_on_demand	Flag indicating that linear scaling is only done if it seems required.	yes
linear_solver	Linear solver used for step computations.	ma27, if IpoptH, otherwise mumps
linear_system_scaling	Method for scaling the linear system.	mc19, if IpoptH, otherwise none
pardisolib	Name of library containing Pardiso routines (from pardiso-project.org) for load at runtime	libpardiso.so (Linux), libpardiso.dylib (macOS), libpardiso.dll (Windows)

5.20.5.11 Step Calculation

Option	Description	Default
fast_step_computation	Indicates if the linear system should be solved quickly.	no
first_hessian_perturbation	Size of first x-s perturbation tried.	0.0001

Option	Description	Default
jacobian_regularization_value	Size of the regularization for rank-deficient constraint Jacobians.	1e-08
max_hessian_perturbation	Maximum value of regularization parameter for handling negative curvature.	1e+20
max_refinement_steps	Maximum number of iterative refinement steps per linear system solve.	10
mehrotra_algorithm	Indicates whether to do Mehrotra's predictor-corrector algorithm.	no
min_hessian_perturbation	Smallest perturbation of the Hessian block.	1e-20
min_refinement_steps	Minimum number of iterative refinement steps per linear system solve.	1
neg_curv_test_reg	Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014).	yes
neg_curv_test_tol	Tolerance for heuristic to ignore wrong inertia.	0
perturb_dec_fact	Decrease factor for x-s perturbation.	0.333333
perturb_inc_fact	Increase factor for x-s perturbation.	8
perturb_inc_fact_first	Increase factor for x-s perturbation for very first perturbation.	100
Options for expert users		
jacobian_regularization_exponent	Exponent for mu in the regularization for rank-deficient constraint Jacobians.	0.25
perturb_always_cd	Active permanent perturbation of constraint linearization.	no
residual_improvement_factor	Minimal required reduction of residual test ratio in iterative refinement.	1
residual_ratio_max	Iterative refinement tolerance	1e-10
residual_ratio_singular	Threshold for declaring linear system singular after failed iterative refinement.	1e-05

5.20.5.12 Restoration Phase

Option	Description	Default
bound_mult_reset_threshold	Threshold for resetting bound multipliers after the restoration phase.	1000
constr_mult_reset_threshold	Threshold for resetting equality and inequality multipliers after restoration phase.	0
evaluate_orig_obj_at_resto_trial	Determines if the original objective function should be evaluated at restoration phase trial points.	yes
expect_infeasible_problem	Enable heuristics to quickly detect an infeasible problem.	no
expect_infeasible_problem_ctol	Threshold for disabling "expect_infeasible_problem" option.	0.001
expect_infeasible_problem_ytol	Multiplier threshold for activating "expect_infeasible_problem" option.	1e+08
required_infeasibility_reduction	Required reduction of infeasibility before leaving restoration phase.	0.9
soft_resto_pderror_reduction_factor	Required reduction in primal-dual error in the soft restoration phase.	0.9999

Option	Description	Default
start_with_resto	Whether to switch to restoration phase in first iteration.	no
Options for expert users		
max_resto_iter	Maximum number of successive iterations in restoration phase.	3000000
max_soft_resto_iters	Maximum number of iterations performed successively in soft restoration phase.	10
resto_failure_feasibility_threshold	Threshold for primal infeasibility to declare failure of restoration phase.	0
resto_penalty_parameter	Penalty parameter in the restoration phase objective function.	1000
resto_proximity_weight	Weighting factor for the proximity term in restoration phase objective.	1

5.20.5.13 Hessian Approximation

Option	Description	Default
hessian_approximation	Indicates what Hessian information is to be used.	exact
limited_memory_init_val	Value for B0 in low-rank update.	1
limited_memory_init_val_max	Upper bound on value for B0 in low-rank update.	1e+08
limited_memory_init_val_min	Lower bound on value for B0 in low-rank update.	1e-08
limited_memory_initialization	Initialization strategy for the limited memory quasi-Newton approximation.	scalar1
limited_memory_max_history	Maximum size of the history for the limited quasi-Newton Hessian approximation.	6
limited_memory_max_skipping	Threshold for successive iterations where update is skipped.	2
limited_memory_special_for_resto	Determines if the quasi-Newton updates should be special during the restoration phase.	no
limited_memory_update_type	Quasi-Newton update formula for the limited memory quasi-Newton approximation.	bfgs
Options for expert users		
hessian_approximation_space	Indicates in which subspace the Hessian information is to be approximated.	nonlinear-variables
limited_memory_aug_solver	Strategy for solving the augmented system for low-rank Hessian.	sherman-morrison

5.20.5.14 MA27 Linear Solver

Option	Description	Default
ma27_la_init_factor	Real workspace memory for MA27.	5
ma27_liw_init_factor	Integer workspace memory for MA27.	5
ma27_meminc_factor	Increment factor for workspace size for MA27.	2

Option	Description	Default
ma27_pivtol	Pivot tolerance for the linear solver MA27.	1e-08
ma27_pivtolmax	Maximum pivot tolerance for the linear solver MA27.	0.0001
ma27_print_level	Debug printing level for the linear solver MA27	0
Options for expert users		
ma27_ignore_singularity	Whether to use MA27's ability to solve a linear system even if the matrix is singular.	no
ma27_skip_inertia_check	Whether to always pretend that inertia is correct.	no

5.20.5.15 MA57 Linear Solver

Option	Description	Default
ma57_automatic_scaling	Controls whether to enable automatic scaling in MA57	no
ma57_block_size	Controls block size used by Level 3 BLAS in MA57BD	16
ma57_node_amalgamation	Node amalgamation parameter	16
ma57_pivot_order	Controls pivot order in MA57	5
ma57_pivtol	Pivot tolerance for the linear solver MA57.	1e-08
ma57_pivtolmax	Maximum pivot tolerance for the linear solver MA57.	0.0001
ma57_pre_alloc	Safety factor for work space memory allocation for the linear solver MA57.	1.05
ma57_print_level	Debug printing level for the linear solver MA57	0
ma57_small_pivot_flag	Handling of small pivots	0

5.20.5.16 MA77 Linear Solver

Option	Description	Default
ma77_buffer_lpage	Number of scalars per MA77 in-core buffer page in the out-of-core solver MA77	4096
ma77_buffer_npage	Number of pages that make up MA77 buffer	1600
ma77_file_size	Target size of each temporary file for MA77, scalars per type	2097152
ma77_maxstore	Maximum storage size for MA77 in-core mode	0
ma77_nemin	Node Amalgamation parameter	8
ma77_order	Controls type of ordering used by MA77	metis
ma77_print_level	Debug printing level for the linear solver MA77	-1
ma77_small	Zero Pivot Threshold	1e-20
ma77_static	Static Pivoting Threshold	0
ma77_u	Pivoting Threshold	1e-08
ma77_umax	Maximum Pivoting Threshold	0.0001

5.20.5.17 MA86 Linear Solver

Option	Description	Default
ma86_nemin	Node Amalgamation parameter	32
ma86_order	Controls type of ordering	auto
ma86_print_level	Debug printing level	-1
ma86_scaling	Controls scaling of matrix	mc64
ma86_small	Zero Pivot Threshold	1e-20
ma86_static	Static Pivoting Threshold	0
ma86_u	Pivoting Threshold	1e-08
ma86_umax	Maximum Pivoting Threshold	0.0001

5.20.5.18 MA97 Linear Solver

Option	Description	Default
ma97_nemin	Node Amalgamation parameter	8
ma97_order	Controls type of ordering	auto
ma97_print_level	Debug printing level	-1
ma97_scaling	Specifies strategy for scaling	dynamic
ma97_small	Zero Pivot Threshold	1e-20
ma97_u	Pivoting Threshold	1e-08
ma97_umax	Maximum Pivoting Threshold	0.0001
Options for expert users		
ma97_scaling1	First scaling.	mc64
ma97_scaling2	Second scaling.	mc64
ma97_scaling3	Third scaling.	mc64
ma97_solve_blas3	Controls if blas2 or blas3 routines are used for solve	no
ma97_switch1	First switch, determine when ma97_scaling1 is enabled.	od_hd_reuse
ma97_switch2	Second switch, determine when ma97_scaling2 is enabled.	never
ma97_switch3	Third switch, determine when ma97_scaling3 is enabled.	never

5.20.5.19 Pardiso (pardiso-project.org) Linear Solver

Option	Description	Default
pardiso_matching_strategy	Matching strategy to be used by Pardiso	complete+2x2
pardiso_max_iterative_refinement_steps	Limit on number of iterative refinement steps.	0

Option	Description	Default
pardiso_msglvl	Pardiso message level	0
pardiso_order	Controls the fill-in reduction ordering algorithm for the input matrix.	metis
Options for expert users		
pardiso_iter_coarse_size	Maximum Size of Coarse Grid Matrix	5000
pardiso_iter_dropping_factor	dropping value for incomplete factor	0.5
pardiso_iter_dropping_schur	dropping value for sparsify schur complement factor	0.1
pardiso_iter_inverse_norm_factor		5e+06
pardiso_iter_max_levels	Maximum Size of Grid Levels	10
pardiso_iter_max_row_fill	max fill for each row	10000000
pardiso_iter_relative_tol	Relative Residual Convergence	1e-06
pardiso_iterative	Switch for iterative solver in Pardiso library	no
pardiso_max_droptol_corrections	Maximal number of decreases of drop tolerance during one solve.	4
pardiso_max_iter	Maximum number of Krylov-Subspace Iteration	500
pardiso_redo_symbolic_fact_only_if_inertia_wrong	Toggle for handling case when elements were perturbed by Pardiso.	no
pardiso_repeated_perturbation_means_singular	Whether to assume that matrix is singular if elements were perturbed after recent symbolic factorization.	no
pardiso_skip_inertia_check	Whether to pretend that inertia is correct.	no

5.20.5.20 Pardiso (MKL) Linear Solver

Option	Description	Default
pardisomkl_matching_strategy	Matching strategy to be used by Pardiso	complete+2x2
pardisomkl_max_iterative_refinement_steps	Limit on number of iterative refinement steps.	1
pardisomkl_msglvl	Pardiso message level	0
pardisomkl_order	Controls the fill-in reduction ordering algorithm for the input matrix.	metis
Options for expert users		
pardisomkl_redo_symbolic_fact_only_if_inertia_wrong	Toggle for handling case when elements were perturbed by Pardiso.	no
pardisomkl_repeated_perturbation_means_singular	Whether to assume that matrix is singular if elements were perturbed after recent symbolic factorization.	no
pardisomkl_skip_inertia_check	Whether to pretend that inertia is correct.	no

5.20.5.21 Mumps Linear Solver

Option	Description	Default
mumps_mem_percent	Percentage increase in the estimated working space for MUMPS.	1000
mumps_permuting_scaling	Controls permuting and scaling in MUMPS	7
mumps_pivot_order	Controls pivot order in MUMPS	7
mumps_pivtol	Pivot tolerance for the linear solver MUMPS.	1e-06
mumps_pivtolmax	Maximum pivot tolerance for the linear solver MUMPS.	0.1
mumps_print_level	Debug printing level for the linear solver MUMPS	0
mumps_scaling	Controls scaling in MUMPS	77
Options for expert users		
mumps_dep_tol	Threshold to consider a pivot at zero in detection of linearly dependent constraints with MUMPS.	0

5.20.6 Detailed Options Description

accept_after_max_steps *(advanced)*: Accept a trial point after maximal this number of steps even if it does not satisfy line search conditions. ↩

Setting this to -1 disables this option.

Range: {-1, ..., ∞}

Default: -1

accept_every_trial_step: Always accept the first trial step. ↩

Setting this option to "yes" essentially disables the line search and makes the algorithm take aggressive steps, without global convergence guarantees.

Range: yes, no

Default: no

acceptable_compl_inf_tol: "Acceptance" threshold for the complementarity conditions. ↩

Absolute tolerance on the complementarity. "Acceptable" termination requires that the max-norm of the (unscaled) complementarity is less than this threshold; see also acceptable_tol.

Range: (0, ∞]

Default: 0.01

acceptable_constr_viol_tol: "Acceptance" threshold for the constraint violation. ↩

Absolute tolerance on the constraint violation. "Acceptable" termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold; see also acceptable_tol.

Range: (0, ∞]

Default: 0.01

acceptable_dual_inf_tol: "Acceptance" threshold for the dual infeasibility. ↩

Absolute tolerance on the dual infeasibility. "Acceptable" termination requires that the (max-norm of the unscaled) dual infeasibility is less than this threshold; see also acceptable_tol.

Range: (0, ∞]

Default: 1e+10

acceptable_iter: Number of "acceptable" iterates before triggering termination. ↩

If the algorithm encounters this many successive "acceptable" iterates (see "acceptable_tol"), it terminates, assuming that the problem has been solved to best possible accuracy given round-off. If it is set to zero, this heuristic is disabled.

Range: {0, ..., ∞}

Default: 0

acceptable_obj_change_tol: "Acceptance" stopping criterion based on objective function change. ↩

If the relative change of the objective function (scaled by Max(1,|f(x)|)) is less than this value, this part of the acceptable tolerance termination is satisfied; see also acceptable_tol. This is useful for the quasi-Newton option, which has trouble to bring down the dual infeasibility.

Range: [0, ∞]

Default: 1e+20

acceptable_tol: "Acceptable" convergence tolerance (relative). ↩

Determines which (scaled) overall optimality error is considered to be "acceptable". There are two levels of termination criteria. If the usual "desired" tolerances (see tol, dual_inf_tol etc) are satisfied at an iteration, the algorithm immediately terminates with a success message. On the other hand, if the algorithm encounters "acceptable_iter" many iterations in a row that are considered "acceptable", it will terminate before the desired convergence tolerance is met. This is useful in cases where the algorithm might not be able to achieve the "desired" level of accuracy.

Range: (0, ∞]

Default: 1e-06

adaptive_mu_globalization: Globalization strategy for the adaptive mu selection mode. ↩

To achieve global convergence of the adaptive version, the algorithm has to switch to the monotone mode (Fiacco-McCormick approach) when convergence does not seem to appear. This option sets the criterion used to decide when to do this switch. (Only used if option "mu_strategy" is chosen as "adaptive".)

value	meaning
kkt-error	nonmonotone decrease of kkt-error
obj-constr-filter	2-dim filter for objective and constraint violation
never-monotone-mode	disables globalization

Default: obj-constr-filter

adaptive_mu_kkt_norm_type *(advanced)*: Norm used for the KKT error in the adaptive mu globalization strategies. ↩

When computing the KKT error for the globalization strategies, the norm to be used is specified with this option. Note, this option is also used in the QualityFunctionMuOracle.

value	meaning
1-norm	use the 1-norm (abs sum)
2-norm-squared	use the 2-norm squared (sum of squares)
max-norm	use the infinity norm (max)
2-norm	use 2-norm

Default: 2-norm-squared

adaptive_mu_kkterror_red_fact *(advanced)*: Sufficient decrease factor for "kkt-error" globalization strategy. ↩

For the "kkt-error" based globalization strategy, the error must decrease by this factor to be deemed sufficient decrease.

Range: (0, 1)

Default: 0.9999

adaptive_mu_kkterror_red_iters *(advanced)*: Maximum number of iterations requiring sufficient progress. ↩

For the "kkt-error" based globalization strategy, sufficient progress must be made for "adaptive_mu_kkterror_red_iters" iterations. If this number of iterations is exceeded, the globalization strategy switches to the monotone mode.

Range: {0, ..., ∞}

Default: 4

adaptive_mu_monotone_init_factor *(advanced)*: Determines the initial value of the barrier parameter when switching to the monotone mode. ↩

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode and fixed_mu_oracle is chosen as "average_compl", the barrier parameter is set to the current average complementarity times the value of "adaptive_mu_monotone_init_factor".

Range: (0, ∞]

Default: 0.8

adaptive_mu_restore_previous_iterate *(advanced)*: Indicates if the previous accepted iterate should be restored if the monotone mode is entered. ↩

When the globalization strategy for the adaptive barrier algorithm switches to the monotone mode, it can either start from the most recent iterate (no), or from the last iterate that was accepted (yes).

Range: yes, no

Default: no

alpha_for_y: Method to determine the step size for constraint multipliers (alpha_y) . ↩

value	meaning
primal	use primal step size
bound-mult	use step size for the bound multipliers (good for LPs)
min	use the min of primal and bound multipliers
max	use the max of primal and bound multipliers
full	take a full step of size one
min-dual-infeas	choose step size minimizing new dual infeasibility
safer-min-dual-infeas	like "min_dual_infeas", but safeguarded by "min" and "max"
primal-and-full	use the primal step size, and full step if delta_x $\leq$ alpha_for_y_tol
dual-and-full	use the dual step size, and full step if delta_x $\leq$ alpha_for_y_tol
acceptor	Call LSAcceptor to get step size for y

Default: primal

alpha_for_y_tol: Tolerance for switching to full equality multiplier steps. ↩

This is only relevant if "alpha_for_y" is chosen "primal-and-full" or "dual-and-full". The step size for the equality constraint multipliers is taken to be one if the max-norm of the primal step is less than this tolerance.

Range: [0, ∞]

Default: 10

alpha_min_frac *(advanced)*: Safety factor for the minimal step size (before switching to restoration phase). ↩

This is gamma_alpha in Eqn. (23) in the implementation paper.

Range: (0, 1)

Default: 0.05

alpha_red_factor *(advanced)*: Fractional reduction of the trial step size in the backtracking line search. ↩

At every step of the backtracking line search, the trial step size is reduced by this factor.

Range: (0, 1)

Default: 0.5

barrier_tol_factor: Factor for mu in barrier stop test. ↩

The convergence tolerance for each barrier problem in the monotone mode is the value of the barrier parameter times "barrier_tol_factor". This option is also used in the adaptive mu strategy during the monotone mode. This is kappa_epsilon in implementation paper.

Range: (0, ∞]

Default: 10

bound_frac: Desired minimum relative distance from the initial point to bound. ↩

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: (0, 0.5]

Default: 0.01

bound_mult_init_method: Initialization method for bound multipliers ↩

This option defines how the iterates for the bound multipliers are initialized. If "constant" is chosen, then all bound multipliers are initialized to the value of "bound_mult_init_val". If "mu-based" is chosen, then each value is initialized to the the value of "mu_init" divided by the corresponding slack variable. This latter option might be useful if the starting point is close to the optimal solution.

value	meaning
constant	set all bound multipliers to the value of bound_mult_init_val
mu-based	initialize to mu_init/x_slack

Default: constant

bound_mult_init_val: Initial value for the bound multipliers. ↩

All dual variables corresponding to bound constraints are initialized to this value.

Range: (0, ∞]

Default: 1

bound_mult_reset_threshold: Threshold for resetting bound multipliers after the restoration phase. ↩

After returning from the restoration phase, the bound multipliers are updated with a Newton step for complementarity. Here, the change in the primal variables during the entire restoration phase is taken to be the corresponding primal Newton step. However, if after the update the largest bound multiplier exceeds the threshold specified by this option, the multipliers are all reset to 1.

Range: [0, ∞]

Default: 1000

bound_push: Desired minimum absolute distance from the initial point to bound. ↩

Determines how much the initial point might have to be modified in order to be sufficiently inside the bounds (together with "bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

Range: (0, ∞]

Default: 0.01

bound_relax_factor: Factor for initial relaxation of the bounds. ↩

Before start of the optimization, the bounds given by the user are relaxed. This option sets the factor for this relaxation. Additional, the constraint violation tolerance constr_viol_tol is used to bound the relaxation by an absolute value. If it is set to zero, then then bounds relaxation is disabled. See Eqn.(35) in implementation paper. Note that the constraint violation reported by Ipopt at the end of the solution process does not include violations of the original (non-relaxed) variable bounds. See also option honor_original_bounds.

Range: [0, ∞]

Default: 1e-10

check_derivatives_for_naninf: Indicates whether it is desired to check for Nan/Inf in derivative matrices ↩

Activating this option will cause an error if an invalid number is detected in the constraint Jacobians or the Lagrangian Hessian. If this is not activated, the test is skipped, and the algorithm might proceed with invalid numbers and fail. If test is activated and an invalid number is detected, the matrix is written to output with print_level corresponding to J_MOREDETAILED (7); so beware of large output!

Range: yes, no

Default: no

compl_inf_tol: Desired threshold for the complementarity conditions. ↩

Absolute tolerance on the complementarity. Successful termination requires that the max-norm of the (unscaled) complementarity is less than this threshold.

Range: (0, ∞]

Default: 0.0001

constr_mult_init_max: Maximum allowed least-square guess of constraint multipliers. ↩

Determines how large the initial least-square guesses of the constraint multipliers are allowed to be (in max-norm). If the guess is larger than this value, it is discarded and all constraint multipliers are set to zero. This options is also used when initializing the restoration phase. By default, "resto.constr_mult_init_max" (the one used in RestoIterateInitializer) is set to zero.

Range: [0, ∞]

Default: 1000

constr_mult_reset_threshold: Threshold for resetting equality and inequality multipliers after restoration phase. ↩

After returning from the restoration phase, the constraint multipliers are recomputed by a least square estimate. This option triggers when those least-square estimates should be ignored.

Range: [0, ∞]

Default: 0

constr_viol_tol: Desired threshold for the constraint and variable bound violation. ↩

Absolute tolerance on the constraint and variable bound violation. Successful termination requires that the max-norm of the (unscaled) constraint violation is less than this threshold. If option bound_relax_factor is not zero 0, then Ipopt relaxes given variable bounds. The value of constr_viol_tol is used to restrict the absolute amount of this bound relaxation.

Range: (0, ∞]

Default: 1e-06

constraint_violation_norm_type *(advanced)*: Norm to be used for the constraint violation in the line search. ↩

Determines which norm should be used when the algorithm computes the constraint violation in the line search.

value	meaning
1-norm	use the 1-norm
2-norm	use the 2-norm
max-norm	use the infinity norm

Default: 1-norm

corrector_compl_avrg_red_fact *(advanced)*: Complementarity tolerance factor for accepting corrector step. ↩

This option determines the factor by which complementarity is allowed to increase for a corrector step to be accepted. Changing this option is experimental.

Range: (0, ∞]

Default: 1

corrector_type *(advanced)*: The type of corrector steps that should be taken. ↩

If "mu_strategy" is "adaptive", this option determines what kind of corrector steps should be tried. Changing this option is experimental.

value	meaning
none	no corrector
affine	corrector step towards mu=0
primal-dual	corrector step towards current mu

Default: none

delta *(advanced)*: Multiplier for constraint violation in the switching rule. ↩

See Eqn. (19) in the implementation paper.

Range: (0, ∞]

Default: 1

dependency_detection_with_rhs *(advanced)*: Indicates if the right hand sides of the constraints should be considered in addition to gradients during dependency detection ↩

Range: yes, no

Default: no

dependency_detector *(advanced)*: Indicates which linear solver should be used to detect linearly dependent equality constraints. ↩

This is experimental and does not work well.

value	meaning
none	don't check; no extra work at beginning
mumps	use MUMPS

Default: none

diverging_iterates_tol: Threshold for maximal value of primal iterates. ↩

If any component of the primal iterates exceeded this value (in absolute terms), the optimization is aborted with the exit message that the iterates seem to be diverging.

Range: (0, ∞]

Default: 1e+20

dual_inf_tol: Desired threshold for the dual infeasibility. ↩

Absolute tolerance on the dual infeasibility. Successful termination requires that the max-norm of the (unscaled) dual infeasibility is less than this threshold.

Range: (0, ∞]

Default: 1

eta_phi *(advanced)*: Relaxation factor in the Armijo condition. ↩

See Eqn. (20) in the implementation paper.

Range: (0, 0.5)

Default: 1e-08

evaluate_orig_obj_at_resto_trial: Determines if the original objective function should be evaluated at restoration phase trial points. ↩

Enabling this option makes the restoration phase algorithm evaluate the objective function of the original problem at every trial point encountered during the restoration phase, even if this value is not required. In this way, it is guaranteed that the original objective function can be evaluated without error at all accepted iterates; otherwise the algorithm might fail at a point where the restoration phase accepts an iterate that is good for the restoration phase problem, but not the original problem. On the other hand, if the evaluation of the original objective is expensive, this might be costly.

Range: yes, no

Default: yes

expect_infeasible_problem: Enable heuristics to quickly detect an infeasible problem. ↩

This options is meant to activate heuristics that may speed up the infeasibility determination if you expect that there is a good chance for the problem to be infeasible. In the filter line search procedure, the restoration phase is called more quickly than usually, and more reduction in the constraint violation is enforced before the restoration phase is left. If the problem is square, this option is enabled automatically.

Range: yes, no

Default: no

expect_infeasible_problem_ctol: Threshold for disabling "expect_infeasible_problem" option. ↩

If the constraint violation becomes smaller than this threshold, the "expect_infeasible_problem" heuristics in the filter line search are disabled. If the problem is square, this options is set to 0.

Range: [0, ∞]

Default: 0.001

expect_infeasible_problem_ytol: Multiplier threshold for activating "expect_infeasible_problem" option. ↩

If the max norm of the constraint multipliers becomes larger than this value and "expect_infeasible_problem" is chosen, then the restoration phase is entered.

Range: (0, ∞]

Default: 1e+08

fast_step_computation: Indicates if the linear system should be solved quickly. ↩

If enabled, the algorithm assumes that the linear system that is solved to obtain the search direction is solved sufficiently well. In that case, no residuals are computed to verify the solution and the computation of the search direction is a little faster.

Range: yes, no

Default: no

filter_margin_fact *(advanced)*: Factor determining width of margin for obj-constr-filter adaptive globalization strategy. ↩

When using the adaptive globalization strategy, "obj-constr-filter", sufficient progress for a filter entry is defined as follows: (new obj) < (filter obj) - filter_margin_fact∗(new constr-viol) OR (new constr-viol) < (filter constr-viol) - filter_margin_fact∗(new constr-viol). For the description of the "kkt-error-filter" option see "filter_max_margin".

Range: (0, 1)

Default: 1e-05

filter_max_margin *(advanced)*: Maximum width of margin in obj-constr-filter adaptive globalization strategy. ↩

Range: (0, ∞]

Default: 1

filter_reset_trigger *(advanced)*: Number of iterations that trigger the filter reset. ↩

If the filter reset heuristic is active and the number of successive iterations in which the last rejected trial step size was rejected because of the filter, the filter is reset.

Range: {1, ..., ∞}

Default: 5

first_hessian_perturbation: Size of first x-s perturbation tried. ↩

The first value tried for the x-s perturbation in the inertia correction scheme. This is delta_0 in the implementation paper.

Range: (0, ∞]

Default: 0.0001

fixed_mu_oracle: Oracle for the barrier parameter when switching to fixed mode. ↩

Determines how the first value of the barrier parameter should be computed when switching to the "monotone mode" in the adaptive strategy. (Only considered if "adaptive" is selected for option "mu_strategy".)

value	meaning
probing	Mehrotra's probing heuristic
loqo	LOQO's centrality rule
quality-function	minimize a quality function
average_compl	base on current average complementarity

Default: average_compl

fixed_variable_treatment: Determines how fixed variables should be handled. ↩

The main difference between those options is that the starting point in the "make_constraint" case still has the fixed variables at their given values, whereas in the case "make_parameter(_nodual)" the functions are always evaluated with the fixed values for those variables. Also, for "relax_bounds", the fixing bound constraints are relaxed (according to" bound_relax_factor"). For all but "make_parameter_nodual", bound multipliers are computed for the fixed variables.

value	meaning
make_parameter	Remove fixed variable from optimization variables
make_parameter_nodual	Remove fixed variable from optimization variables and do not compute bound multipliers for fixed variables
make_constraint	Add equality constraints fixing variables
relax_bounds	Relax fixing bound constraints

Default: make_parameter

gamma_phi *(advanced)*: Relaxation factor in the filter margin for the barrier function. ↩

See Eqn. (18a) in the implementation paper.

Range: (0, 1)

Default: 1e-08

gamma_theta *(advanced)*: Relaxation factor in the filter margin for the constraint violation. ↩

See Eqn. (18b) in the implementation paper.

Range: (0, 1)

Default: 1e-05

hessian_approximation: Indicates what Hessian information is to be used. ↩

This determines which kind of information for the Hessian of the Lagrangian function is used by the algorithm.

value	meaning
exact	Use second derivatives provided by the NLP.
limited-memory	Perform a limited-memory quasi-Newton approximation

Default: exact

hessian_approximation_space *(advanced)*: Indicates in which subspace the Hessian information is to be approximated. ↩

value	meaning
nonlinear-variables	only in space of nonlinear variables.
all-variables	in space of all variables (without slacks)

Default: nonlinear-variables

honor_original_bounds: Indicates whether final points should be projected into original bounds. ↩

Ipopt might relax the bounds during the optimization (see, e.g., option "bound_relax_factor"). This option determines whether the final point should be projected back into the user-provide original bounds after the optimization. Note that violations of constraints and complementarity reported by Ipopt at the end of the solution process are for the non-projected point.

Range: yes, no

Default: no

hsllib: Name of library containing HSL routines for load at runtime ↩

Range: string

Default: libhsl.so (Linux), libhsl.dylib (macOS), libhsl.dll (Windows)

inf_pr_output: Determines what value is printed in the "inf_pr" output column. ↩

Ipopt works with a reformulation of the original problem, where slacks are introduced and the problem might have been scaled. The choice "internal" prints out the constraint violation of this formulation. With "original" the true constraint violation in the original NLP is printed.

value	meaning
internal	max-norm of violation of internal equality constraints
original	maximal constraint violation in original NLP

Default: original

jacobian_regularization_exponent *(advanced)*: Exponent for mu in the regularization for rank-deficient constraint Jacobians. ↩

This is kappa_c in the implementation paper.

Range: [0, ∞]

Default: 0.25

jacobian_regularization_value: Size of the regularization for rank-deficient constraint Jacobians. ↩

This is bar delta_c in the implementation paper.

Range: [0, ∞]

Default: 1e-08

kappa_d *(advanced)*: Weight for linear damping term (to handle one-sided bounds). ↩

See Section 3.7 in implementation paper.

Range: [0, ∞]

Default: 1e-05

kappa_sigma *(advanced)*: Factor limiting the deviation of dual variables from primal estimates. ↩

If the dual variables deviate from their primal estimates, a correction is performed. See Eqn. (16) in the implementation paper. Setting the value to less than 1 disables the correction.

Range: (0, ∞]

Default: 1e+10

kappa_soc *(advanced)*: Factor in the sufficient reduction rule for second order correction. ↩

This option determines how much a second order correction step must reduce the constraint violation so that further correction steps are attempted. See Step A-5.9 of Algorithm A in the implementation paper.

Range: (0, ∞]

Default: 0.99

least_square_init_duals: Least square initialization of all dual variables ↩

If set to yes, Ipopt tries to compute least-square multipliers (considering ALL dual variables). If successful, the bound multipliers are possibly corrected to be at least bound_mult_init_val. This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP. This overwrites option "bound_mult_init_method".

value	meaning
no	use bound_mult_init_val and least-square equality constraint multipliers
yes	overwrite user-provided point with least-square estimates

Default: no

least_square_init_primal: Least square initialization of the primal variables ↩

If set to yes, Ipopt ignores the user provided point and solves a least square problem for the primal variables (x and s) to fit the linearized equality and inequality constraints.This might be useful if the user doesn't know anything about the starting point, or for solving an LP or QP.

value	meaning
no	take user-provided point
yes	overwrite user-provided point with least-square estimates

Default: no

limited_memory_aug_solver *(advanced)*: Strategy for solving the augmented system for low-rank Hessian. ↩

value	meaning
sherman-morrison	use Sherman-Morrison formula
extended	use an extended augmented system

Default: sherman-morrison

limited_memory_init_val: Value for B0 in low-rank update. ↩

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Range: (0, ∞]

Default: 1

limited_memory_init_val_max: Upper bound on value for B0 in low-rank update. ↩

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Range: (0, ∞]

Default: 1e+08

limited_memory_init_val_min: Lower bound on value for B0 in low-rank update. ↩

The starting matrix in the low rank update, B0, is chosen to be this multiple of the identity in the first iteration (when no updates have been performed yet), and is constantly chosen as this value, if "limited_memory_initialization" is "constant".

Range: (0, ∞]

Default: 1e-08

limited_memory_initialization: Initialization strategy for the limited memory quasi-Newton approximation. ↩

Determines how the diagonal Matrix B_0 as the first term in the limited memory approximation should be computed.

value	meaning
scalar1	sigma = s^Ty/s^Ts
scalar2	sigma = y^Ty/s^Ty
scalar3	arithmetic average of scalar1 and scalar2
scalar4	geometric average of scalar1 and scalar2
constant	sigma = limited_memory_init_val

Default: scalar1

limited_memory_max_history: Maximum size of the history for the limited quasi-Newton Hessian approximation. ↩

This option determines the number of most recent iterations that are taken into account for the limited-memory quasi-Newton approximation.

Range: {0, ..., ∞}

Default: 6

limited_memory_max_skipping: Threshold for successive iterations where update is skipped. ↩

If the update is skipped more than this number of successive iterations, the quasi-Newton approximation is reset.

Range: {1, ..., ∞}

Default: 2

limited_memory_special_for_resto: Determines if the quasi-Newton updates should be special during the restoration phase. ↩

Until Nov 2010, Ipopt used a special update during the restoration phase, but it turned out that this does not work well. The new default uses the regular update procedure and it improves results. If for some reason you want to get back to the original update, set this option to "yes".

Range: yes, no

Default: no

limited_memory_update_type: Quasi-Newton update formula for the limited memory quasi-Newton approximation. ↩

value	meaning
bfgs	BFGS update (with skipping)
sr1	SR1 (not working well)

Default: bfgs

line_search_method *(advanced)*: Globalization method used in backtracking line search ↩

Only the "filter" choice is officially supported. But sometimes, good results might be obtained with the other choices.

value	meaning
filter	Filter method
cg-penalty	Chen-Goldfarb penalty function
penalty	Standard penalty function

Default: filter

linear_scaling_on_demand: Flag indicating that linear scaling is only done if it seems required. ↩

This option is only important if a linear scaling method (e.g., mc19) is used. If you choose "no", then the scaling factors are computed for every linear system from the start. This can

be quite expensive. Choosing "yes" means that the algorithm will start the scaling method only when the solutions to the linear system seem not good, and then use it until the end.

Range: yes, no

Default: yes

linear_solver: Linear solver used for step computations. ↩

Determines which linear algebra package is to be used for the solution of the augmented linear system (for obtaining the search directions). Note, that MA27, MA57, MA86, and MA97 are included with a commercially supported GAMS/IpoptH license only. To use MA27, MA57, MA86, or MA97 with GAMS/Ipopt, or to use HSL_MA77, a HSL library needs to be provided by the user. To use Pardiso from pardiso-project.org, a Pardiso library needs to be provided by the user. **ATTENTION**: Before Ipopt 3.14 (GAMS 36), value ***pardiso*** specified to use Pardiso from Intel MKL. With GAMS 36, this value has been renamed to ***pardisomkl***. On GAMS systems for ARM64 CPUs, option value `pardisomkl` is not available.

value	meaning
ma27	IpoptH: use the Harwell routine MA27; Ipopt: load the Harwell routine MA27 from user-provided library
ma57	IpoptH: use the Harwell routine MA57; Ipopt: load the Harwell routine MA57 from user-provided library
ma77	load the Harwell routine HSL_MA77 from user-provided library
ma86	IpoptH: use the Harwell routine HSL_MA86; Ipopt: load the Harwell routine HSL_MA86 from user-provided library
ma97	IpoptH: use the Harwell routine HSL_MA97; Ipopt: load the Harwell routine HSL_MA97 from user-provided library
pardiso	load the Pardiso package from pardiso-project.org from user-provided library at runtime
pardisomkl	use the Pardiso package from Intel MKL
mumps	use the Mumps package

Default: ma27, if IpoptH, otherwise mumps

linear_system_scaling: Method for scaling the linear system. ↩

Determines the method used to compute symmetric scaling factors for the augmented system (see also the "linear_scaling_on_demand" option). This scaling is independent of the NLP problem scaling. Note, that MC19 is included with a commercially supported GAMS/IpoptH license only. To use MC19 with GAMS/Ipopt, a HSL library needs to be provided by the user.

value	meaning
none	no scaling will be performed
mc19	IpoptH: use the Harwell routine MC19; Ipopt: load the Harwell routine MC19 from user-provided library
slack-based	use the slack values

Default: mc19, if IpoptH, otherwise none

ma27_ignore_singularity *(advanced)*: Whether to use MA27's ability to solve a linear system even if the matrix is singular. ↩

Setting this option to "yes" means that Ipopt will call MA27 to compute solutions for right hand sides, even if MA27 has detected that the matrix is singular (but is still able to solve the linear system). In some cases this might be better than using Ipopt's heuristic of small perturbation of the lower diagonal of the KKT matrix.

Range: yes, no

Default: no

ma27_la_init_factor: Real workspace memory for MA27. ↩

The initial real workspace memory = la_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by ma27_meminc_factor if required.

Range: [1, ∞]

Default: 5

ma27_liw_init_factor: Integer workspace memory for MA27. ↩

The initial integer workspace memory = liw_init_factor * memory required by unfactored system. Ipopt will increase the workspace size by ma27_meminc_factor if required.

Range: [1, ∞]

Default: 5

ma27_meminc_factor: Increment factor for workspace size for MA27. ↩

If the integer or real workspace is not large enough, Ipopt will increase its size by this factor.

Range: [1, ∞]

Default: 2

ma27_pivtol: Pivot tolerance for the linear solver MA27. ↩

A smaller number pivots for sparsity, a larger number pivots for stability.

Range: (0, 1)

Default: 1e-08

ma27_pivtolmax: Maximum pivot tolerance for the linear solver MA27. ↩

Ipopt may increase pivtol as high as ma27_pivtolmax to get a more accurate solution to the linear system.

Range: (0, 1)

Default: 0.0001

ma27_print_level: Debug printing level for the linear solver MA27 ↩

0: no printing; 1: Error messages only; 2: Error and warning messages; 3: Error and warning messages and terse monitoring; 4: All information.

Range: {0, ..., 4}

Default: 0

ma27_skip_inertia_check *(advanced)*: Whether to always pretend that inertia is correct. ↩

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Range: yes, no

Default: no

ma57_automatic_scaling: Controls whether to enable automatic scaling in MA57 ↩

For higher reliability of the MA57 solver, you may want to set this option to yes. This is ICNTL(15) in MA57.

Range: yes, no

Default: no

ma57_block_size: Controls block size used by Level 3 BLAS in MA57BD ↩

This is ICNTL(11) in MA57.

Range: {1, ..., ∞}

Default: 16

ma57_node_amalgamation: Node amalgamation parameter ↩

This is ICNTL(12) in MA57.

Range: {1, ..., ∞}

Default: 16

ma57_pivot_order: Controls pivot order in MA57 ↩

This is ICNTL(6) in MA57.

Range: {0, ..., 5}

Default: 5

ma57_pivtol: Pivot tolerance for the linear solver MA57. ↩

A smaller number pivots for sparsity, a larger number pivots for stability.

Range: (0, 1)

Default: 1e-08

ma57_pivtolmax: Maximum pivot tolerance for the linear solver MA57. ↩

Ipopt may increase pivtol as high as ma57_pivtolmax to get a more accurate solution to the linear system.

Range: (0, 1)

Default: 0.0001

ma57_pre_alloc: Safety factor for work space memory allocation for the linear solver MA57. ↩

If 1 is chosen, the suggested amount of work space is used. However, choosing a larger number might avoid reallocation if the suggest values do not suffice.

Range: [1, ∞]

Default: 1.05

ma57_print_level: Debug printing level for the linear solver MA57 ↩

0: no printing; 1: Error messages only; 2: Error and warning messages; 3: Error and warning messages and terse monitoring; ≥4: All information.

Range: {0, ..., ∞}

Default: 0

ma57_small_pivot_flag: Handling of small pivots ↩

If set to 1, then when small entries defined by CNTL(2) are detected they are removed and the corresponding pivots placed at the end of the factorization. This can be particularly efficient if the matrix is highly rank deficient. This is ICNTL(16) in MA57.

Range: {0, ..., 1}

Default: 0

ma77_buffer_lpage: Number of scalars per MA77 in-core buffer page in the out-of-core solver MA77 ↩

Must be at most ma77_file_size.

Range: {1, ..., ∞}

Default: 4096

ma77_buffer_npage: Number of pages that make up MA77 buffer ↩

Number of pages of size buffer_lpage that exist in-core for the out-of-core solver MA77.

Range: {1, ..., ∞}

Default: 1600

ma77_file_size: Target size of each temporary file for MA77, scalars per type ↩

MA77 uses many temporary files, this option controls the size of each one. It is measured in the number of entries (int or double), NOT bytes.

Range: {1, ..., ∞}

Default: 2097152

ma77_maxstore: Maximum storage size for MA77 in-core mode ↩

If greater than zero, the maximum size of factors stored in core before out-of-core mode is invoked.

Range: {0, ..., ∞}

Default: 0

ma77_nemin: Node Amalgamation parameter ↩

Two nodes in elimination tree are merged if result has fewer than ma77_nemin variables.

Range: {1, ..., ∞}

Default: 8

ma77_order: Controls type of ordering used by MA77 ↩

value	meaning
amd	Use the HSL_MC68 approximate minimum degree algorithm
metis	Use the MeTiS nested dissection algorithm (if available)

Default: metis

ma77_print_level: Debug printing level for the linear solver MA77 ↩

<0: no printing; 0: Error and warning messages only; 1: Limited diagnostic printing; >1 Additional diagnostic printing.

Range: {-∞, ..., ∞}

Default: -1

ma77_small: Zero Pivot Threshold ↩

Any pivot less than ma77_small is treated as zero.

Range: [0, ∞]

Default: 1e-20

ma77_static: Static Pivoting Threshold ↩

See MA77 documentation. Either ma77_static=0.0 or ma77_static>ma77_small. ma77_static=0.0 disables static pivoting.

Range: [0, ∞]

Default: 0

ma77_u: Pivoting Threshold ↩

See MA77 documentation.

Range: [0, 0.5]

Default: 1e-08

ma77_umax: Maximum Pivoting Threshold ↩

Maximum value to which u will be increased to improve quality.

Range: [0, 0.5]

Default: 0.0001

ma86_nemin: Node Amalgamation parameter ↩

Two nodes in elimination tree are merged if result has fewer than ma86_nemin variables.

Range: {1, ..., ∞}

Default: 32

ma86_order: Controls type of ordering ↩

value	meaning
auto	Try both AMD and MeTiS, pick best
amd	Use the HSL_MC68 approximate minimum degree algorithm
metis	Use the MeTiS nested dissection algorithm (if available)

Default: auto

ma86_print_level: Debug printing level ↩

<0: no printing; 0: Error and warning messages only; 1: Limited diagnostic printing; >1 Additional diagnostic printing.

Range: {-∞, ..., ∞}

Default: -1

ma86_scaling: Controls scaling of matrix ↩

value	meaning
none	Do not scale the linear system matrix
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]

Default: mc64

ma86_small: Zero Pivot Threshold ↩

Any pivot less than ma86_small is treated as zero.

Range: [0, ∞]

Default: 1e-20

ma86_static: Static Pivoting Threshold ↩

See MA86 documentation. Either ma86_static=0.0 or ma86_static>ma86_small. ma86_static=0.0 disables static pivoting.

Range: [0, ∞]

Default: 0

ma86_u: Pivoting Threshold ↩

See MA86 documentation.

Range: [0, 0.5]

Default: 1e-08

ma86_umax: Maximum Pivoting Threshold ↩

Maximum value to which u will be increased to improve quality.

Range: [0, 0.5]

Default: 0.0001

ma97_nemin: Node Amalgamation parameter ↩

Two nodes in elimination tree are merged if result has fewer than ma97_nemin variables.

Range: {1, ..., ∞}

Default: 8

ma97_order: Controls type of ordering ↩

value	meaning
auto	Use HSL_MA97 heuristic to guess best of AMD and METIS
best	Try both AMD and MeTiS, pick best
amd	Use the HSL_MC68 approximate minimum degree algorithm
metis	Use the MeTiS nested dissection algorithm
matched-auto	Use the HSL_MC80 matching with heuristic choice of AMD or METIS
matched-metis	Use the HSL_MC80 matching based ordering with METIS
matched-amd	Use the HSL_MC80 matching based ordering with AMD

Default: auto

ma97_print_level: Debug printing level ↩

<0: no printing; 0: Error and warning messages only; 1: Limited diagnostic printing; >1 Additional diagnostic printing.

Range: {-∞, ..., ∞}

Default: -1

ma97_scaling: Specifies strategy for scaling ↩

value	meaning
none	Do not scale the linear system matrix
mc30	Scale all linear system matrices using MC30
mc64	Scale all linear system matrices using MC64
mc77	Scale all linear system matrices using MC77 [1,3,0]
dynamic	Dynamically select scaling according to rules specified by ma97_scalingX and ma97_switchX options.

Default: dynamic

ma97_scaling1 *(advanced)*: First scaling. ↩

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch1. If ma97_switch2 is triggered it is disabled.

value	meaning
none	No scaling
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]

Default: mc64

ma97_scaling2 *(advanced)*: Second scaling. ↩

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch2. If ma97_switch3 is triggered it is disabled.

value	meaning
none	No scaling
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]

Default: mc64

ma97_scaling3 *(advanced)*: Third scaling. ↩

If ma97_scaling=dynamic, this scaling is used according to the trigger ma97_switch3.

value	meaning
none	No scaling
mc30	Scale linear system matrix using MC30
mc64	Scale linear system matrix using MC64
mc77	Scale linear system matrix using MC77 [1,3,0]

Default: mc64

ma97_small: Zero Pivot Threshold ↩

Any pivot less than ma97_small is treated as zero.

Range: [0, ∞]

Default: 1e-20

ma97_solve_blas3 *(advanced)*: Controls if blas2 or blas3 routines are used for solve ↩

value	meaning
no	Use BLAS2 (faster, some implementations bit incompatible)
yes	Use BLAS3 (slower)

Default: no

ma97_switch1 *(advanced)*: First switch, determine when ma97_scaling1 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling1 is enabled according to this condition. If ma97_switch2 occurs this option is henceforth ignored.

value	meaning
never	Scaling is never enabled.
at_start	Scaling to be used from the very start.
at_start_reuse	Scaling to be used on first iteration, then reused thereafter.
on_demand	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse	As on_demand, but reuse scaling from previous itr
high_delay	Scaling to be used after more than 0.05∗n delays are present

value	meaning
high_delay_reuse	Scaling to be used only when previous itr created more that 0.05∗n additional delays, otherwise reuse scaling from previous itr
od_hd	Combination of on_demand and high_delay
od_hd_reuse	Combination of on_demand_reuse and high_delay_reuse

Default: od_hd_reuse

ma97_switch2 *(advanced)*: Second switch, determine when ma97_scaling2 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling2 is enabled according to this condition. If ma97_switch3 occurs this option is henceforth ignored.

value	meaning
never	Scaling is never enabled.
at_start	Scaling to be used from the very start.
at_start_reuse	Scaling to be used on first iteration, then reused thereafter.
on_demand	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse	As on_demand, but reuse scaling from previous itr
high_delay	Scaling to be used after more than 0.05∗n delays are present
high_delay_reuse	Scaling to be used only when previous itr created more that 0.05∗n additional delays, otherwise reuse scaling from previous itr
od_hd	Combination of on_demand and high_delay
od_hd_reuse	Combination of on_demand_reuse and high_delay_reuse

Default: never

ma97_switch3 *(advanced)*: Third switch, determine when ma97_scaling3 is enabled. ↩

If ma97_scaling=dynamic, ma97_scaling3 is enabled according to this condition.

value	meaning
never	Scaling is never enabled.
at_start	Scaling to be used from the very start.
at_start_reuse	Scaling to be used on first iteration, then reused thereafter.
on_demand	Scaling to be used after Ipopt request improved solution (i.e. iterative refinement has failed).
on_demand_reuse	As on_demand, but reuse scaling from previous itr
high_delay	Scaling to be used after more than 0.05∗n delays are present
high_delay_reuse	Scaling to be used only when previous itr created more that 0.05∗n additional delays, otherwise reuse scaling from previous itr
od_hd	Combination of on_demand and high_delay
od_hd_reuse	Combination of on_demand_reuse and high_delay_reuse

Default: never

ma97_u: Pivoting Threshold ↩

See MA97 documentation.

Range: [0, 0.5]

Default: 1e-08

ma97_umax: Maximum Pivoting Threshold ↩

See MA97 documentation.

Range: [0, 0.5]

Default: 0.0001

max_cpu_time: Maximum number of CPU seconds. ↩

A limit on CPU seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding message.

Range: (0, ∞]

Default: 1e+20

max_filter_resets *(advanced)*: Maximal allowed number of filter resets ↩

A positive number enables a heuristic that resets the filter, whenever in more than "filter_reset_trigger" successive iterations the last rejected trial steps size was rejected because of the filter. This option determine the maximal number of resets that are allowed to take place.

Range: {0, ..., ∞}

Default: 5

max_hessian_perturbation: Maximum value of regularization parameter for handling negative curvature. ↩

In order to guarantee that the search directions are indeed proper descent directions, Ipopt requires that the inertia of the (augmented) linear system for the step computation has the correct number of negative and positive eigenvalues. The idea is that this guides the algorithm away from maximizers and makes Ipopt more likely converge to first order optimal points that are minimizers. If the inertia is not correct, a multiple of the identity matrix is added to the Hessian of the Lagrangian in the augmented system. This parameter gives the maximum value of the regularization parameter. If a regularization of that size is not enough, the algorithm skips this iteration and goes to the restoration phase. This is delta_w^max in the implementation paper.

Range: (0, ∞]

Default: 1e+20

max_iter: Maximum number of iterations. ↩

The algorithm terminates with a message if the number of iterations exceeded this number.

Range: {0, ..., ∞}

Default: GAMS iterlim

max_refinement_steps: Maximum number of iterative refinement steps per linear system solve. ↩

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the maximum number of iterative refinement steps.

Range: {0, ..., ∞}

Default: 10

max_resto_iter *(advanced)*: Maximum number of successive iterations in restoration phase. ↩

The algorithm terminates with an error message if the number of iterations successively taken in the restoration phase exceeds this number.

Range: {0, ..., ∞}

Default: 3000000

max_soc: Maximum number of second order correction trial steps at each iteration. ↩

Choosing 0 disables the second order corrections. This is p^{max} of Step A-5.9 of Algorithm A in the implementation paper.

Range: {0, ..., ∞}

Default: 4

max_soft_resto_iters *(advanced)*: Maximum number of iterations performed successively in soft restoration phase. ↩

If the soft restoration phase is performed for more than so many iterations in a row, the regular restoration phase is called.

Range: {0, ..., ∞}

Default: 10

max_wall_time: Maximum number of walltime clock seconds. ↩

A limit on walltime clock seconds that Ipopt can use to solve one problem. If during the convergence check this limit is exceeded, Ipopt will terminate with a corresponding message.

Range: (0, ∞]

Default: GAMS reslim

mehrotra_algorithm: Indicates whether to do Mehrotra's predictor-corrector algorithm. ↩

If enabled, line search is disabled and the (unglobalized) adaptive mu strategy is chosen with the "probing" oracle, and "corrector_type=affine" is used without any safeguards; you should not set any of those options explicitly in addition. Also, unless otherwise specified, the values of "bound_push", "bound_frac", and "bound_mult_init_val" are set more aggressive, and sets "alpha_for_y=bound_mult". The Mehrotra's predictor-corrector algorithm works usually very well for LPs and convex QPs.

Range: yes, no

Default: no

min_hessian_perturbation: Smallest perturbation of the Hessian block. ↩

The size of the perturbation of the Hessian block is never selected smaller than this value, unless no perturbation is necessary. This is delta_w^min in implementation paper.

Range: [0, ∞]

Default: 1e-20

min_refinement_steps: Minimum number of iterative refinement steps per linear system solve. ↩

Iterative refinement (on the full unsymmetric system) is performed for each right hand side. This option determines the minimum number of iterative refinements (i.e. at least "min_refinement_steps" iterative refinement steps are enforced per right hand side.)

Range: {0, ..., ∞}

Default: 1

mu_allow_fast_monotone_decrease *(advanced)*: Allow skipping of barrier problem if barrier test is already met. ↩

value	meaning
no	Take at least one iteration per barrier problem even if the barrier test is already met for the updated barrier parameter
yes	Allow fast decrease of mu if barrier test it met

Default: yes

mu_init: Initial value for the barrier parameter. ↩

This option determines the initial value for the barrier parameter (mu). It is only relevant in the monotone, Fiacco-McCormick version of the algorithm. (i.e., if "mu_strategy" is chosen as "monotone")

Range: (0, ∞]

Default: 0.1

mu_linear_decrease_factor: Determines linear decrease rate of barrier parameter. ↩

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of mu∗"mu_linear_decrease_factor" and mu^"superlinear_decrease_power". This is kappa_mu in implementation paper. This option is also used in the adaptive mu strategy during the monotone mode.

Range: (0, 1)

Default: 0.2

mu_max: Maximum value for barrier parameter. ↩

This option specifies an upper bound on the barrier parameter in the adaptive mu selection mode. If this option is set, it overwrites the effect of mu_max_fact. (Only used if option "mu_strategy" is chosen as "adaptive".)

Range: (0, ∞]

Default: 100000

mu_max_fact: Factor for initialization of maximum value for barrier parameter. ↩

This option determines the upper bound on the barrier parameter. This upper bound is computed as the average complementarity at the initial point times the value of this option. (Only used if option "mu_strategy" is chosen as "adaptive".)

Range: (0, ∞]

Default: 1000

mu_min: Minimum value for barrier parameter. ↩

This option specifies the lower bound on the barrier parameter in the adaptive mu selection mode. By default, it is set to the minimum of 1e-11 and min("tol","compl_inf_tol")/("barrier_tol_factor"+1), which should be a reasonable value. (Only used if option "mu_strategy" is chosen as "adaptive".)

Range: (0, ∞]

Default: 1e-11

mu_oracle: Oracle for a new barrier parameter in the adaptive strategy. ↩

Determines how a new barrier parameter is computed in each "free-mode" iteration of the adaptive barrier parameter strategy. (Only considered if "adaptive" is selected for option "mu_strategy").

value	meaning
probing	Mehrotra's probing heuristic
loqo	LOQO's centrality rule
quality-function	minimize a quality function

Default: quality-function

mu_strategy: Update strategy for barrier parameter. ↩

Determines which barrier parameter update strategy is to be used.

value	meaning
monotone	use the monotone (Fiacco-McCormick) strategy
adaptive	use the adaptive update strategy

Default: adaptive

mu_superlinear_decrease_power: Determines superlinear decrease rate of barrier parameter. ↩

For the Fiacco-McCormick update procedure the new barrier parameter mu is obtained by taking the minimum of mu∗"mu_linear_decrease_factor" and mu^"superlinear_decrease_power". This is theta_mu in implementation paper. This option is also used in the adaptive mu strategy during the monotone mode.

Range: (1, 2)

Default: 1.5

mu_target: Desired value of complementarity. ↩

Usually, the barrier parameter is driven to zero and the termination test for complementarity is measured with respect to zero complementarity. However, in some cases it might be desired to have Ipopt solve barrier problem for strictly positive value of the barrier parameter. In this case, the value of "mu_target" specifies the final value of the barrier parameter, and the termination tests are then defined with respect to the barrier problem for this value of the barrier parameter.

Range: [0, ∞]

Default: 0

mumps_dep_tol *(advanced)*: Threshold to consider a pivot at zero in detection of linearly dependent constraints with MUMPS. ↩

This is CNTL(3) in MUMPS.

Range: real

Default: 0

mumps_mem_percent: Percentage increase in the estimated working space for MUMPS. ↩

When significant extra fill-in is caused by numerical pivoting, larger values of mumps_mem_percent may help use the workspace more efficiently. On the other hand, if memory requirement are too large at the very beginning of the optimization, choosing a much smaller value for this option, such as 5, might reduce memory requirements.

Range: {0, ..., ∞}

Default: 1000

mumps_permuting_scaling: Controls permuting and scaling in MUMPS ↩

This is ICNTL(6) in MUMPS.

Range: {0, ..., 7}

Default: 7

mumps_pivot_order: Controls pivot order in MUMPS ↩

This is ICNTL(7) in MUMPS.

Range: {0, ..., 7}

Default: 7

mumps_pivtol: Pivot tolerance for the linear solver MUMPS. ↩

A smaller number pivots for sparsity, a larger number pivots for stability.

Range: [0, 1]

Default: 1e-06

mumps_pivtolmax: Maximum pivot tolerance for the linear solver MUMPS. ↩

Ipopt may increase pivtol as high as pivtolmax to get a more accurate solution to the linear system.

Range: [0, 1]

Default: 0.1

mumps_print_level: Debug printing level for the linear solver MUMPS ↩

0: no printing; 1: Error messages only; 2: Error, warning, and main statistic messages; 3: Error and warning messages and terse diagnostics; ≥4: All information.

Range: {0, ..., ∞}

Default: 0

mumps_scaling: Controls scaling in MUMPS ↩

This is ICNTL(8) in MUMPS.

Range: {-2, ..., 77}

Default: 77

neg_curv_test_reg: Whether to do the curvature test with the primal regularization (see Zavala and Chiang, 2014). ↩

value	meaning
yes	use primal regularization with the inertia-free curvature test
no	use original IPOPT approach, in which the primal regularization is ignored

Default: yes

neg_curv_test_tol: Tolerance for heuristic to ignore wrong inertia. ↩

If nonzero, incorrect inertia in the augmented system is ignored, and Ipopt tests if the direction is a direction of positive curvature. This tolerance is alpha_n in the paper by Zavala and Chiang (2014) and it determines when the direction is considered to be sufficiently positive. A value in the range of [1e-12, 1e-11] is recommended.

Range: [0, ∞]

Default: 0

nlp_scaling_constr_target_gradient *(advanced)*: Target value for constraint function gradient size. ↩

If a positive number is chosen, the scaling factors for the constraint functions are computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the constraint functions.

Range: [0, ∞]

Default: 0

nlp_scaling_max_gradient: Maximum gradient after NLP scaling. ↩

This is the gradient scaling cut-off. If the maximum gradient is above this value, then gradient based scaling will be performed. Scaling parameters are calculated to scale the maximum gradient back to this value. (This is g_max in Section 3.8 of the implementation paper.) Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Range: (0, ∞]

Default: 100

nlp_scaling_method: Select the technique used for scaling the NLP. ↩

Selects the technique used for scaling the problem internally before it is solved. For user-scaling, the parameters come from the NLP.

value	meaning
none	no problem scaling will be performed
gradient-based	scale the problem so the maximum gradient at the starting point is nlp_scaling_max_gradient
equilibration-based	scale the problem so that first derivatives are of order 1 at random points (GAMS/Ipopt: requires user-provided library with HSL routine MC19)

Default: gradient-based if GAMS scaleopt is not set, otherwise none

nlp_scaling_min_value: Minimum value of gradient-based scaling values. ↩

This is the lower bound for the scaling factors computed by gradient-based scaling method. If some derivatives of some functions are huge, the scaling factors will otherwise become very small, and the (unscaled) final constraint violation, for example, might then be significant. Note: This option is only used if "nlp_scaling_method" is chosen as "gradient-based".

Range: [0, ∞]

Default: 1e-08

nlp_scaling_obj_target_gradient *(advanced)*: Target value for objective function gradient size. ↩

If a positive number is chosen, the scaling factor for the objective function is computed so that the gradient has the max norm of the given size at the starting point. This overrides nlp_scaling_max_gradient for the objective function.

Range: [0, ∞]

Default: 0

nu_inc *(advanced)*: Increment of the penalty parameter. ↩

Range: (0, ∞]

Default: 0.0001

nu_init *(advanced)*: Initial value of the penalty parameter. ↩

Range: (0, ∞]

Default: 1e-06

obj_max_inc *(advanced)*: Determines the upper bound on the acceptable increase of barrier objective function. ↩

Trial points are rejected if they lead to an increase in the barrier objective function by more than obj_max_inc orders of magnitude.

Range: (1, ∞]

Default: 5

pardiso_iter_coarse_size *(advanced)*: Maximum Size of Coarse Grid Matrix ↩

DPARM(3)

Range: {1, ..., ∞}

Default: 5000

pardiso_iter_dropping_factor *(advanced)*: dropping value for incomplete factor ↩

DPARM(5)

Range: (0, 1)

Default: 0.5

pardiso_iter_dropping_schur *(advanced)*: dropping value for sparsify schur complement factor ↩

DPARM(6)

Range: (0, 1)

Default: 0.1

pardiso_iter_inverse_norm_factor *(advanced)*: ↩

DPARM(8)

Range: (1, ∞]

Default: 5e+06

pardiso_iter_max_levels *(advanced)*: Maximum Size of Grid Levels ↩

DPARM(4)

Range: {1, ..., ∞}

Default: 10

pardiso_iter_max_row_fill *(advanced)*: max fill for each row ↩

DPARM(7)

Range: {1, ..., ∞}

Default: 10000000

pardiso_iter_relative_tol *(advanced)*: Relative Residual Convergence ↩

DPARM(2)

Range: (0, 1)

Default: 1e-06

pardiso_iterative *(advanced)*: Switch for iterative solver in Pardiso library ↩

Range: yes, no

Default: no

pardiso_matching_strategy: Matching strategy to be used by Pardiso ↩

This is IPAR(13) in Pardiso manual.

value	meaning
complete	Match complete (IPAR(13)=1)
complete+2x2	Match complete+2x2 (IPAR(13)=2)
constraints	Match constraints (IPAR(13)=3)

Default: complete+2x2

pardiso_max_droptol_corrections *(advanced)*: Maximal number of decreases of drop tolerance during one solve. ↩

This is relevant only for iterative Pardiso options.

Range: {1, ..., ∞}

Default: 4

pardiso_max_iter *(advanced)*: Maximum number of Krylov-Subspace Iteration ↩

DPARM(1)

Range: {1, ..., ∞}

Default: 500

pardiso_max_iterative_refinement_steps: Limit on number of iterative refinement steps. ↩

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

Range: {-∞, ..., ∞}

Default: 0

pardiso_msglvl: Pardiso message level ↩

This is MSGLVL in the Pardiso manual.

Range: {0, ..., ∞}

Default: 0

pardiso_order: Controls the fill-in reduction ordering algorithm for the input matrix. ↩

value	meaning
amd	minimum degree algorithm
one	
metis	MeTiS nested dissection algorithm
pmetis	parallel (OpenMP) version of MeTiS nested dissection algorithm
four	
five	

Default: metis

pardiso_redo_symbolic_fact_only_if_inertia_wrong *(advanced)*: Toggle for handling case when elements were perturbed by Pardiso. ↩

value	meaning
no	Always redo symbolic factorization when elements were perturbed

value	meaning
yes	Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

Default: no

pardiso_repeated_perturbation_means_singular *(advanced)*: Whether to assume that matrix is singular if elements were perturbed after recent symbolic factorization. ↩

Range: yes, no

Default: no

pardiso_skip_inertia_check *(advanced)*: Whether to pretend that inertia is correct. ↩

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Range: yes, no

Default: no

pardisolib: Name of library containing Pardiso routines (from pardiso-project.org) for load at runtime ↩

Range: string

Default: libpardiso.so (Linux), libpardiso.dylib (macOS), libpardiso.dll (Windows)

pardisomkl_matching_strategy: Matching strategy to be used by Pardiso ↩

This is IPAR(13) in Pardiso manual.

value	meaning
complete	Match complete (IPAR(13)=1)
complete+2x2	Match complete+2x2 (IPAR(13)=2)
constraints	Match constraints (IPAR(13)=3)

Default: complete+2x2

pardisomkl_max_iterative_refinement_steps: Limit on number of iterative refinement steps. ↩

The solver does not perform more than the absolute value of this value steps of iterative refinement and stops the process if a satisfactory level of accuracy of the solution in terms of backward error is achieved. If negative, the accumulation of the residue uses extended precision real and complex data types. Perturbed pivots result in iterative refinement. The solver automatically performs two steps of iterative refinements when perturbed pivots are obtained during the numerical factorization and this option is set to 0.

Range: {-∞, ..., ∞}

Default: 1

pardisomkl_msglvl: Pardiso message level ↩

This is MSGLVL in the Pardiso manual.

Range: {0, ..., ∞}

Default: 0

pardisomkl_order: Controls the fill-in reduction ordering algorithm for the input matrix. ↩

value	meaning
amd	minimum degree algorithm
one	undocumented
metis	MeTiS nested dissection algorithm
pmetis	parallel (OpenMP) version of MeTiS nested dissection algorithm

Default: metis

pardisomkl_redo_symbolic_fact_only_if_inertia_wrong *(advanced)*: Toggle for handling case when elements were perturbed by Pardiso. ↩

value	meaning
no	Always redo symbolic factorization when elements were perturbed
yes	Only redo symbolic factorization when elements were perturbed if also the inertia was wrong

Default: no

pardisomkl_repeated_perturbation_means_singular *(advanced)*: Whether to assume that matrix is singular if elements were perturbed after recent symbolic factorization. ↩

Range: yes, no

Default: no

pardisomkl_skip_inertia_check *(advanced)*: Whether to pretend that inertia is correct. ↩

Setting this option to "yes" essentially disables inertia check. This option makes the algorithm non-robust and easily fail, but it might give some insight into the necessity of inertia control.

Range: yes, no

Default: no

perturb_always_cd *(advanced)*: Active permanent perturbation of constraint linearization. ↩

Enabling this option leads to using the delta_c and delta_d perturbation for the computation of every search direction. Usually, it is only used when the iteration matrix is singular.

Range: yes, no

Default: no

perturb_dec_fact: Decrease factor for x-s perturbation. ↩

The factor by which the perturbation is decreased when a trial value is deduced from the size of the most recent successful perturbation. This is kappa_w^- in the implementation paper.

Range: (0, 1)

Default: 0.333333

perturb_inc_fact: Increase factor for x-s perturbation. ↩

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of all perturbations except for the first. This is kappa_w^+ in the implementation paper.

Range: (1, ∞]

Default: 8

perturb_inc_fact_first: Increase factor for x-s perturbation for very first perturbation. ↩

The factor by which the perturbation is increased when a trial value was not sufficient - this value is used for the computation of the very first perturbation and allows a different value for the first perturbation than that used for the remaining perturbations. This is bar_kappa_w^+ in the implementation paper.

Range: (1, ∞]

Default: 100

print_advanced_options *(advanced)*: whether to print also advanced options ↩

Range: yes, no

Default: no

print_eval_error: Switch to enable printing information about function evaluation errors into the GAMS listing file. ↩

Range: no, yes

Default: yes

print_frequency_iter: Determines at which iteration frequency the summarizing iteration output line should be printed. ↩

Summarizing iteration output is printed every print_frequency_iter iterations, if at least print_frequency_time seconds have passed since last output.

Range: {1, ..., ∞}

Default: 1

print_frequency_time: Determines at which time frequency the summarizing iteration output line should be printed. ↩

Summarizing iteration output is printed if at least print_frequency_time seconds have passed since last output and the iteration number is a multiple of print_frequency_iter.

Range: [0, ∞]

Default: 0

print_info_string: Enables printing of additional info string at end of iteration output. ↩

This string contains some insider information about the current iteration. For details, look for "Diagnostic Tags" in the Ipopt documentation.

Range: yes, no

Default: no

print_level: Output verbosity level. ↩

Sets the default verbosity level for console output. The larger this value the more detailed is the output.

Range: {0, ..., 12}

Default: 5

print_options_mode: format in which to print options documentation ↩

value	meaning
text	Ordinary text
latex	LaTeX formatted
doxygen	Doxygen (markdown) formatted

Default: text

print_timing_statistics: Switch to print timing statistics. ↩

If selected, the program will print the time spend for selected tasks. This implies timing_statistics=yes.

Range: yes, no

Default: no

quality_function_balancing_term *(advanced)*: The balancing term included in the quality function for centrality. ↩

This determines whether a term is added to the quality function that penalizes situations where the complementarity is much smaller than dual and primal infeasibilities. Only used if option "mu_oracle" is set to "quality-function".

value	meaning
none	no balancing term is added
cubic	Max(0,Max(dual_inf,primal_inf)-compl)^3

Default: none

quality_function_centrality *(advanced)*: The penalty term for centrality that is included in quality function. ↩

This determines whether a term is added to the quality function to penalize deviation from centrality with respect to complementarity. The complementarity measure here is the xi in the Loqo update rule. Only used if option "mu_oracle" is set to "quality-function".

value	meaning
none	no penalty term is added
log	complementarity $*$ the log of the centrality measure
reciprocal	complementarity $*$ the reciprocal of the centrality measure
cubed-reciprocal	complementarity $*$ the reciprocal of the centrality measure cubed

Default: none

quality_function_max_section_steps: Maximum number of search steps during direct search procedure determining the optimal centering parameter. ↩

The golden section search is performed for the quality function based mu oracle. Only used if option "mu_oracle" is set to "quality-function".

Range: {0, ..., ∞}

Default: 8

quality_function_norm_type *(advanced)*: Norm used for components of the quality function. ↩

Only used if option "mu_oracle" is set to "quality-function".

value	meaning
1-norm	use the 1-norm (abs sum)
2-norm-squared	use the 2-norm squared (sum of squares)
max-norm	use the infinity norm (max)
2-norm	use 2-norm

Default: 2-norm-squared

quality_function_section_qf_tol *(advanced)*: Tolerance for the golden section search procedure determining the optimal centering parameter (in the function value space). ↩

The golden section search is performed for the quality function based mu oracle. Only used if option "mu_oracle" is set to "quality-function".

Range: [0, 1)

Default: 0

quality_function_section_sigma_tol *(advanced)*: Tolerance for the section search procedure determining the optimal centering parameter (in sigma space). ↩

The golden section search is performed for the quality function based mu oracle. Only used if option "mu_oracle" is set to "quality-function".

Range: [0, 1)

Default: 0.01

recalc_y: Tells the algorithm to recalculate the equality and inequality multipliers as least square estimates. ↩

This asks the algorithm to recompute the multipliers, whenever the current infeasibility is less than recalc_y_feas_tol. Choosing yes might be helpful in the quasi-Newton option. However, each recalculation requires an extra factorization of the linear system. If a limited memory quasi-Newton option is chosen, this is used by default.

value	meaning
no	use the Newton step to update the multipliers
yes	use least-square multiplier estimates

Default: no

recalc_y_feas_tol: Feasibility threshold for recomputation of multipliers. ↩

If recalc_y is chosen and the current infeasibility is less than this value, then the multipliers are recomputed.

Range: (0, ∞]

Default: 1e-06

replace_bounds *(advanced)*: Whether all variable bounds should be replaced by inequality constraints ↩

This option must be set for the inexact algorithm.

Range: yes, no

Default: no

report_mininfeas_solution: Switch to report intermediate solution with minimal constraint violation to GAMS if the final solution is not feasible. ↩

This option allows to obtain the most feasible solution found by Ipopt during the iteration process, if it stops at a (locally) infeasible solution, due to a limit (time, iterations, ...), or with a failure in the restoration phase.

Range: no, yes

Default: no

required_infeasibility_reduction: Required reduction of infeasibility before leaving restoration phase. ↩

The restoration phase algorithm is performed, until a point is found that is acceptable to the filter and the infeasibility has been reduced by at least the fraction given by this option.

Range: [0, 1)

Default: 0.9

residual_improvement_factor *(advanced)*: Minimal required reduction of residual test ratio in iterative refinement. ↩

If the improvement of the residual test ratio made by one iterative refinement step is not better than this factor, iterative refinement is aborted.

Range: (0, ∞]

Default: 1

residual_ratio_max *(advanced)*: Iterative refinement tolerance ↩

Iterative refinement is performed until the residual test ratio is less than this tolerance (or until "max_refinement_steps" refinement steps are performed).

Range: (0, ∞]

Default: 1e-10

residual_ratio_singular *(advanced)*: Threshold for declaring linear system singular after failed iterative refinement. ↩

If the residual test ratio is larger than this value after failed iterative refinement, the algorithm pretends that the linear system is singular.

Range: (0, ∞]

Default: 1e-05

resto_failure_feasibility_threshold *(advanced)*: Threshold for primal infeasibility to declare failure of restoration phase. ↩

If the restoration phase is terminated because of the "acceptable" termination criteria and the primal infeasibility is smaller than this value, the restoration phase is declared to have failed. The default value is actually 1e2∗tol, where tol is the general termination tolerance.

Range: [0, ∞]

Default: 0

resto_penalty_parameter *(advanced)*: Penalty parameter in the restoration phase objective function. ↩

This is the parameter rho in equation (31a) in the Ipopt implementation paper.

Range: (0, ∞]

Default: 1000

resto_proximity_weight *(advanced)*: Weighting factor for the proximity term in restoration phase objective. ↩

This determines how the parameter zeta in equation (29a) in the implementation paper is computed. zeta here is resto_proximity_weight∗sqrt(mu), where mu is the current barrier parameter.

Range: [0, ∞]

Default: 1

rho *(advanced)*: Value in penalty parameter update formula. ↩

Range: (0, 1)

Default: 0.1

s_max *(advanced)*: Scaling threshold for the NLP error. ↩

See paragraph after Eqn. (6) in the implementation paper.

Range: (0, ∞]

Default: 100

s_phi *(advanced)*: Exponent for linear barrier function model in the switching rule. ↩

See Eqn. (19) in the implementation paper.

Range: (1, ∞]

Default: 2.3

s_theta *(advanced)*: Exponent for current constraint violation in the switching rule. ↩

See Eqn. (19) in the implementation paper.

Range: (1, ∞]

Default: 1.1

sigma_max *(advanced)*: Maximum value of the centering parameter. ↩

This is the upper bound for the centering parameter chosen by the quality function based barrier parameter update. Only used if option "mu_oracle" is set to "quality-function".

Range: (0, ∞]

Default: 100

sigma_min *(advanced)*: Minimum value of the centering parameter. ↩

This is the lower bound for the centering parameter chosen by the quality function based barrier parameter update. Only used if option "mu_oracle" is set to "quality-function".

Range: [0, ∞]

Default: 1e-06

skip_corr_if_neg_curv *(advanced)*: Whether to skip the corrector step in negative curvature iteration. ↩

The corrector step is not tried if negative curvature has been encountered during the computation of the search direction in the current iteration. This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Range: yes, no

Default: yes

skip_corr_in_monotone_mode *(advanced)*: Whether to skip the corrector step during monotone barrier parameter mode. ↩

The corrector step is not tried if the algorithm is currently in the monotone mode (see also option "barrier_strategy"). This option is only used if "mu_strategy" is "adaptive". Changing this option is experimental.

Range: yes, no

Default: yes

slack_bound_frac: Desired minimum relative distance from the initial slack to bound. ↩

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_push"). (This is kappa_2 in Section 3.6 of implementation paper.)

Range: (0, 0.5]

Default: 0.01

slack_bound_push: Desired minimum absolute distance from the initial slack to bound. ↩

Determines how much the initial slack variables might have to be modified in order to be sufficiently inside the inequality bounds (together with "slack_bound_frac"). (This is kappa_1 in Section 3.6 of implementation paper.)

Range: (0, ∞]

Default: 0.01

slack_move *(advanced)*: Correction size for very small slacks. ↩

Due to numerical issues or the lack of an interior, the slack variables might become very small. If a slack becomes very small compared to machine precision, the corresponding bound is moved slightly. This parameter determines how large the move should be. Its default value is mach_eps^{3/4}. See also end of Section 3.5 in implementation paper - but actual implementation might be somewhat different.

Range: [0, ∞]

Default: 1.81899e-12

soc_method: Ways to apply second order correction ↩

This option determines the way to apply second order correction, 0 is the method described in the implementation paper. 1 is the modified way which adds alpha on the rhs of x and s rows.

Range: {0, ..., 1}

Default: 0

soft_resto_pderror_reduction_factor: Required reduction in primal-dual error in the soft restoration phase. ↩

The soft restoration phase attempts to reduce the primal-dual error with regular steps. If the damped primal-dual step (damped only to satisfy the fraction-to-the-boundary rule) is not decreasing the primal-dual error by at least this factor, then the regular restoration phase is called. Choosing "0" here disables the soft restoration phase.

Range: [0, ∞]

Default: 0.9999

start_with_resto: Whether to switch to restoration phase in first iteration. ↩

Setting this option to "yes" forces the algorithm to switch to the feasibility restoration phase in the first iteration. If the initial point is feasible, the algorithm will abort with a failure.

Range: yes, no

Default: no

tau_min *(advanced)*: Lower bound on fraction-to-the-boundary parameter tau. ↩

This is tau_min in the implementation paper. This option is also used in the adaptive mu strategy during the monotone mode.

Range: (0, 1)

Default: 0.99

theta_max_fact *(advanced)*: Determines upper bound for constraint violation in the filter. ↩

The algorithmic parameter theta_max is determined as theta_max_fact times the maximum of 1 and the constraint violation at initial point. Any point with a constraint violation larger than theta_max is unacceptable to the filter (see Eqn. (21) in the implementation paper).

Range: (0, ∞]

Default: 10000

theta_min_fact *(advanced)*: Determines constraint violation threshold in the switching rule. ↩

The algorithmic parameter theta_min is determined as theta_min_fact times the maximum of 1 and the constraint violation at initial point. The switching rule treats an iteration as an h-type iteration whenever the current constraint violation is larger than theta_min (see paragraph before Eqn. (19) in the implementation paper).

Range: (0, ∞]

Default: 0.0001

timing_statistics: Indicates whether to measure time spend in components of Ipopt and NLP evaluation ↩

The overall algorithm time is unaffected by this option.

Range: yes, no

Default: no

tiny_step_tol *(advanced)*: Tolerance for detecting numerically insignificant steps. ↩

If the search direction in the primal variables (x and s) is, in relative terms for each component, less than this value, the algorithm accepts the full step without line search. If this happens repeatedly, the algorithm will terminate with a corresponding exit message. The default value is 10 times machine precision.

Range: [0, ∞]

Default: 2.22045e-15

tiny_step_y_tol *(advanced)*: Tolerance for quitting because of numerically insignificant steps. ↩

If the search direction in the primal variables (x and s) is, in relative terms for each component, repeatedly less than tiny_step_tol, and the step in the y variables is smaller than this threshold, the algorithm will terminate.

Range: [0, ∞]

Default: 0.01

tol: Desired convergence tolerance (relative). ↩

Determines the convergence tolerance for the algorithm. The algorithm terminates successfully, if the (scaled) NLP error becomes smaller than this value, and if the (absolute) criteria according to "dual_inf_tol", "constr_viol_tol", and "compl_inf_tol" are met. This is epsilon_tol in Eqn. (6) in implementation paper. See also "acceptable_tol" as a second termination criterion. Note, some other algorithmic features also use this quantity to determine thresholds etc.

Range: (0, ∞]

Default: 1e-08

warm_start_bound_frac: same as bound_frac for the regular initializer ↩

Range: (0, 0.5]

Default: 0.001

warm_start_bound_push: same as bound_push for the regular initializer ↩

Range: (0, ∞]

Default: 0.001

warm_start_init_point: Warm-start for initial point ↩

Indicates whether this optimization should use a warm start initialization, where values of primal and dual variables are given (e.g., from a previous optimization of a related problem.)

value	meaning
no	do not use the warm start initialization
yes	use the warm start initialization

Default: yes, if run on modified model instance (e.g., from GUSS), otherwise no

warm_start_mult_bound_push: same as mult_bound_push for the regular initializer ↩

Range: (0, ∞]

Default: 0.001

warm_start_mult_init_max: Maximum initial value for the equality multipliers. ↩

Range: real

Default: 1e+06

warm_start_slack_bound_frac: same as slack_bound_frac for the regular initializer ↩

Range: (0, 0.5]

Default: 0.001

warm_start_slack_bound_push: same as slack_bound_push for the regular initializer ↩

Range: (0, ∞]

Default: 0.001

warm_start_target_mu *(advanced)*: ↩

Experimental!

Range: real

Default: 0

watchdog_shortened_iter_trigger: Number of shortened iterations that trigger the watchdog. ↩

If the number of successive iterations in which the backtracking line search did not accept the first trial point exceeds this number, the watchdog procedure is activated. Choosing "0" here disables the watchdog procedure.

Range: {0, ..., ∞}

Default: 10

watchdog_trial_iter_max: Maximum number of watchdog iterations. ↩

This option determines the number of trial iterations allowed before the watchdog procedure is aborted and the algorithm returns to the stored point.

Range: {1, ..., ∞}

Default: 3

5.21 JAMS and LogMIP

5.21.1 Introduction

EMP (Extended Mathematical Programming) is not a solver but an (experimental) framework for automated mathematical programming reformulations. The idea behind EMP is that new upcoming types of models which currently cannot be solved reliably are reformulated into models of established math programming classes in order to use mature solver technology. At this stage, EMP supports the modeling of Bilevel Programs, Variational Inequalities, Disjunctive Programs, Extended Nonlinear Programs and Embedded Complementarity Systems.

Extended mathematical programs are collections of functions and variables joined together using specific optimization and complementarity primitives. EMP annotates the existing relationships within a model to facilitate higher level structure identification. A specific implementation of this framework is outlined that reformulates the original GAMS model automatically using directives contained in an "empinfo" file into an equivalent model that can be solved using existing GAMS solvers.

The reformulation is done by the solver JAMS which currently is the only solver that is capable of handling EMP models. Examples showing how to use the EMP framework and the solver JAMS are made available through the GAMS EMP Library which is included in the GAMS Distribution. In order to generate a copy of and EMPLIB model, one can use the library facility of the GAMS IDE, or execute the command line directive `emplib <modelname>`, where modelname is the (stem of the) file containing the model.

EMP has been developed jointly by Michael Ferris of UW-Madison, Ignacio Grossmann of Carnegie Mellon University, and GAMS Development Corporation. EMP and JAMS come free of charge with any licensed GAMS system but require a subsolver to solve the generated models.

5.21.2 JAMS: a reformulation tool

EMP models are currently processed by the JAMS solver. The solver JAMS creates a scalar version of the given GAMS model. This scalar version of the model is then solved by an appropriate subsolver. By default, there are no reformulations carried out, so the model generated is simply a GAMS scalar form of the model the actual subsolver will process. The subsolver used is by default the currently specified solver for the given model type.

5.21.2.1 The JAMS Option File

As with any GAMS solver, JAMS has an option file, typically called `jams.opt`. A JAMS option subsolver is available to change the subsolver used for the reformulated model, along with an option to utilize a subsolver option file (subsolveropt).

The actual scalar version of the model can also be seen by the modeler using the option filename. For example, the option file

```
subsolver path
subsolveropt 1
filename mcpmod.gms
```

when applied to an EMP model that is a complementarity problem will create a file called `mcpmod.gms` in the current directory and solve that model using the solver PATH utilizing any options for PATH that are specified in `path.opt`. The scalarized model is not particularly useful to look at since all of the original variables have been renamed into a scalar form. The mapping between original variables and the ones used in the scalar version of the model is given in a dictionary file that can also be seen by the modeler using the dict option. If a user simply wants to generate this scalar model, then the option terminate will not solve the generated model.

After the scalar version of the model is solved, the solution values are mapped back into the original namespace and returned to the modeler as usual in the listing file. The JAMS option margtol allows the modeler to suppress reporting marginals that have (absolute) values smaller than this tolerance.

Obviously, all of the above functionality is not of much value: the key part of JAMS is to interpret additional directives to take the original model and produce a *reformulated* scalar model. This is carried out using an "empinfo" file. The syntax and use of this file is the content of the remaining sections of this document.

The option EMPInfoFile allows the user to specify the path and name of a file containing additional EMP information. The syntax of this file is described by examples elsewhere in this document and in section Empinfo file details . There is a subtlety that should be mentioned when a user writes this file in the default location in the GAMS scratch directory of the current run specified in the gams file using:

```
file empinfo / '%emp.info%' /;
```

wherein certain additional formatting instructions are given. If instead, the file handle is given by:

```
file empinfo / 'empinfo.txt' /;
```

for example, then to produce the same file the following lines must be added to the gams source file:

```
empinfo.pc = 8;
empinfo.pw = 255;
```

The following tables list all available options.

5.21.2.2 Reformulation Options

Option	Description	Default
KeepObj	Keep original objective function and variables in generated MCP model. This option is only valid for reformulations into complementarity systems. If KeepObj is set the generated MCP program will incorporate the original objective function and the Karush-Kuhn-Tucker condition of the original objective variable.	0
MargTol	Only report marginals with an absolute value above the tolerance	1e-6
ObjVarName	Name of objective variable in generated model.	objvar

5.21.2.3 General Options

Option	Description	Default
CompModel	Complementarity model (MCP or VI) to generate for equilibrium problems.	mcp
Dict	Generate a dictionary file that maps original variable and equation names to the ones of the scalar JAMS model.	dict.txt
DisjBinRelax	Relax requirement that disjunction variables are binary.	0
EMPInfoFile	Path and name of file containing additional EMP information as disjunctions, perpendiculars, bilevel characterization etc. By default, the file is expected to exist in the GAMS scratch directory of the current run (gams.scrdir). If this option is used, EMP searches for the file based on the provided relative or absolute path. In case of a given relative path, it is assumed to be relative to the working directory.	
FileName	Filename of generated scalar reformulated GAMS model.	emp.gms
ImplVarModel	Reformulation model to generate for implicit variables. Replication: Shared variables are replicated for each agent Substitution: Shared multipliers are substituted out: assumes implicit function Switching: Switch function-variable matching to avoid replication	switching
NLConsToFunc	Stick nonlinear constraints into functional part when generating a VI model for equilibrium problems	0
SharedEqu	Allow shared equations in equilibrium problems	0
SubSolvePar	User defined GAMS parameters for subsolve	
SubSolver	Subsolver used to solve the reformulated model. The subsolver chosen has to be suitable for the chosen reformulation type. The user can also provide a solver related option file, see SubSolverOpt.	
SubSolverOpt	Optfile value to pass to the subsolver Range: {1, ..., 999}	1
Terminate	Generate the GAMS source code of the reformulated model in a file and terminate without solving the model.	0
UserPFFile	Filename of extra GAMS options when running the reformulated GAMS model.	
ZipDebug	Zip-file name to create if the internal GAMS model fails	none

5.21.3 Forming Optimality Conditions: NLP2MCP

The first nontrivial use of the JAMS solver is to automatically generate the first order conditions of a linear or nonlinear program; essentially we *reformulate* the optimization problem as a mixed complementarity problem (MCP). The "empinfo" file to do this simply contains the following line:

```
modeltype mcp
```

Behind the scenes, JAMS forms the Lagrangian of the nonlinear program and then forms its Karush-Kuhn-Tucker optimality conditions. To be clear, given the original nonlinear program

$$\min_x f(x) \text{ s.t. } g(x) \leq 0,\ h(x) = 0, \tag{NLP}$$

the Lagrangian is:

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \langle \lambda, g(x) \rangle - \langle \mu, h(x) \rangle.$$

The first order conditions are the following MCP:

$$\begin{array}{lcl} 0 = \nabla_x \mathcal{L}(x, \lambda, \mu) & \perp & x \text{ free} \\ 0 \geq -\nabla_\lambda \mathcal{L}(x, \lambda, \mu) & \perp & \lambda \leq 0 \\ 0 = -\nabla_\mu \mathcal{L}(x, \lambda, \mu) & \perp & \mu \text{ free} \end{array}$$

A specific example is:

$$\begin{array}{ll} \min\limits_{x,y,z} & -3x + y \\ \text{s.t.} & x + y \leq 1,\ x + y - z = 2,\ x, y \geq 0 \end{array}$$

which is found in the EMPLIB model **nlp2mcp**:

```
Positive Variables x, y;
Variables f, z;
Equations g, h, defobj;
g..      x + y     =l= 1;
h..      x + y - z =e= 2;
defobj.. f         =e= -3*x + y;
model comp / defobj, g, h /;
file info / '%emp.info%' /;
putclose info / 'modeltype mcp';
solve comp using emp minimizing f;
```

The putclose line writes out the default "empinfo" file whose location is provided in the system string `%emp.info%`. Armed with this additional information, the EMP tool automatically creates the following MCP:

$$\begin{array}{lcl} 0 \leq -3 - \lambda - \mu & \perp & x \geq 0 \\ 0 \leq 1 - \lambda - \mu & \perp & y \geq 0 \\ 0 = \mu & \perp & z \text{ free} \\ 0 \geq x + y - 1 & \perp & \lambda \leq 0 \\ 0 = x + y - z - 2 & \perp & \mu \text{ free.} \end{array}$$

5.21.4 Soft Constraints

In many cases, we wish to relax certain constraints in a model during solution (to help identify feasibility issues for example). As an example, consider the problem

$$\begin{aligned} \min_{x_1,x_2,x_3} \quad & \exp(x_1) \\ \text{s.t.} \quad & \log(x_1) = 1, \\ & x_2^2 \leq 2, \\ & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0, \end{aligned}$$

which can be formulated in GAMS as

```
$title simple example of ENLP
Variables obj, x1, x2, x3;
Equations f0, f1, f2, f3, f4;
f0.. obj       =e= exp(x1);
f1.. log(x1)   =e= 1;
f2.. sqr(x2)   =g= 2;
f3.. x1/x2     =e= log(x3);
f4.. 3*x1 + x2 =l= 5;
x1.lo = 0; x2.lo = 0;
model enlpemp /all/;
x1.l = 1; x2.l = 1; x3.l = 1;
solve enlpemp using nlp min obj;
```

5.21.4.1 Reformulation as a classical NLP

Soft constraints allow us to treat certain equations in the model as "soft" by removing the constraints and adding a penalty term to the objective function. Explicitly, we replace the above problem by:

$$\begin{aligned} \min_{x_1,x_2,x_3} \quad & \exp(x_1) + 5\left\|\log(x_1) - 1\right\|^2 + 2\max(x_2^2 - 2, 0) \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0. \end{aligned}$$

In this problem, we still force $x_1/x_2 = \log(x_3)$, but apply a least squares penalty to $\log(x_1) - 1$ and a smaller one-sided penalization to $x_2^2 - 2$.

The above formulation is nonsmooth due to the "max" term in the objective function; in practice we would replace this by:

$$\begin{aligned} \min_{x_1,x_2,x_3,w} \quad & \exp(x_1) + 5\left(\log(x_1) - 1\right)^2 + 2w \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0 \\ & w \geq x_2^2 - 2, w \geq 0 \end{aligned}$$

and recover a standard form NLP.

The "empinfo" file:

```
modeltype NLP
adjustequ
f1 sqr 5
f2 maxz 2
```

coupled with replacing the last line with

```
solve enlpemp using emp min obj;
```

achieves this goal. The parameter values provide the penalty coefficients above.

5.21.4.2 Reformulation as an MCP

As an alternative, we can rewrite the problem as an MCP, also dealing explicitly with the nonsmoothness. The "empinfo" file is given by:

```
modeltype NLP
adjustequ
f1 sqr 5
f2 maxz 2
```

and this generates the following MCP:

$$\begin{array}{lcl}
0 = \log(x_1) - 1 + y_1/10 & \perp & y_1 \text{ free}, \\
0 \le x_2^2 - 2 & \perp & y_2 \ge 0, \\
0 = x_1/x_2 - \log(x_3) & \perp & y_3 \text{ free}, \\
0 \ge 3x_1 + x_2 - 5 & \perp & y_4 \le 0, \\
0 \le exp(x_1) - y_1/x_1 - y_3/x_2 - 3y_4 & \perp & x_1 \ge 0, \\
0 \le -2y_2x_2 + x_1y_3/x_2^2 - y_4 & \perp & x_2 \ge 0, \\
0 = y_3/x_3 & \perp & x_3 \text{ free},
\end{array}$$

where y represent the multipliers.

A complete description of the process to derive this MCP will be given later in section Extended Nonlinear Programs .

5.21.5 Bilevel Programs

Mathematical programs with optimization problems in their constraints have a long history in operations research including [15] [29] [71] . New codes are being developed that exploit this structure, at least for simple hierarchies, and attempt to define and implement algorithms for their solution.

The simplest case is that of bilevel programming, where an upper level problem depends on the solution of a lower level optimization. For example:

$$\begin{array}{ll}
\min\limits_{x,y} & f(x,y) \\
\text{s.t.} & g(x,y) \le 0, \\
& y \text{ solves } \min\limits_{y} v(x,y) \text{ s.t.} \quad h(x,y) \ge 0.
\end{array}$$

Often, the upper level is referred to as the "leader", while the lower level is the "follower".

This problem can be reformulated as a Mathematical Program with Complementarity Constraints (MPCC) by replacing the lower level optimization problem by its first order optimality conditions:

$$\begin{array}{ll}
\min\limits_{x,y} & f(x,y) \\
\text{s.t.} & g(x,y) \le 0, \\
& 0 = \nabla_y v(x,y) - \lambda^T \nabla_y h(x,y) \perp x \text{ free} \\
& 0 \le h(x,y) \perp \lambda \ge 0.
\end{array}$$

We find a solution of the MPCC, not of the bilevel program. This approach allows the MPCC to be solved using the NLPEC code, for example. Note that this reformulation is potentially problematic. First order

conditions require theoretical assumptions to be necessary and sufficient for *local optimality*. There may be cases where the lower level problem has multiple local solutions, but the modeler really was interested in the *global* solution. The approach here may not produce this solution, even if a global solver is used within NLPEC.

The following example is Example 5.1.1, page 197 from [15] . Mathematically, the problem is

$$\begin{aligned}
&\min_{x,y} x - 4y \\
&\text{s.t. } y \text{ solves } \min_{y} y \\
&\qquad\qquad \text{s.t. } x + y \geq 3 \\
&\qquad\qquad\qquad 2x - y \geq 0 \\
&\qquad\qquad\qquad -2x - y \geq -12 \\
&\qquad\qquad\qquad -3x + 2y \geq -4
\end{aligned}$$

and the EMPLIB model **bard511** contains the following code:

```
Positive variables x, y;
Variables objout, objin;
equations defout, defin, e1, e2, e3, e4;
defout.. objout  =e= x - 4*y;
defin..  objin   =e= y;
e1..     x +   y =g=   3;
e2..   2*x -   y =g=   0;
e3..  -2*x -   y =g= -12;
e4..  -3*x + 2*y =g=  -4;
model bard / all /;
$echo bilevel x min objin y defin e1 e2 e3 e4 > "%emp.info%"
solve bard using emp minimizing objout;
```

Note that first the functions that form the objectives and constraints of the model are defined and the model is assembled. The `$echo` line writes the "empinfo" file and states that the lower level problem involves the objective `objin` which is to be minimized by choice of variables `y` subject to the constraints specified in (`defin`), `e1`, `e2`, `e3`, and `e4`.

Note that the variables x are declared to be variables of the upper level problem and this example has no upper level constraints g. Having written the problem in this way, the MPCC is generated automatically, and passed on to a solver. In the case where that solver is NLPEC, a further reformulation of the model is carried out to convert the MPCC into an equivalent NLP or a parametric sequence of NLP's.

Further examples of bilevel models in **EMPLIB** are named: `bard`$*$, `ccmg74`, `ccmg153`, `flds`$*$, `jointc1`, `jointc2`, `mirrlees`, `transbp`.

The EMP model type allows multiple lower level problems to be specified within the bilevel format. An example of this is given in EMPLIB as **ccmg71**. The equations and objectives are specified in the normal manner; the only change is the definition of the "empinfo" file, shown below:

```
...
defh1.. h1 =e= sqr(u1-x1) + sqr(u2-x2) + sqr(u3-x3) + sqr(u4-x4);
e1.. 3*u1 + u2 + 2*u3 + u4 =e= 6;
...
$onecho > "%emp.info%"
bilevel x1 x2 x3 x4
min h1 u1 u2 u3 u4 defh1 e1
min h2 v1 v2 v3 v4 defh2 e2
$offecho
```

This corresponds to a bilevel program with two followers, both solving minimization problems. The first follower minimizes the objective function `h1` (defined in `defh1`) over the variables `u1`, `u2`, `u3`, and `u4` subject to the constraint given in `e1`. The second followers problem is defined analogously next. Note that `h1` involves the variables `x1`, `x2`, `x3`, and `x4` that are optimization variables of the leader. The constraint in `e1` could also include these variables, and also the variables `v1`, `v2`, `v3`, or `v4` of the second follower, but all of these would be treated as parameters by the first follower.

The actual model (**ccmg71**) in EMPLIB uses a shortcut notation:

```
...
defh1.. h1 =e= sqr(u1-x1) + sqr(u2-x2) + sqr(u3-x3) + sqr(u4-x4);
```

```
e1.. 3*u1 + u2 + 2*u3 + u4 =e= 6;
...
$onecho > "%emp.info%"
bilevel x1 x2 x3 x4
min h1 * defh1 e1
min h2 * defh2 e2
$offecho
```

In the followers problem, the '$*$' notation indicates that this agent will optimize over all the variables used in `defh1` and `e1` that are not under the control of any other follower or the leader. In this case, this means `u1`, `u2`, `u3`, and `u4`. To avoid confusion, it is recommended that the modeler explicity names all the variables in each followers problem as shown before.

5.21.6 Variational Inequalities

A variational inequality $VI(F, X)$ is to find $x \in X$:

$$\langle F(x), (z - x)\rangle \geq 0, \text{ for all } z \in X.$$

Here X is a closed (frequently assumed convex) set, defined for example as

$$X = \{x \mid x \geq 0, h(x) \geq 0\}.$$

Note that the first-order (minimum principle) conditions of a nonlinear program

$$\min_{z \in X} f(z)$$

are precisely of this form with $F(x) = \nabla f(x)$.

It is well known that such problems can be reformulated as complementarity problems when the set X has the representation $\{x \mid x \geq 0, h(x) \geq 0\}$ by introducing multipliers λ on the constraints $h(x) \geq 0$:

$$\begin{array}{lcl} 0 \leq F(x) - \langle \lambda, \nabla h(x)\rangle & \perp & x \geq 0 \\ 0 \leq h(x) & \perp & \lambda \geq 0. \end{array}$$

If X has a different representation, this construction would be modified appropriately.

A simple two dimensional example may be useful to improve understanding. Let

$$F(x) = \begin{bmatrix} x_1 + 2 \\ x_1 + x_2 - 3 \end{bmatrix}, \; X = \{x \geq 0\}\, x_1 + x_2 \leq 1,$$

so that F is an affine function, but F is not the gradient of any function $f : \mathbf{R}^2 \to \mathbf{R}$. For this particular data, $VI(F, X)$ has a unique solution $x = (0, 1)$.

```
Set J  / 1, 2 /;
Positive Variable x(J)  'vars, perp to f(J)';
Equations F(J), h;
F(J)..  (x('1') + 2)$sameas(J,'1') + (x('1') + x('2') - 3)$sameas(J,'2') =n= 0 ;
h..     x('1') + x('2') =l= 1;
model simpleVI / F, h/;
file fx /"%emp.info%"/;
putclose fx 'vi F x h';
solve simpleVI using emp;
```

Note that the first lines of this file define the F and h using standard GAMS syntax and include the defining equations in the model `simpleVI`. The extension is the annotation "empinfo" file that indicates certain equations are to be treated differently by the EMP tool. The annotation simply says that the model is a VI (`vi`) that pairs `F` with `x` and that the remaining (unpaired) equations form the constraint set X. (Alternative notation allows the keyword `vi` to be replaced by `vifunc`.) Thus, model equations `F` define a function F that is to be part of a variational inequality, while the equations `h` define constraints of X. It is also acceptable in this setting to use the "empinfo" file defined by:

```
putclose fx 'vi F x';
```

In this case, by default, any equations that are given in the model statement but not included as a pair in the `vi` statement are automatically used to form X. An alternative way to write this model without using `sameas` is given in EMPLIB as **affinevi**.

Some subtleties related to VI's are demonstrated in the EMPLIB model **zerofunc**. In this model, the set X is defined using variables y and z, for which z does not appear in the definition of $F \equiv F_y$. In this case, the variable z is then matched with a "0" function. The "empinfo" file can be written in a number of different ways:

```
putclose fx 'vi F_y y';
```

or

```
putclose fx 'vi z F_y y';
```

or

```
putclose fx 'vi z F_y y gCons';
```

or

```
putclose fx 'vi F_z z F_y y gCons';
```

where `F_z` simply defines the zero function. To some extent, our preferred notation is the one listed third: it explicitly includes all the variables and constraints that are present in the model and does not require the modeler to define `F_z` at all.

Further example models in EMPLIB are **simplevi**, **simplevi2**, **simplevi3**, **target**, **traffic**, **traffic2**, and **transvi**.

Note also that the lower level problems of a bilevel program could be VI's instead of optimization problems - these problems are called Mathematical Programs with Equilibrium Constraints (MPEC) in the literature. Note that since MPCC is a special case of MPEC, the GAMS model type MPEC covers both. An example demonstrating this setup is given in EMPLIB as **multmpec**. The actual model to solve is:

$$\begin{aligned}
\min_{u,v,w,z}\ & z \\
\text{s.t.}\ & \exp(z) + w = 2, z \geq 1 \\
& (u,v) \text{ solves VI}([v+w+z-1; u-\log(v)], \{(u,v) \mid u \geq 0, v \geq 0\}) \\
& w \text{ solves VI}(w+z+3, \{w \mid w \text{ free}\})
\end{aligned}$$

Note that the two VI's (due to the definitional sets) correspond respectively to a complementarity problem:

$$\begin{aligned}
0 \leq v+w+z-1 \quad & \perp \quad u \geq 0 \\
0 \leq u-\log(v) \quad & \perp \quad v \geq 0
\end{aligned}$$

and a linear equation:

$$w+z+3=0$$

The actual GAMS code is as follows:

```
Positive Variable u;
Variables v, w, z;
Equations f1, f2, f3, h;
f1.. v + w + z  =n= 1;
f2.. u          =n= log(v);
f3.. w + z      =n= -3;
h..  exp(z) + w =e= 2;
v.lo = 0; v.l = 1; z.lo = 1;
model mpecmod /all/;
$onecho > %emp.info%
bilevel
 vi f1 u
    f2 v
 vi f3 w
$offecho
solve mpecmod using emp min z;
```

The initial level value for `v` (`v.l = 1`) is needed to protect the evaluation of $\log(v)$. The two complementarity problems are specified in the empinfo file (it is not really necessary to split these VI's apart, but it may give information to a solver that can be exploited). It is of course possible to write the MPCC directly in GAMS using the model type MPEC instead of EMP.

5.21.7 Embedded Complementarity Systems

A different type of embedded optimization model that arises frequently in applications is:

$$\begin{array}{lll} \max\limits_{x} & f(x,y) & \\ \text{s.t.} & g(x,y) \leq 0 & (\perp\ p \geq 0) \\ & H(x,y,p) = 0 & (\perp\ y \text{ free}) \end{array}$$

Note the difference here: the optimization problem is over the variable x, and is parameterized by the variable y. The choice of y is fixed by the (auxiliary) complementarity relationships depicted here by H. Note that the H equations are not part of the optimization problem, but are essentially auxiliary constraints to tie down remaining variables in the model.

A specific example is:

$$\begin{array}{lll} \max\limits_{x} & x & \\ \text{s.t.} & x + y \leq 1 & \\ & -3x + y = 0.5 & (\perp\ y \text{ free}) \end{array}$$

which is found in the EMPLIB model **simpequil2**:

```
Variables y;
Positive variables x;
Equations optcons, vicons;
optcons.. x + y =l= 1;
vicons.. -3*x + y =e= 0.5;
model comp / optcons, vicons /;
file info / '%emp.info%' /;
put info / 'equilibrium';
put      / 'max x optcons';
putclose / 'vi vicons y';
solve comp using emp;
```

In order that this model can be processed correctly as an EMP, the modeler provides additional annotations to the model defining equations in an "empinfo" file. Specifically, first it is indicated that the problem is an equilibrium problem involving one or more agent problems. Next, the first agent is defined as an optimizer (over `x`). Finally, the second agent is defined as solving a VI in `y`. Armed with this additional information, the EMP tool automatically creates the following MCP:

$$\begin{array}{lll} 0 \leq -1 + p & \perp & x \geq 0 \\ 0 \leq 1 - x - y & \perp & p \geq 0 \\ 0 = -3x + y - 0.5 & \perp & y \text{ free}, \end{array}$$

(which is formed by the steps we outline below). EMP explicitly enforces the rule that every variable and constraint is under the control of exactly one agent. Thus a constraint or a variable cannot appear in both the max problem and the VI problem.

The above example is slightly simpler than the general form described above in which H is a function of x, y, and p, the multiplier on the constraint of the optimization problem. The problem is that we do not have that variable around in the model code if we only specify the optimization problem there. This occurs for example in the classical PIES Model due to Hogan. In this setting, the problem is described by a linear program

$$\begin{array}{ll} \min_x & c^T x \\ \text{s.t.} & Ax = q(p) \\ & Bx = b \\ & x \geq 0 \end{array}$$

in which the quantity q is a function of p, which is a multiplier on one of the LP constraints. To do this in EMP, we simply add the annotation:

```
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;
file myinfo /'%emp.info%'/;
put myinfo 'equilibrium ';
put 'min obj c o ct ot lt ht defobj dembal cmbal ombal lmbal hmbal ruse';
putclose 'dualvar p dembal';
solve piesemp using emp;
```

where `dembal` is the name of the constraint for which `p` needs to be the multiplier. The full model is found in the EMPLIB model **pies**. Two final points: the `dualvar` directive identifies the variable `p` with the multiplier on the `dembal` constraint, and all variables and constraints must be owned by a single agent. In this case, since there is only one agent (the minimizer), all constraints of the model are explicitly claimed, along with all variables except for `p`. However, next, `p` is identified with the `dembal` constraint, which is owned by the min agent, and hence `p` is also owned by that agent.

There are several shorthands possible here. The first is that the explicit statement of variables can be replaced by the '$*$' form:

```
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;
file myinfo /'%emp.info%'/;
put myinfo 'equilibrium ';
put 'min obj * defobj dembal cmbal ombal lmbal hmbal ruse ';
```

Alternatively, an even shorter version is possible since there is only one agent present in this model, namely:

```
model piesemp / defobj, dembal, cmbal, ombal, lmbal, hmbal, ruse /;
file myinfo /'%emp.info%'/;
putclose myinfo 'dualvar p dembal';
solve piesemp using emp minimizing obj;
```

Note that in this form, all the variables and constraints of the original model are included in the (single) agents problem, and the original variable `p` is identified in the constructed MCP with the multiplier on the `dembal` constraint.

In the general case where the "empinfo" file contains all three lines:

```
min x optcons
vi vicons y
dualvar p optcons
```

namely that the function H that is defined in `vicons` is complementary to the variable `y` (and hence the variable `y` is a parameter to the optimization problem), and furthermore that the dual variable associated with the equation optcons in the optimization problem is one and the same as the variable `p` used to define H, the EMP tool automatically creates the following MCP:

$$\begin{array}{lcl} 0 = \nabla_x \mathcal{L}(x,y,p) & \perp & x \text{ free} \\ 0 \geq -\nabla_p \mathcal{L}(x,y,p) & \perp & p \leq 0 \\ 0 = H(x,y,p) & \perp & y \text{ free,} \end{array}$$

where the Lagrangian is defined as

$$\mathcal{L}(x, y, p) = f(x, y) - \langle p, g(x, y)\rangle.$$

Essentially, this MCP consists of the first order optimality conditions of the optimization problem, coupled with the VI that is the second agents problem. An example that does both of these things together is provided in EMPLIB as **scarfemp-primal**.

Note that since the PIES model has no y variables, this is a special case of the general form in which the second agents (VI) problem is simply not present.

Example models are **ferris43**, **flipper pies**, **scarfemp-dual**, **simpequil**, **transecs**, and **transeql**.

5.21.8 MOPECs

MOPECs (Multiple Optimization Problems with Equilibrium Constraints) are a simple conceptual extension of the aforementioned embedded complementarity system. Instead of having a single optimizing agent and a variational inequality, one instead allows multiple optimizing agents coupled with an equilibrium constraint (the variational inequality).

Perhaps the most popular use of this formulation is where competition is allowed between agents. A standard method to deal with such cases is via the concept of Nash Games. In this setting x^* is a Nash Equilibrium if

$$x_i^* \in \arg\min_{x_i \in X_i} \ell_i(x_i, x_{-i}^*, q), \forall i \in \mathcal{I},$$

where x_{-i} are other players decisions and the quantities q are given exogenously, or via complementarity:

$$0 \leq H(x, q) \quad \perp \quad q \geq 0.$$

This mechanism is extremely popular in economics, and Nash famously won the Nobel Prize for his contributions to this literature.

This format is again an EMP, more general than the example given above in two respects. Firstly, there is more than one optimization problem specified in the embedded complementarity system. Secondly, the parameters in each optimization problem consist of two types. Firstly, there are the variables q that are tied down by the auxiliary complementarity condition and hence are treated as parameters by the ith Nash player. Also there are the variables x_{-i} that are treated as parameters by the ith Nash player, but are treated as variables by a different player j.

While we do not specify the syntax here for these issues, **EMPLIB** provides examples that outline how to carry out this matching within GAMS. Finally, two points of note: first it is clear that the resulting model is a complementarity problem and can be solved using PATH, for example. Secondly, performing the conversion from an embedded complementarity system or a Nash Game automatically is a critical step in making such models practically useful.

We note that there is a large literature on discrete-time finite-state stochastic games: this has become a central tool in analysis of strategic interactions among forward-looking players in dynamic environments. The model of dynamic competition in an oligopolistic industry given in [55] is exactly in the format described above, and has been used extensively in applications such as advertising, collusion, mergers, technology adoption, international trade and finance. Ongoing work aims to use the EMP format to model these problems.

5.21.9 Extended Nonlinear Programs

Optimization models have traditionally been of the form (NLP). Specialized codes have allowed certain problem structures to be exploited algorithmically, for example simple bounds on variables. However, for the most part, assumptions of smoothness of f, g and h are required for many solvers to process these problems effectively. In a series of papers, Rockafellar and colleagues [152] [153] [154] have introduced the notion of extended nonlinear programming, where the (primal) problem has the form:

$$\min_{x \in X} f(x) + \theta(g_1(x), \ldots, g_m(x)). \tag{ENLP}$$

In this setting, X is assumed to be a nonempty polyhedral set, and the functions $f, g_1, \ldots, g_m$ are smooth. The function θ can be thought of as a generalized penalty function that may well be nonsmooth. However, when θ has the form

$$\theta(u) = \sup_{y \in Y} \{\langle y, u \rangle - k(y)\}, \tag{θ conj}$$

a computationally exploitable and theoretically powerful framework can be developed based on conjugate duality. A key point for computation and modeling is that the function θ can be fully described by defining the set Y and the function k. Furthermore, from a modeling perspective, an extended nonlinear program can be specified simply by defining the functions $f, g_1, \ldots, g_m$ in the manner already provided by the modeling system, with the additional issue of simply defining Y and k. Conceptually, this is not much harder than what is carried out already, but leads to significant enhancements to the types of models that are available. Once a modeler determines which constraints are treated via which choice of k and Y, the EMP model interface automatically forms an equivalent variational inequality or complementarity problem. As we show later, there may be alternative formulations that are computationally more appealing; such reformulations can be generated using different options to JAMS.

5.21.9.1 Forms of theta

The EMP model type makes the problem format (ENLP) available to users in GAMS. As special cases, we can model piecewise linear penalties, least squares, and L_1 approximation problems, as well as the notion of soft and hard constraints.

For ease of exposition, we now describe a subset of the types of functions θ that can be generated by particular choices of Y and k. In many cases, the function θ is separable, that is

$$\theta(u) = \sum_{i=1}^{m} \theta_i(u_i),$$

so we can either specify θ_i or θ itself.

Extended nonlinear programs include the classical nonlinear programming form (NLP) as a special case. This follows from the observation that if K is a closed convex cone, and we let ψ_K denote the "indicator function" of K defined by:

$$\psi_K(u) = \begin{cases} 0 & \text{if } u \in K, \\ \infty & \text{else,} \end{cases}$$

then (NLP) can be rewritten as:

$$\min_x f(x) + \psi_K((g(x), h(x)), \qquad K = \mathbf{R}^m_- \times \{0\}^p,$$

where m and p are the dimensions of g and h respectively and $\mathbf{R}^m_- = \{u \in \mathbf{R}^m \mid u \leq 0\}$. An elementary calculation shows that

$$\psi_K(u) = \sup_{v \in K^\circ} \langle u, v \rangle,$$

where $K^\circ = \{u \mid \langle u, v\rangle \leq 0, \forall v \in K\}$ is the polar cone of the given cone K. Thus, when $\theta(u) = \psi_K(u)$ we simply take $k \equiv 0$ and $Y = K^\circ$. In our example, $K^\circ = \mathbf{R}^m_+ \times \mathbf{R}^p$. To some extent, this is just a formalism that allows us to claim the classical case as a specialization; however when we take the cone K to be more general than the polyhedral cone used above, we can generate conic programs for example.

The second example involves a piecewise linear function θ: Formally, for $u \in \mathbf{R}$,

$$\theta(u) = \begin{cases} \rho u & \text{if } u \geq 0, \\ \sigma u & \text{else.} \end{cases}$$

In this case, simple calculations prove that θ has the form (θ conj) for the choices

$$k \equiv 0 \text{ and } Y = [\sigma, \rho].$$

The special case where $\sigma = -\rho$ results in

$$\theta(u) = \rho\,|u|\,.$$

This allows us to model nonsmooth L_1 approximation problems. Another special case results from the choice of $\sigma = -\gamma$, $\rho = 0$, whereby

$$\theta(u) = \gamma \max\{-u, 0\}.$$

This formulation corresponds to a soft penalization on an inequality constraint, namely if $\theta(-g_1(x))$ is used then nothing is added to the objective function if $g_1(x) \leq 0$, but $\gamma g_1(x)$ is added if the constraint $g_1(x) \leq 0$ is violated. Contrast this to the classical setting above, where ∞ is added to the objective if the inequality constraint is violated. It is interesting to see that truncating the set Y, which amounts to bounding the multipliers, results in replacing the classical constraint by a linearized penalty.

The third example involves a more interesting choice of k. If we wish to replace the "absolute value" penalization given above by a quadratic penalization (as in classical least squares analysis), that is

$$\theta(u) = \gamma u^2$$

then a simple calculation shows that we should take

$$k(y) = \frac{1}{4\gamma} y^2 \text{ and } Y = \mathbf{R}.$$

By simply specifying this different choice of k and Y we can generate such models easily and quickly within the modeling system. Note, however, that the reformulation we would use in $\theta(u) = \rho\,|u|$ and $\theta(u) = \gamma u^2$ are very different as we shall explain in the simple example below. Furthermore, in many applications it has become popular to penalize violations using a quadratic penalty only within a certain interval, afterwards switching to a linear penalty (chosen to make the penalty function θ continuously differentiable - see [99]). That is:

$$\theta(u) = \begin{cases} \gamma u - \frac{1}{2}\gamma^2 & \text{if } u \geq \gamma \\ \frac{1}{2} u^2 & \text{if } u \in [-\gamma, \gamma] \\ -\gamma u - \frac{1}{2}\gamma^2 & \text{else.} \end{cases}$$

Such functions arise from quadratic k and simple bound sets Y. In particular, the somewhat more general function

$$\theta(u) = \begin{cases} \gamma\beta^2 + \rho(u - \beta) & \text{if } u \geq \beta \\ \gamma u^2 & \text{if } u \in [\alpha, \beta] \\ \gamma\alpha^2 + \sigma(u - \alpha) & \text{else} \end{cases}$$

arises from the choice of

$$k(y) = \frac{1}{4\gamma} y^2 \text{ and } Y = [\sigma, \rho],$$

with $\alpha = \frac{\sigma}{2\gamma}$ and $\beta = \frac{\rho}{2\gamma}$.

The final example that we give is that of L_∞ penalization. This example is different to the examples given above in that θ is not separable. However, straightforward calculation can be used to show

$$\theta(u) = \max_{i=1,\ldots,m} u_i$$

results from the choice of

$$k \equiv 0 \text{ and } Y = \left\{ y \in \mathbf{R}^m \mid y \geq 0, \sum_{i=1}^m y_i = 1 \right\},$$

that is, Y is the unit simplex.

5.21.9.2 Underlying theory

The underlying structure of θ leads to a set of extended optimality conditions and an elegant duality theory. This is based on an extended form of the Lagrangian:

$$\mathcal{L}(x,y) = f(x) + \sum_{i=1}^{m} y_i g_i(x) - k(y)$$
$$x \in X, y \in Y$$

Note that the Lagrangian $\mathcal{L}$ is smooth - all the nonsmoothness is captured in the θ function. The theory is an elegant combination of calculus arguments related to g_i and its derivatives, and variational analysis for features related to θ.

It is shown in [153] that under a standard constraint qualification, the first-order conditions of (ENLP) are precisely in the form of the following variational inequality:

$$\text{VI}\left(\begin{bmatrix} \nabla_x \mathcal{L}(x,y) \\ -\nabla_y \mathcal{L}(x,y) \end{bmatrix}, X \times Y\right). \tag{ENLP VI}$$

When X and Y are simple bound sets, this is simply a complementarity problem.

Note that EMP exploits this result. In particular, if an extended nonlinear program of the form (ENLP) is given to EMP, then the optimality conditions (ENLP VI) are formed as a variational inequality problem and can be processed as outlined above. For a specific example, we cite the fact that if we use the (classical) choice $k \equiv 0$ and $Y = K^\circ$, then the optimality conditions of (ENLP) are precisely the standard complementarity problem given as (ENLP VI). While this is of interest, we believe that other choices of k and Y may be more useful and lead to models that have more practical significance.

Under appropriate convexity assumptions on this Lagrangian, it can be shown that a solution of the VI (ENLP VI) is a saddle point for the Lagrangian on $X \times Y$. Furthermore, in this setting, the saddle point generates solutions to the primal problem (ENLP) and its dual problem:

$$\max_{y \in Y} d(y), \quad \text{where } d(y) = \inf_{x \in X} \mathcal{L}(x,y),$$

with no duality gap.

Unfortunately, the perturbations y in Rockafellar's theory are precisely the negative of those used throughout the GAMS system. Thus, we need to replace y by $-y$ throughout in the above to recover the same multipliers as those GAMS uses.

5.21.9.3 A simple example

As an example, consider the problem

$$\begin{aligned} \min_{x_1,x_2,x_3} \quad & \exp(x_1) + 5\|\log(x_1) - 1\|^2 + 2\max(x_2^2 - 2, 0) \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \le 5, x_1 \ge 0, x_2 \ge 0. \end{aligned}$$

In this problem, we would take

$$X = \left\{x \in \mathbf{R}^3 \mid 3x_1 + x_2 \le 5, x_1 \ge 0, x_2 \ge 0\right\}.$$

The function θ essentially treats 3 separable pieces:

$$\begin{aligned} g_1(x) &= \log(x_1) - 1, \\ g_2(x) &= x_2^2 - 2, \\ g_3(x) &= x_1/x_2 - \log(x_3). \end{aligned}$$

A classical problem would force $g_1(x) = 0$, $g_2(x) \leq 0$. and $g_3(x) = 0$, while minimizing $f(x) = \exp(x_1)$. In our problem, we still force $g_3(x) = 0$, but apply a (soft) least squares penalty on $g_1(x)$ and a smaller one-sided penalization on $g_2(x)$. The above formulation is nonsmooth due to the "max" term in the objective function; in practice we could replace this by:

$$\begin{aligned} \min_{x_1,x_2,x_3,w} \quad & \exp(x_1) + 5\|\log(x_1) - 1\|^2 + 2w \\ \text{s.t.} \quad & x_1/x_2 = \log(x_3), \\ & 3x_1 + x_2 \leq 5, x_1 \geq 0, x_2 \geq 0 \\ & w \geq x_2^2 - 2, w \geq 0 \end{aligned}$$

and recover a standard form NLP. If the penalty on $g_1(x)$ would be replaced by a one-norm penalization (instead of least squares), we would have to play a similar game, moving the function $g_1(x)$ into the constraints and adding additional variable(s). To some extent, this seems unnatural - a modeler should be able to interchange the penalization without having to reformulate the problem from scratch. The proposed extended NLP would not be reformulated at all by the modeler, but allows all these "generalized constraints" to be treated in a similar manner within the modeling system. The actual formulation would take:

$$\theta(u) = \theta_1(u_1) + \theta_2(u_2) + \theta_3(u_3)$$

where

$$\begin{aligned} \theta_1(u_1) &= 5u_1^2, \\ \theta_2(u_2) &= 2\max(u_2, 0), \\ \theta_3(u_3) &= \psi_{\{0\}}(u_3). \end{aligned}$$

The discussion above allows us to see that

$$\begin{aligned} Y &= -(\mathbf{R} \times [0, 2] \times \mathbf{R}), \\ k(y) &= \frac{1}{20}y_1^2 + 0 + 0. \end{aligned}$$

The corresponding Lagrangian is the smooth function:

$$\mathcal{L}(x, y) = f(x) - \sum_{i=1}^{3} y_i g_i(x) - k(y).$$

The corresponding VI (ENLP VI) can almost be formulated in GAMS (except that the linear constraint in X cannot be handled currently except by introducing a $\theta_4(x)$). Thus

$$g_4(x) = 3x_1 + x_2 - 5, \ \theta_4(u) = \psi_{\mathbf{R}_-}$$

resulting in the following choices for Y and k:

$$\begin{aligned} Y &= -(\mathbf{R} \times [0, 2] \times \mathbf{R} \times \mathbf{R}_+), \\ k(y) &= \frac{1}{20}y_1^2 + 0 + 0 + 0. \end{aligned}$$

Since X and Y are now simple bound sets, (ENLP VI) is now a complementarity problem and can be solved for example using PATH. A simple "empinfo" file details the choices of Y and k from the implemented library:

```
Adjustequ
e1 sqr 5
e2 MaxZ 2
```

5.21.9.4 Reformulation as a classical NLP

Suppose

$$\theta(u) = \sup_{y \in Y} \{\langle u, y\rangle - \frac{1}{2}\langle y, Qy\rangle\}$$

for a polyhedral set $Y \in \mathbf{R}^m$ and a symmetric positive semidefinite $Q \in \mathbf{R}^{m\times m}$ (possibly $Q = 0$).

Suppose further that

$$X = \{x \mid Rx \leq r \ \}, Y = \left\{y \mid S^T y \leq s\right\},$$
$$Q = DJ^{-1}D^T, \ \ F(x) = (g_1(x), \ldots, g_m(x)),$$

where J is symmetric and positive definite (for instance $J = I$). Then, as outlined by [154] , the optimal solutions $\overline{x}$ of (ENLP) are the $\bar{x}$ components of the optimal solutions $(\bar{x}, \bar{z}, \bar{w})$ to

$$\begin{array}{ll} \min & f(x) + \langle s, z\rangle + \frac{1}{2}\langle w, Jw\rangle \\ \text{s.t.} & Rx \leq r, z \geq 0, F(x) - Sz - Dw = 0. \end{array}$$

The multiplier on the equality constraint in the usual sense is the multiplier associated with $\bar{x}$ in the extended Lagrangian for (ENLP). (Note that a Cholesky factorization may be needed to determine D.)

It may be better to solve this reformulated NLP than to solve the VI (ENLP VI). However, it is important that we can convey all types of nonsmooth optimization problems to a solver as smooth optimization problems, and hence it is important to communicate the appropriate structure to the solver interface. We believe that specifying Y and k is a theoretically sound way to do this.

5.21.10 Disjunctive Programs (LogMIP)

There are many ways that the EMP model type can be used for further extensions to the modeling capabilities of a given system. In particular, the procedures outlined in [190] [73] for disjunctive programming extensions are also implemented within the EMP model type.

The disjunctive programming procedures are also known as the LogMIP 2.0 solver for solving linear and nonlinear disjunctive programming problems involving binary variables and disjunction definitions for modeling discrete choices. While the modeling and solution of these disjunctive optimization problems has not yet reached the stage of maturity and reliability as LP, MIP and NLP modeling, these problems have a rich area of applications.

LogMIP 2.0 has been developed by Dr. Aldo Vecchietti from INGAR (Santa Fe, Argentina) and Professor Ignacio E. Grossmann from Carnegie Mellon University (Pittsburgh, USA), and supersedes its previous version, LogMIP 1.0 (GAMS releases 22.6 (December 2007) to 23.6 (December 2010)). Changes in version 2.0 are at the level of language, where now the EMP syntax and modeltype is used, and at the level of solvers, where Big-M and convex-hull relaxations are combined. For more information see the website http://www.logmip.ceride.gov.ar/ and the documentation http://www.logmip.ceride.gov.ar/files/pdfs/newUserManual.pdf.

One simple example to highlight the disjunctive programming feature is the notion of an ordering of tasks, namely that either job i completes before job j starts or the converse, i.e., that the jobs cannot overlap. Such a disjunction can be specified using an empinfo file containing lines:

```
disjunction * seq(i,j) else seq(j,i)
```

In such an example, one can implement a Big-M method, employ indicator constraints, or utilize a convex hull reformulation. The convex hull reformulation is the default strategy; to utilize the Big-M formulation, the additional option

```
default bigm 1000
```

would add binary variables and constraints to impose the disjunction using a Big-M value of 1000. Alternatively, the option setting

```
default indic
```

writes out a model and an option file that implements a reformulation using indicator constraints, which can be handled by the solvers CPLEX, SCIP, and XPRESS. The EMPLIB model **sequence** is sequencing model that implements all of these options.

More complicated (nonlinear) examples make the utility of this approach clearer. The design of a multiproduct batch plan with intermediate storage described in [189] and a synthesis problem involving 8 processes from [186] are also included in the EMP model library. As a final example, the gasoline emission model outlined in [72] is precisely in the form that could exploit the features of EMP related to (nonlinear) disjunctive programming.

5.21.11 Empinfo file details

We have explained above many of the standard vectorized ways to write an "empinfo" file. The "empinfo" file has a vectorized format and a more powerful (but more complex) scalar version. We describe some of the features of the scalar version in this section.

An example of the use of the scalar syntax is given below

```
file info / '%emp.info%' /;
put info / 'equilibrium';
loop(h,
  put / 'min' obj(h) /;
  loop(j, put x(h,j)); put /;
  loop(k, put z(h,k));
  put / costdef(h) /;
  loop(i, put nodebal(h,i)); put /;
);
loop(a, put 'vi',flowdef(a),f(a) /);
```

This is an example of a MOPEC in which the optimization agents are indexed by h and each of these agents control variables x_{hj} and z_{hk}. The objective function for each h is defined in `costdef`, the constraints of each minimization problem are defined in `nodebal`, and the equilibrium constraints that link all the optimization problems are a VI defined by variables f_a and functions within `flowdef`. Sometimes it is necessary to use syntax that generates the "tl" fields of sets, but this seems only necessary when abnormal side conditions are needed.

The format of the empinfo file is given below:

```
Disjunction [chull [big eps] | bigM [big eps threshold] | indic]
            [NOT] var|* [NOT] {equ} {ELSEIF [NOT] var|* [NOT] {equ}} [ELSE [NOT] {equ}]

Default [chull [big eps] | bigM [big eps threshold] | indic]

ParallelStep1 {equ|*}

AdjustEqu equ abs|sqr|maxz|... {weight}
```

```
ModelType MCP|NLP|MIP|...

BiLevel {var} {MAX|MIN obj {var|*} {[-] equ}} {VI {var|*} {[-] equ var} {[-] equ}}
              {DualVar {var [-] equ}}

Equilibrium {Implicit {var equ}} {VIsol {equ}}
            {MAX|MIN obj {var|*} {[-] equ}} {VI {var|*} {[-] equ var} {[-] equ}}
            {DualVar {var [-] equ}}

VI {var|*} {[-] equ var} {[-] equ}

DualEqu {[-] equ var}

DualVar {var [-] equ}

-------
[ ] optional      | exclusive      { } can be repeated
```

5.22 KESTREL - Remote Solver Execution on NEOS Servers

5.22.1 Background

The Kestrel client/server is a way of sending your GAMS solve job to be solved via the NEOS Server from within your usual GAMS modeling environment and receiving results that can be processed as with any local solver. Starting with distribution 23.7 the Kestrel solver is part of the GAMS distribution. The solve statement using the GAMS/KESTREL solver invokes a client program that sends your problem to a solver running on one of the NEOS Server's remote computers. The results from the NEOS Server are eventually returned through Kestrel to GAMS, where you can view and manipulate them locally in the usual way. The introduction below covers everything you need to know to start using GAMS/KESTREL. Further information about more advanced features and other uses of Kestrel can be found at the Kestrel page on the NEOS Server. Please also make you familiar with the NEOS optimization service terms of use before submitting jobs via GAMS/KESTREL.

5.22.2 Using GAMS/KESTREL

The Kestrel solver can be used to solve a GAMS model remotely. For example, consider the **trnsport model**. It can be solved locally in GAMS through the following statements,

```
model transport /all/;
solve transport using lp minimizing z;
```

which specify the `trnsport` model and solve it with the default linear programming solver. We can add an option statement to the code to explicitly specify the solver. For example, if we change the linear programming solver to GAMS/MINOS, the code becomes

```
model transport /all/;
option lp=minos;
solve transport using lp minimizing z;
```

To solve the same problem remotely through the NEOS Server, we simply change the linear programming solver to Kestrel. In addition we have to provide an email address. This can be specified either in an option file or by setting the environment variable NEOS_EMAIL. In order to set this permanently, it is recommended to add NEOS_EMAIL to your gamsconfig.yaml file.

Without an option file Kestrel will submit the model instance to the NEOS server and use the default LP solver on NEOS.

```
option lp=kestrel;
solve transport using lp minimizing z;
```

We can support a Kestrel option file and determine the solver on the NEOS server:

```
transport.optfile=1;
option lp=kestrel;
solve transport using lp minimizing z;
$echo email jdoe@jeangreyhigh.edu > kestrel.opt
$echo kestrel_solver minos >> kestrel.opt
```

The statement transport.optfile=1 specifies that an options file, called kestrel.opt, will be used. The options file contains the email address, the remote solver name as well as any options for the remote solver. We instruct the Kestrel solver to use GAMS/MINOS as the remote solver by writing the following kestrel.opt file:

```
email jdoe@jeangreyhigh.edu
kestrel_solver minos
```

If you do not know what solvers are available via GAMS/KESTREL on NEOS, submitting a job with a nonexistent solver set will return a list of enabled solvers. If you want to change the URL of the server, you can specify the option neos_server. The complete format of the parameter is protocol://host:port.

A subsequent run of the code through the GAMS interpreter results in the transport model being solved through the NEOS Server with the GAMS/MINOS solver. Once the job is submitted to the NEOS Server, a job number, password, and Web address are displayed to the screen, which provide information on accessing the job and viewing the intermediate output, for example,

```
--- Executing KESTREL
Job has been submitted to Kestrel
Kestrel/NEOS Job number    : 93478
Kestrel/NEOS Job password  : utiwtxTK
Check the following URL for progress report :
     https://neos-server.org/neos/cgi-bin/nph-neos-solver.cgi?admin=results&jobnumber=93478&pass=utiw
```

To look at the solver's output while it is running, point your browser at the URL given in the Kestrel output as shown above, and click on View Intermediate Results in the web page that appears. This will take you to another page that shows all of the output produced by the solver for your problem so far. To track the solver's progress, simply update this page periodically.

If the NEOS Server or the network becomes unavailable after the submission, a particular job can be retrieved by setting both the kestrel job and kestrel password in the options file.

```
kestrel_solver minos
kestrel_job 93478
kestrel_password utiwtxTK
```

Re-issuing the command gams trnsport with this options file will retrieve the results for the specified job number.

By specifying neos_username and neos_user_password in the option file, you can submit an authenticated job using your NEOS user account. Authenticated jobs will appear in your user account on the NEOS website.

```
neos_username username
neos_user_password password
```

5.23 KNITRO

5.23.1 Introduction

Artelys Knitro is a software package for finding local solutions of both continuous (i.e. smooth) optimization problems, with or without constraints, and discrete optimization problems with integer or binary variables. Even though Knitro has been designed for solving large-scale general problems, it is efficient for solving all of the following classes of optimization problems:

- unconstrained,
- bound constrained,
- equality constrained,
- systems of nonlinear equations,
- least squares problems,
- linear programming problems (LPs),
- quadratic programming problems (QPs),
- general (inequality) constrained problems,
- (convex) mixed integer nonlinear programs (MINLP) of moderate size.

The Knitro package provides the following features:

- Efficient and robust solution of small or large problems,
- Solvers for both continuous and discrete problems,
- Derivative-free, 1st derivative and 2nd derivative options,
- Both interior-point (barrier) and active-set optimizers,
- Both feasible and infeasible versions,
- Both iterative and direct approaches for computing steps,

The problems solved by Knitro have the form

$$\begin{aligned} \text{minimize} \quad & f(x) && (1a)\\ \text{subject to} \quad & c^L \leq c(x) \leq c^U && (1b)\\ & b^L \leq x \leq b^U, && (1c) \end{aligned}$$

where the variables x can be continuous, binary, or integer. This allows many forms of constraints, including bounds on the variables. Knitro requires that the functions $f(x)$ and $c(x)$ be smooth functions.

Knitro implements both state-of-the-art interior-point and active-set methods for solving nonlinear optimization problems. In the interior method (also known as a barrier method), the nonlinear programming problem is replaced by a series of barrier sub-problems controlled by a barrier parameter μ. The algorithm uses trust regions and a merit function to promote convergence. The algorithm performs one or more minimization steps on each barrier problem, then decreases the barrier parameter, and repeats the process until the original problem (1) has been solved to the desired accuracy.

Knitro provides two procedures for computing the steps within the interior point approach. In the version known as `Interior/CG` each step is

computed using a projected conjugate gradient iteration. This approach differs from most interior methods proposed in the literature in that it does not compute each step by solving a linear system involving the KKT (or primal-dual) matrix. Instead, it factors a projection matrix, and uses the conjugate gradient method, to approximately minimize a quadratic model of the barrier problem.

The second procedure for computing the steps, which we call `Interior/Direct`, always attempts to compute a new iterate by solving the primal-dual KKT matrix using direct linear algebra. In the case when this step cannot be guaranteed to be of good quality, or if negative curvature is detected, then the new iterate is computed by the `Interior/CG` procedure.

`Knitro` also implements an active-set sequential linear-quadratic programming (SLQP) algorithm which we call `Active`. This method is similar in nature to a sequential quadratic programming method but uses linear programming sub-problems to estimate the active-set at each iteration. This active-set code may be preferable when a good initial point can be provided, for example, when solving a sequence of related problems.

For problems with discrete variables, Knitro provides two variants of the branch and bound algorithm. The first is a standard implementation, while the second is specialized for convex, mixed-integer nonlinear problems.

We encourage the user to try all algorithmic options to determine which one is more suitable for the application at hand. For guidance on choosing the best algorithm see section Algorithm Options .

For a detailed description of the algorithm implemented in `Interior/CG` see [30] and for the global convergence theory see [31] . The method implemented in `Interior/Direct` is described in [198] . The `Active` algorithm is described in [33] and the global convergence theory for this algorithm is in [34] . An important component of Knitro is the HSL routine `MA27` [92] which is used to solve the linear systems arising at every iteration of the algorithm. In addition, the `Active Set` algorithm in Knitro may make use of the COIN-OR `Clp` linear programming solver module. The version used in `Knitro` may be downloaded from `http://www.artelys.com/tools/clp/`.

5.23.2 Usage

Basic details of solver usage, including how to choose Knitro as the solver and how to use a solver-specific option file, are part of Chapter Solver Usage.

As an NLP solver, Knitro can also be used to solve linear programs (LP), and both convex and nonconvex quadratic programs (QCP).

5.23.3 GAMS Options

The following GAMS options are used by the GAMS/Knitro link:

- Option ResLim = x;

  Sets the time limit in seconds. If this limit is exceeded the solver will terminate and pass on the current solution to GAMS. See also reslim in section GAMS options .

- Option SysOut = On;

  This option sends additional Knitro messages to the GAMS listing file. It is useful in case of a solver failure or to get algorithmic details. See also sysout in section GAMS options

- *ModelName*.optCA = x;

  Absolute gap stop criterion for a discrete problem. The Knitro option mip_opt_gap_abs takes its default from this value. See also optca in section GAMS options.

- *ModelName*.optCR = x;

  Relative gap stop criterion for a discrete problem. The Knitro option mip_opt_gap_rel takes its default from this value. See also optcr in section GAMS options.

5.23.4 Summary of Knitro Options

The Knitro options file `knitro.opt` allows the user to easily set options controlling `Knitro`'s behavior. Options are set by specifying a keyword and a corresponding value on a line in the `knitro.opt` file. Lines that begin with a `#` character are treated as comments and blank lines are ignored. For example, to set the maximum allowable number of iterations to 500, one could use the following options file:

```
maxit 500
```

5.23.4.1 General options

Option	Description	Default
algorithm	Indicates which algorithm to use to solve the problem	0
blasoption	Specifies the BLAS/LAPACK function library to use for basic vector and matrix computations	1 or 4 (platform dependent)
blasoptionlib	Specifies a dynamic library name that contains object code for BLAS/LAPACK functions	
bndrange	Specifies max limits on the magnitude of constraint and variable bounds	1e+20
cg_maxit	Determines the maximum allowable number of inner conjugate gradient (CG) iterations per Knitro minor iteration	-1
cg_pmem	Specifies the amount of nonzero elements per column of the Hessian of the Lagrangian which are retained when computing the incomplete Cholesky preconditioner	10
cg_precond	Specifies whether an incomplete Cholesky preconditioner is applied during CG iterations in barrier algorithms	0
cg_stoptol	Specifies the relative stopping tolerance used for the conjugate gradient (CG) subproblem solves	0.01
convex	Declare the problem as convex by setting to 1 or non-convex by setting to 0	-1
cpuplatform	This option can be used to specify the target instruction set architecture for the machine on which Knitro is running	-1
datacheck	Specifies whether to perform more extensive data checks to look for errors in the problem input to Knitro (in particular, this option looks for errors in the sparse Jacobian and/or sparse Hessian structure)	0
delta	Specifies the initial trust region radius scaling factor used to determine the initial trust region size	1
honorbnds	Indicates whether or not to enforce satisfaction of simple variable bounds throughout the optimization	-1
initpenalty	Specifies the initial penalty parameter used in the Knitro merit functions	10
initvalues	Enable use of initial guess for levels and marginals (subsequent solves)	1

Option	Description	Default
initvalues0	Enable use of initial guess for levels and marginals (first solve)	2
linesearch	Indicates which linesearch strategy to use for the Interior/Direct or SQP algorithm to search for a new acceptable iterate	0
linesearch_maxtrials	Indicates the maximum allowable number of trial points during the linesearch of the Interior/Direct or SQP algorithm before treating the linesearch step as a failure and generating a new step	3
linsolver	Indicates which linear solver to use to solve linear systems arising in Knitro algorithms	0
linsolver_maxitref	Indicates the maximum allowable number of iterative refinement steps applied when a linear system is solved inside Knitro	2
linsolver_nodeamalg	Controls the node amalgamation setting for the MA57, MA86 and MA97 linear solvers	0
linsolver_ooc	Indicates whether to use Intel MKL PARDISO out-of-core solve of linear systems when linsolver = mklpardiso	0
linsolver_ordering	Sets the ordering method used for the linear system solver	-1
linsolver_pivottol	Specifies the initial pivot threshold used in factorization routines	1e-08
linsolver_scaling	Enables scaling for the linear system solver	0
names	Enable to pass variable and equation names to Knitro	0
ncvx_qcqp_init	Specifies the initialization strategy used for non-convex QPs and QCQPs	-1
newpoint	Specifies additional action to take after every iteration in a solve of a continuous problem, or after every new incumbent of the NLPBB algorithm	0
objrange	Specifies the extreme limits of the objective function for purposes of determining unboundedness	1e+20
option_file	additional option file name - read only by KNITRO solver lib	
output_time	print output on where time is used	0
qextractalg	quadratic extraction algorithm in GAMS interface	0
restarts	Specifies whether or not to enable automatic restarts in Knitro	-1
restarts_maxit	When restarts are enabled, this option can be used to specify a maximum number of iterations before enforcing a restart	0
scale	Specifies whether to perform problem scaling of the objective function, constraint functions, or possibly variables	1
scale_vars	Specifies the strategy for scaling variables	0
soc	Specifies whether or not to try second order corrections (SOC)	1

Option	Description	Default
strat_warm_start	Specifies whether or not to invoke a warm-start strategy	0

5.23.4.2 Derivative options

Option	Description	Default
bfgs_scaling	Specify the initial scaling to use for the BFGS or L-BFGS Hessian approximation	0
findiff_estnoise	This option can be used to enable an estimate of the noise in the model when using finite-difference gradients	0
findiff_relstepsize	Specifies the relative stepsize used for finite-difference gradients during the optimization	0
findiff_terminate	This option specifies the termination criteria when using finite-difference gradients	1
gradopt	Specifies how to compute the gradients of the objective and constraint functions	1
hessopt	Specifies how to compute the (approximate) Hessian of the Lagrangian	0
lmsize	Specifies the number of limited memory pairs stored when approximating the Hessian using the limited-memory quasi-Newton BFGS option	10

5.23.4.3 Termination options

Option	Description	Default
feastol	Specifies the final relative stopping tolerance for the feasibility error	1e-06
feastolabs	Specifies the final absolute stopping tolerance for the feasibility error	0.001
fstopval	Used to implement a custom stopping condition based on the objective function value	maxdouble
ftol	The optimization process will terminate if the relative change in the objective function is less than ftol for ftol_iters consecutive feasible iterations	1e-15
ftol_iters	The optimization process will terminate if the relative change in the objective function is less than ftol for ftol_iters consecutive feasible iterations	5
infeastol	Specifies the (relative) tolerance used for declaring infeasibility of a model	1e-08
infeastol_iters	The optimization process will terminate if the relative change in the feasibility error is less than infeastol for infeastol_iters consecutive infeasible iterations	50
maxfevals	Specifies the maximum number of function evaluations before termination	-1
maxit	Specifies the maximum number of iterations before termination	GAMS iterlim
maxtime_cpu	Specifies, in seconds, the maximum allowable CPU time before termination	100000000
maxtime_real	Specifies, in seconds, the maximum allowable real time before termination	GAMS reslim
opttol	Specifies the final relative stopping tolerance for the KKT (optimality) error	1e-06

Option	Description	Default
opttolabs	Specifies the final absolute stopping tolerance for the KKT (optimality) error	0.001
xtol	The optimization process will terminate if the relative change in all components of the solution point estimate is less than xtol for xtol_iters	1e-12
xtol_iters	The optimization process will terminate if the relative change in the solution estimate is less than xtol for xtol_iters consecutive iterations	0

5.23.4.4 Presolve options

Option	Description	Default
presolve	Determine whether or not to use the Knitro presolver to try to simplify the model by removing variables or constraints	1
presolveop_redundant	Determine whether or not to enable the Knitro presolve operation to detect and remove redundant constraints	1
presolveop_substitution	Determine whether or not to enable the Knitro presolve operation to substitute out variables when possible	-1
presolveop_substitution_tol	Tolerance for applying a substitution	0.01
presolveop_tighten	Determine whether or not to enable the Knitro presolve operation to tighten variable bounds or coefficients	-1
presolve_initpt	Control whether the Knitro presolver can shift a user-supplied initial point	-1
presolve_level	Set the level of presolve operations to enable through the Knitro presolver	-1
presolve_passes	Set a maximum limit on the number of passes through the Knitro presolve operations	10
presolve_tol	Determines the tolerance used by the Knitro presolver to remove variables and constraints from the model	1e-06

5.23.4.5 Barrier options

Option	Description	Default
bar_conic_enable	Enable special treatments for conic constraints when using the Interior/Direct algorithm (has no affect when using the Interior/CG algorithm)	-1
bar_directinterval	Controls the maximum number of consecutive conjugate gradient (CG) steps before Knitro will try to enforce that a step is taken using direct linear algebra	-1
bar_feasible	Specifies whether special emphasis is placed on getting and staying feasible in the interior-point algorithms	0
bar_feasmodetol	Specifies the tolerance in equation that determines whether Knitro will force subsequent iterates to remain feasible	0.0001
bar_globalize	Specifies the globalization strategy used in the interior-point algorithms	2
bar_initmu	Specifies the initial value for the barrier parameter μ used with the barrier algorithms	-1
bar_initpi_mpec	Specifies the initial value for the MPEC penalty parameter π used when solving problems with complementarity constraints using the barrier algorithms	0

Option	Description	Default
bar_initpt	Indicates initial point strategy for x, slacks and multipliers when using a barrier algorithm	0
bar_linsys	Indicates which linear system form is used inside the Interior/Direct algorithm for computing primal-dual steps	-1
bar_linsys_storage	Indicates how to store in memory the linear systems used inside the Interior/Direct algorithm for computing primal-dual steps	-1
bar_maxcorrectors	Specifies the maximum number of corrector steps allowed for primal-dual steps	-1
bar_maxcrossit	Specifies the maximum number of crossover iterations before termination	0
bar_maxmu	Specifies the maximum allowable value for the barrier parameter μ used with the barrier algorithms	1e+16
bar_maxrefactor	Indicates the maximum number of refactorizations of the KKT system per iteration of the Interior/Direct algorithm before reverting to a CG step	-1
bar_mpec_heuristic	Specifies whether or not to use a heuristic approach when solving MPEC models with the barrier algorithm	0
bar_murule	Indicates which strategy to use for modifying the barrier parameter mu in the barrier algorithms	0
bar_penaltycons	Indicates whether a penalty approach is applied to the constraints	-1
bar_penaltyrule	Indicates which penalty parameter strategy to use for determining whether or not to accept a trial iterate	0
bar_refinement	Specifies whether to try to refine the barrier solution for better precision	0
bar_relaxcons	Indicates whether a relaxation approach is applied to the constraints	2
bar_slackboundpush	Specifies the amount by which the barrier slack variables are initially pushed inside the bounds	-1
bar_switchobj	Indicates which objective function to use when the barrier algorithms switch to a pure feasibility phase	1
bar_switchrule	Indicates whether or not the barrier algorithms will allow switching from an optimality phase to a pure feasibility phase	-1
bar_watchdog	Specifies whether to enable watchdog heuristic for barrier algorithms	0

5.23.4.6 Active-set options

Option	Description	Default
act_lpfeastol	Specifies the feasibility tolerance used for linear programming subproblems solved when using the Active Set or SQP algorithms	1e-08
act_lppenalty	Indicates whether to use a penalty formulation for linear programming subproblems in the Knitro Active Set or SQP algorithms	1
act_lppresolve	Indicates whether to apply a presolve for linear programming subproblems in the Knitro Active Set or SQP algorithms	0
act_parametric	Indicates whether to use a parametric approach when solving linear programming (LP) subproblems when using the Knitro Active Set or SQP algorithms	1
act_qpalg	Indicates which algorithm to use to solve quadratic programming (QP) subproblems when using the Knitro Active Set or SQP algorithms	0

Option	Description	Default
act_qppenalty	Indicates whether to use a penalty formulation for quadratic programming subproblems in the Knitro SQP algorithm	-1
mip_cut_flowcover	Specifies rules for adding flow cover cuts	-1
mip_cut_probing	Specifies rules for adding probing cuts	-1
mip_heuristic_localsearch	Specifies whether or not to enable the MIP local search heuristic	-1

5.23.4.7 MIP options

Option	Description	Default
mip_branchrule	Specifies which branching rule to use for MIP branch and bound procedure	0
mip_clique	Specifies rules for adding clique cuts	-1
mip_cutfactor	This value specifies a limit on the number of cuts added to a node subproblem	1
mip_cutoff	This value specifies the objective cutoff value for MIP	maxdouble
mip_cutting_plane	Specifies when to apply the cutting plane procedure	1
mip_gomory	Specifies rules for adding Gomory mixed-integer cuts	-1
mip_gub_branch	Specifies whether or not to branch on generalized upper bounds (GUBs)	0
mip_heuristic_diving	Specifies whether or not to enable the MIP diving heuristic	-1
mip_heuristic_feaspump	Specifies whether or not to enable the MIP feasibility pump heuristic	-1
mip_heuristic_lns	Specifies whether or not to enable the MIP large neighborhood search (LNS) heuristics	-1
mip_heuristic_maxit	Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled	100
mip_heuristic_misqp	Specifies whether or not to enable the MIP MISQP heuristic	-1
mip_heuristic_mpec	Specifies whether or not to enable the MIP MPEC heuristic	-1
mip_heuristic_strategy	Specifies the level of effort applied for the MIP heuristic search used to try to find an initial integer feasible point	-1
mip_heuristic_terminate	Specifies the condition for terminating the MIP heuristic	1
mip_implications	Specifies whether or not to add constraints to the MIP derived from logical implications	1
mip_integer_tol	This value specifies the threshold for deciding whether or not a variable is determined to be an integer	1e-08
mip_intvar_strategy	Specifies how to handle integer variables	0
mip_knapsack	Specifies rules for adding MIP knapsack cuts	-1
mip_liftproject	Specifies rules for adding lift and project cuts	-1
mip_lpalg	Specifies which algorithm to use for any linear programming (LP) subproblem solves that may occur in the MIP branch-and-bound procedure	0

Option	Description	Default
mip_maxnodes	Specifies the maximum number of nodes explored (0 means no limit)	GAMS nodlim
mip_maxsolves	Specifies the maximum number of subproblem solves allowed (0 means no limit)	0
mip_maxtime_cpu	Specifies the maximum allowable CPU time in seconds for the complete MIP solution	100000000
mip_maxtime_real	Specifies the maximum allowable real time in seconds for the complete MIP solution	GAMS reslim
mip_method	Specifies which MIP method to use	0
mip_mir	Specifies rules for adding mixed-integer rounding cuts	-1
mip_multistart	Use to enable MIP multi-start at the branch-and-bound level	0
mip_nodealg	Specifies which algorithm to use for standard node subproblem solves in MIP (same options as algorithm user option)	0
mip_numthreads	Specify the number of threads to use for MIP branch-and-bound (when mip_method = 1)	0
mip_opt_gap_abs	The absolute optimality gap stop tolerance for MIP	GAMS optca
mip_opt_gap_rel	The relative optimality gap stop tolerance for MIP	GAMS optcr
mip_outinterval	Specifies node printing interval for mip_outlevel when mip_outlevel > 0	0
mip_outlevel	Specifies how much MIP information to print	2
mip_pseudoinit	Specifies the method used to initialize pseudo-costs corresponding to variables that have not yet been branched on in the MIP method	0
mip_relaxable	Specifies whether integer variables are relaxable	1
mip_restart	Specifies whether to enable the MIP restart procedure	1
mip_rootalg	Specifies which algorithm to use for the root node solve in MIP (same options as algorithm user option)	0
mip_rounding	Specifies the MIP rounding rule to apply	-1
mip_selectdir	Specifies the MIP node selection direction rule (for tiebreakers) for choosing the next node in the branch-and-bound tree	0
mip_selectrule	Specifies the MIP select rule for choosing the next node in the branch-and-bound tree	0
mip_strong_candlim	Specifies the maximum number of candidates to explore for MIP strong branching	128
mip_strong_level	Specifies the maximum number of tree levels on which to perform MIP strong branching	10
mip_strong_maxit	Specifies the maximum number of iterations to allow for MIP strong branching solves	1000
mip_terminate	Specifies conditions for terminating the MIP algorithm	0
mip_zerohalf	Specifies rules for adding zero-half cuts	-1

5.23.4.8 Multi-algorithm options

Option	Description	Default
ma_maxtime_cpu	Specifies, in seconds, the maximum allowable CPU time before termination for the multi-algorithm ("MA") procedure (algorithm=5)	100000000
ma_maxtime_real	Specifies, in seconds, the maximum allowable real time before termination for the multi-algorithm ("MA") procedure (algorithm=5)	100000000
ma_outsub	Enable writing algorithm output to files for the multi-algorithm (algorithm=5) procedure	0
ma_terminate	Define the termination condition for the multi-algorithm (algorithm=5) procedure	1

5.23.4.9 Multi-start options

Option	Description	Default
ms_deterministic	Indicates whether Knitro multi-start procedure will be deterministic (when ms_terminate = 0)	1
ms_enable	Indicates whether Knitro will solve from multiple start points to find a better local minimum	0
ms_initpt_cluster	The strategy for clustering initial points in multi-start	0
ms_maxbndrange	Specifies the maximum range that an unbounded variable can take when determining new start points	1000
ms_maxsolves	Specifies how many start points to try in multi-start	0
ms_maxtime_cpu	Specifies, in seconds, the maximum allowable CPU time before termination	100000000
ms_maxtime_real	Specifies, in seconds, the maximum allowable real time before termination	100000000
ms_numthreads	Specify the number of threads to use for multi-start (when ms_enable = 1)	0
ms_num_to_save	Specifies the number of distinct feasible points to save in a file named knitro_mspoints	0
ms_outsub	Enable writing algorithm output to files for the parallel multi-start procedure	0
ms_savetol	Specifies the tolerance for deciding if two feasible points are distinct	1e-06
ms_seed	Seed value used to generate random initial points in multi-start; should be a non-negative integer	0
ms_startptrange	Specifies the maximum range that each variable can take when determining new start points	1e+20
ms_terminate	Specifies the condition for terminating multi-start	0

5.23.4.10 Parallelism options

Option	Description	Default
blas_numthreads	Specify the number of threads to use for BLAS operations when blasoption = 1	0
conic_numthreads	Specify the number of threads to use for operations in the conic algorithm (when bar_conic_enable = 1)	0
linsolver_numthreads	Specify the number of threads to use for linear system solve operations when linsolver = 6	0
threads	default thread count	GAMS threads

5.23.4.11 Output options

Option	Description	Default
outappend	Specifies whether output should be started in a new file, or appended to existing files	0
outdir	Specifies a single directory as the location to write all output files	
outlev	Controls the level of output produced by Knitro	2
outmode	Specifies where to direct the output from Knitro	0
outname	Use to specify a custom filename when output is written to a file using outmode	
out_csvinfo	Controls whether or not to generates a file knitro_solve	0
out_csvname	Use to specify a custom csv filename when using out_csvinfo	
out_hints	Specifies whether to print diagnostic hints (e.g. about user option settings) after solving	1

5.23.4.12 Tuner options

Option	Description	Default
tuner	Indicates whether to invoke the Knitro-Tuner	0
tuner_maxtime_cpu	Specifies, in seconds, the maximum allowable CPU time before terminating the Knitro-Tuner	100000000
tuner_maxtime_real	Specifies, in seconds, the maximum allowable real time before terminating the Knitro-Tuner	100000000
tuner_optionsfile	Can be used to specify the location of a Tuner options file	
tuner_outsub	Enable writing additional Tuner subproblem solve output to files for the Knitro-Tuner procedure (tuner=1)	0
tuner_terminate	Define the termination condition for the Knitro-Tuner procedure (tuner=1)	0

5.23.5 Detailed Descriptions of Knitro Options

act_lpfeastol *(real)*: Specifies the feasibility tolerance used for linear programming subproblems solved when using the Active Set or SQP algorithms ↩

Range: [0, ∞]

Default: 1e-08

act_lppenalty *(integer)*: Indicates whether to use a penalty formulation for linear programming subproblems in the Knitro Active Set or SQP algorithms ↩

Default: 1

value	meaning
1	(all) penalize all constraints.
2	(nonlinear) penalize only nonlinear constraints.
3	(dynamic) dynamically choose which constraints to penalize.

act_lppresolve *(boolean)*: Indicates whether to apply a presolve for linear programming subproblems in the Knitro Active Set or SQP algorithms ↩

Default: `0`

value	meaning
`0`	(off) presolve turned off for LP subproblems.
`1`	(on) presolve turned on for LP subproblems.

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

Indicates whether to use a parametric approach when solving linear programming (LP) subproblems when using the Knitro Active Set or SQP algorithms. A parametric approach will solve a sequence of closely related LPs instead of one LP. It may increase the cost of an active-set iteration, but perhaps lead to convergence in fewer iterations.

Default: `1`

value	meaning
`0`	(no) do not use a parametric solve (i.e. solve a single LP).
`1`	(maybe) use a parametric solve sometimes.
`2`	(yes) always try a parametric solve.

act_qpalg *(integer)*: Indicates which algorithm to use to solve quadratic programming (QP) subproblems when using the Knitro Active Set or SQP algorithms ↩

Indicates which algorithm to use to solve quadratic programming (QP) subproblems when using the Knitro Active Set or SQP algorithms. This option has no effect on the Interior/Direct and Interior/CG algorithms.

Default: `0`

value	meaning
`0`	(auto) let Knitro automatically choose an algorithm, based on the problem characteristics.
`1`	(direct) use the Interior/Direct algorithm.
`2`	(cg) use the Interior/CG algorithm.
`3`	(active) use the Active Set algorithm.

act_qppenalty *(integer)*: Indicates whether to use a penalty formulation for quadratic programming subproblems in the Knitro SQP algorithm ↩

Default: `-1`

value	meaning
`-1`	(auto) let Knitro automatically decide.
`0`	(none) do not penalize constraints in QP subproblems.
`1`	(all) penalize all constraints in QP subproblems.

algorithm *(integer)*: Indicates which algorithm to use to solve the problem ↩

Default: 0

value	meaning
0	(auto) let Knitro automatically choose an algorithm, based on the problem characteristics.
1	(direct) use the Interior/Direct algorithm.
2	(cg) use the Interior/CG algorithm.
3	(active) use the Active Set algorithm.
4	(sqp) use the SQP algorithm.
5	(multi) run all algorithms, perhaps in parallel (see Algorithms).

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

Default: -1

value	meaning
-1	(auto) Let Knitro automatically choose the strategy.
0	(none) Do not apply any special treatment for conic constraints.
1	(soc) Apply special treatments for any Second Order Cone (SOC) constraints identified in the model.

bar_directinterval *(integer)*: Controls the maximum number of consecutive conjugate gradient (CG) steps before Knitro will try to enforce that a step is taken using direct linear algebra ↩

Controls the maximum number of consecutive conjugate gradient (CG) steps before Knitro will try to enforce that a step is taken using direct linear algebra. This option is only valid for the Interior/Direct algorithm and may be useful on problems where Knitro appears to be taking lots of conjugate gradient steps. Setting bar_directinterval to 0 will try to enforce that only direct steps are taken which may produce better results on some problems.

Range: {-1, ..., ∞}

Default: -1

bar_feasible *(integer)*: Specifies whether special emphasis is placed on getting and staying feasible in the interior-point algorithms ↩

Specifies whether special emphasis is placed on getting and staying feasible in the interior-point algorithms. Note This option can only be used with the Interior/Direct and Interior/CG algorithms. If bar_feasible = stay or bar_feasible = get_stay, this will activate the feasible version of Knitro. The feasible version of Knitro will force iterates to strictly satisfy inequalities, but does not require satisfaction of equality constraints at intermediate iterates. This option and the honorbnds option may be useful in applications where functions are undefined outside the region defined by inequalities. The initial point must satisfy inequalities to a sufficient degree; if not, Knitro may generate infeasible iterates and does not switch to the feasible version until a sufficiently feasible point is found. Sufficient satisfaction occurs at a point x if it is true for all inequalities that $cl + tol \leq c(x) \leq cu - tol$ The constant tol is determined by the option bar_feasmodetol. If bar_feasible = get or bar_feasible = get_stay, Knitro will place special emphasis on first trying to get feasible before trying to optimize.

Default: 0

value	meaning
0	(no) No special emphasis on feasibility.
1	(stay) Iterates must satisfy inequality constraints once they become sufficiently feasible.
2	(get) Special emphasis is placed on getting feasible before trying to optimize.
3	(get_stay) Implement both options 1 and 2 above.

bar_feasmodetol *(real)*: Specifies the tolerance in equation that determines whether Knitro will force subsequent iterates to remain feasible ↩

Specifies the tolerance in equation that determines whether Knitro will force subsequent iterates to remain feasible. The tolerance applies to all inequality constraints in the problem. This option only has an effect if option bar_feasible = stay or bar_feasible = get_stay.

Range: [0, ∞]

Default: 0.0001

bar_globalize *(integer)*: Specifies the globalization strategy used in the interior-point algorithms ↩

Default: 2

value	meaning
0	(none) No globalization strategy is applied.
1	(kkt) Apply a globalization strategy based on decreasing the KKT error.
2	(filter) Apply a globalization strategy using a filter based on the objective and constraint violation.

bar_initmu *(real)*: Specifies the initial value for the barrier parameter μ used with the barrier algorithms ↩

Specifies the initial value for the barrier parameter μ used with the barrier algorithms. This option has no effect on the Active Set algorithm.

Range: [-∞, ∞]

Default: -1

bar_initpi_mpec *(real)*: Specifies the initial value for the MPEC penalty parameter π used when solving problems with complementarity constraints using the barrier algorithms ↩

Specifies the initial value for the MPEC penalty parameter π used when solving problems with complementarity constraints using the barrier algorithms. If this value is non-positive, then Knitro uses an internal formula to initialize the MPEC penalty parameter.

Range: [0, ∞]

Default: 0

bar_initpt *(integer)*: Indicates initial point strategy for x, slacks and multipliers when using a barrier algorithm ↩

Indicates initial point strategy for x, slacks and multipliers when using a barrier algorithm. Note, this option only alters the initial x values if the user does not specify an initial x. This option has no effect on the Active Set algorithm.

Default: 0

value	meaning
0	(auto) Let Knitro automatically choose the strategy.
1	(convex) Initialization designed for convex models.
2	(nearbnd) Initialization strategy that stays closer to the bounds.
3	(central) Initialization strategy that is more central on double-bounded variables.

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

Indicates which linear system form is used inside the Interior/Direct algorithm for computing primal-dual steps. Eliminating more elements results in a smaller dimensional linear system (but also one that is, perhaps, less numerically stable). The bounds option may be preferable for very large problems with many bounded variables. The ineq option may generate significant speedups on models where the number of variables is small, but the number of inequality constraints is large.

Default: -1

value	meaning
-1	(auto) Let Knitro automatically choose the linear system form.
0	(full) Use the full linear system.
1	(slacks) Eliminate the slack variables.
2	(bounds) Eliminate the slack variables and bounds.
3	(ineqs) Eliminate the slack variables, bounds, and some inequalities.

bar_linsys_storage *(integer)*: Indicates how to store in memory the linear systems used inside the Interior/Direct algorithm for computing primal-dual steps ↩

Indicates how to store in memory the linear systems used inside the Interior/Direct algorithm for computing primal-dual steps. The lowmem option uses one storage location for multiple linear systems. As a result it may use much less memory, but also may be less efficient when the Interior/Direct algorithm takes a lot of CG steps. The normal option uses separate storage for different linear systems.

Default: -1

value	meaning
-1	(auto) Let Knitro automatically choose the linear system storage approach.
1	(lowmem) Use common storage for multiple linear systems.
2	(normal) Use separate storage for different linear systems.

bar_maxcorrectors *(integer)*: Specifies the maximum number of corrector steps allowed for primal-dual steps ↩

Specifies the maximum number of corrector steps allowed for primal-dual steps. If the value is positive and the algorithm used is Interior/Direct, then Knitro may add at most bar_maxcorrectors corrector steps to the primal-dual step to try to stay closer to the central path. This may speedup convergence on some models (although it may make the cost per iteration a little more expensive). If the value is negative, Knitro automatically determines the maximum number of corrector steps to apply.

Range: {-∞, ..., ∞}

Default: -1

bar_maxcrossit *(integer)*: Specifies the maximum number of crossover iterations before termination ↩

Specifies the maximum number of crossover iterations before termination. If the value is positive and the algorithm in operation is Interior/Direct or Interior/CG, then Knitro will crossover to the Active Set algorithm near the solution. The Active Set algorithm will then perform at most bar_maxcrossit iterations to get a more exact solution. If the value is 0, no Active Set crossover occurs and the interior-point solution is the final result. If Active Set crossover is unable to improve the approximate interior-point solution, then Knitro will restore the interior-point solution. In some cases (especially on large-scale problems or difficult degenerate problems) the cost of the crossover procedure may be significant – for this reason, crossover is disabled by default. Enabling crossover generally provides a more accurate solution than Interior/Direct or Interior/CG.

Range: {0, ..., ∞}

Default: 0

bar_maxmu *(real)*: Specifies the maximum allowable value for the barrier parameter μ used with the barrier algorithms ↩

Range: [2.08157871384174e-07, 1e+16]

Default: 1e+16

bar_maxrefactor *(integer)*: Indicates the maximum number of refactorizations of the KKT system per iteration of the Interior/Direct algorithm before reverting to a CG step ↩

Indicates the maximum number of refactorizations of the KKT system per iteration of the Interior/Direct algorithm before reverting to a CG step. If this value is set to -1, it will use a dynamic strategy. These refactorizations are performed if negative curvature is detected in the model. Rather than reverting to a CG step, the Hessian matrix is modified in an attempt to make the subproblem convex and then the KKT system is refactorized. Increasing this value will make the Interior/Direct algorithm less likely to take CG steps. If the Interior/Direct algorithm is taking a large number of CG steps (as indicated by a positive value for "CGits" in the output), this may improve performance. This option has no effect on the Active Set algorithm.

Range: {-1, ..., ∞}

Default: -1

bar_mpec_heuristic *(boolean)*: Specifies whether or not to use a heuristic approach when solving MPEC models with the barrier algorithm ↩

Specifies whether or not to use a heuristic approach when solving MPEC models with the barrier algorithm. In some cases enabling this heuristic can speedup the convergence to the solution and provide a more precise solution on MPEC models (i.e., models with complementarity constraints).

Default: 0

value	meaning
0	(no) Do not enable the heuristic for MPEC models.
1	(yes) Enable the heuristic for MPEC models.

bar_murule *(integer)*: Indicates which strategy to use for modifying the barrier parameter *mu* in the barrier algorithms ↩

Indicates which strategy to use for modifying the barrier parameter *mu* in the barrier algorithms. Not all strategies are available for both barrier algorithms, as described below. This option has no effect on the Active Set algorithm.

Default: `0`

value	meaning
`0`	(auto) Let Knitro automatically choose the strategy.
`1`	(monotone) Monotonically decrease the barrier parameter. Available for both barrier algorithms.
`2`	(adaptive) Use an adaptive rule based on the complementarity gap to determine the value of the barrier parameter. Available for both barrier algorithms.
`3`	(probing) Use a probing (affine-scaling) step to dynamically determine the barrier parameter. Available only for the Interior/Direct algorithm.
`4`	(dampmpc) Use a Mehrotra predictor-corrector type rule to determine the barrier parameter, with safeguards on the corrector step. Available only for the Interior/Direct algorithm.
`5`	(fullmpc) Use a Mehrotra predictor-corrector type rule to determine the barrier parameter, without safeguards on the corrector step. Available only for the Interior/Direct algorithm.
`6`	(quality) Minimize a quality function at each iteration to determine the barrier parameter. Available only for the Interior/Direct algorithm.

bar_penaltycons *(integer)*: Indicates whether a penalty approach is applied to the constraints ↩

Indicates whether a penalty approach is applied to the constraints. Using a penalty approach may be helpful when the problem has degenerate or difficult constraints. It may also help to more quickly identify infeasible problems, or achieve feasibility in problems with difficult constraints. This option has no effect on the Active Set algorithm.

Default: `-1`

value	meaning
`-1`	(auto) Let Knitro automatically choose the strategy.
`0`	(none) No constraints are penalized.
`2`	(all) A penalty approach is applied to all general constraints.
`3`	(equalities) Apply a penalty approach to equality constraints only.

bar_penaltyrule *(integer)*: Indicates which penalty parameter strategy to use for determining whether or not to accept a trial iterate ↩

Indicates which penalty parameter strategy to use for determining whether or not to accept a trial iterate. This option has no effect on the Active Set algorithm.

Default: `0`

value	meaning
`0`	(auto) Let Knitro automatically choose the strategy.

value	meaning
1	(single) Use a single penalty parameter in the merit function to weight feasibility versus optimality.
2	(flex) Use a more tolerant and flexible step acceptance procedure based on a range of penalty parameter values.

bar_refinement *(boolean)*: Specifies whether to try to refine the barrier solution for better precision ↩

Specifies whether to try to refine the barrier solution for better precision. If enabled, once the optimality conditions are satisfied, Knitro will apply an additional refinement/postsolve phase to try to obtain more precision in the barrier solution. The effect is similar to the effect of enabling bar_maxcrossit, but it is usually much more efficient since it does not involve switching to the Active Set algorithm.

Default: 0

bar_relaxcons *(integer)*: Indicates whether a relaxation approach is applied to the constraints ↩

Indicates whether a relaxation approach is applied to the constraints. Using a relaxation approach may be helpful when the problem has degenerate or difficult constraints. This option has no effect on the Active Set algorithm.

Default: 2

value	meaning
0	(none) No constraints are relaxed.
1	(eqs) A relaxation approach is applied to general equality constraints.
2	(ineqs) A relaxation approach is applied to general inequality constraints.
3	(all) A relaxation approach is applied to all general constraints.

bar_slackboundpush *(real)*: Specifies the amount by which the barrier slack variables are initially pushed inside the bounds ↩

Specifies the amount by which the barrier slack variables are initially pushed inside the bounds. A smaller value may be preferable when warm-starting from a point close to the solution.

Range: [-∞, ∞]

Default: -1

bar_switchobj *(integer)*: Indicates which objective function to use when the barrier algorithms switch to a pure feasibility phase ↩

Default: 1

value	meaning
0	(none) No (or zero) objective.
1	(scalarprox) Proximal point objective with scalar weighting.
2	(diagprox) Proximal point objective with diagonal weighting.

bar_switchrule *(integer)*: Indicates whether or not the barrier algorithms will allow switching from an

optimality phase to a pure feasibility phase ↩

Indicates whether or not the barrier algorithms will allow switching from an optimality phase to a pure feasibility phase. This option has no effect on the Active Set algorithm.

Default: `-1`

value	meaning
`-1`	(auto) Let Knitro determine the switching procedure.
`0`	(never) Never switch to feasibility phase.
`2`	(moderate) Allow switches to feasibility phase.
`3`	(aggressive) Use a more aggressive switching rule.

bar_watchdog *(boolean)*: Specifies whether to enable watchdog heuristic for barrier algorithms ↩

Specifies whether to enable watchdog heuristic for barrier algorithms. In general, enabling the watchdog heuristic makes the barrier algorithms more likely to accept trial points. Specifically, the watchdog heuristic may occasionally accept trial points that increase the merit function, provided that subsequent iterates decrease the merit function.

Default: `0`

bfgs_scaling *(integer)*: Specify the initial scaling to use for the BFGS or L-BFGS Hessian approximation ↩

Default: `0`

value	meaning
`0`	(dynamic) Dynamically choose which scaling to use.
`1`	(invhess) The scaling approximates the scale of the inverse Hessian.
`2`	(hess) The scaling approximates the scale of the Hessian.

blasoption *(integer)*: Specifies the BLAS/LAPACK function library to use for basic vector and matrix computations ↩

Default: `1 or 4 (platform dependent)`

value	meaning
`-1`	(auto) Let Knitro automatically choose which BLAS to use.
`0`	(knitro) Use Knitro built-in functions.
`1`	(intel) Use Intel Math Kernel Library (MKL) functions on available platforms.
`2`	(dynamic) Use the dynamic library specified with option blasoptionlib.
`3`	(blis) Use BLIS functions on available platforms (currently not available on Windows OS).
`4`	(apple) Use Apple Accelerate (only available on Mac with M1 processor).

blasoptionlib *(string)*: Specifies a dynamic library name that contains object code for BLAS/LAPACK functions ↩

Specifies a dynamic library name that contains object code for BLAS/LAPACK functions. The library must implement all the functions declared in the file include/blas_lapack.h.

blas_numthreads *(integer)*: Specify the number of threads to use for BLAS operations when blasoption = 1 ↩

Range: {0, ..., ∞}

Default: 0

bndrange *(real)*: Specifies max limits on the magnitude of constraint and variable bounds ↩

Specifies max limits on the magnitude of constraint and variable bounds. Any constraint or variable bounds whose magnitude is greater than or equal to bndrange will be treated as infinite by Knitro. Using very large, finite bounds is discouraged (and is generally an indication of a poorly scaled model).

Range: [0, ∞]

Default: 1e+20

cg_maxit *(integer)*: Determines the maximum allowable number of inner conjugate gradient (CG) iterations per Knitro minor iteration ↩

Default: -1

value	meaning
-1	Let Knitro automatically determine a value.
0	Knitro will set a maximum value based on the problem size.
n	At most n>0 CG iterations may be performed during one minor iteration of Knitro.

cg_pmem *(integer)*: Specifies the amount of nonzero elements per column of the Hessian of the Lagrangian which are retained when computing the incomplete Cholesky preconditioner ↩

Default: 10

value	meaning
n	At most n>0 nonzero elements per column.

cg_precond *(boolean)*: Specifies whether an incomplete Cholesky preconditioner is applied during CG iterations in barrier algorithms ↩

Default: 0

value	meaning
0	(no) Not applied.
1	(chol) Preconditioner is applied.

cg_stoptol *(real)*: Specifies the relative stopping tolerance used for the conjugate gradient (CG) subproblem solves ↩

Range: [0, ∞]

Default: 0.01

conic_numthreads *(integer)*: Specify the number of threads to use for operations in the conic algorithm (when bar_conic_enable = 1) ↩

Specify the number of threads to use for operations in the conic algorithm (when bar_conic_enable = 1). 0 Let Knitro choose the number of threads (currently sets conic_numthreads based on numthreads). n>0 Use n threads for the conic solver.

Range: {0, ..., ∞}

Default: 0

convex *(integer)*: Declare the problem as convex by setting to 1 or non-convex by setting to 0 ↩

Declare the problem as convex by setting to 1 or non-convex by setting to 0. Otherwise, Knitro will try to determine this automatically, but may only be able to do so for simple model forms such as QPs or QCQPs. If your model is specified as (or automatically determined to be) convex, this will cause Knitro to apply specializations and tunings that are often beneficial for convex models to speed up the solution. Currently this option is only active for the Interior/Direct algorithm, but may be applied to other algorithms in the future.

Range: {-1, ..., 1}

Default: -1

cpuplatform *(integer)*: This option can be used to specify the target instruction set architecture for the machine on which Knitro is running ↩

This option can be used to specify the target instruction set architecture for the machine on which Knitro is running. This can be used, for example , to try to produce more consistent Knitro performance across various architectures (at the expense of, perhaps, slower performance on some platforms). This option is currently only used for the Intel Math Kernel Library (MKL) functions used inside Knitro.

Range: {-1, ..., 5}

Default: -1

datacheck *(boolean)*: Specifies whether to perform more extensive data checks to look for errors in the problem input to Knitro (in particular, this option looks for errors in the sparse Jacobian and/or sparse Hessian structure) ↩

Specifies whether to perform more extensive data checks to look for errors in the problem input to Knitro (in particular, this option looks for errors in the sparse Jacobian and/or sparse Hessian structure). The datacheck may have a non- trivial cost for large problems.

Default: 0

delta *(real)*: Specifies the initial trust region radius scaling factor used to determine the initial trust region size ↩

Range: [1e-14, ∞]

Default: 1

feastol *(real)*: Specifies the final relative stopping tolerance for the feasibility error ↩

Specifies the final relative stopping tolerance for the feasibility error. Smaller values of feastol result in a higher degree of accuracy in the solution with respect to feasibility.

Range: [0, ∞]

Default: 1e-06

feastolabs *(real)*: Specifies the final absolute stopping tolerance for the feasibility error ↩

Synonym: feastol_abs

Specifies the final absolute stopping tolerance for the feasibility error. Smaller values of feastol_abs result in a higher degree of accuracy in the solution with respect to feasibility.

Range: [0, ∞]

Default: 0.001

findiff_estnoise *(integer)*: This option can be used to enable an estimate of the noise in the model when using finite-difference gradients ↩

This option can be used to enable an estimate of the noise in the model when using finite-difference gradients. This noise estimate can then be used to set a finite-difference steplength appropriate for the estimated noise level. This can improve performance on models with noise (e.g. noisy black-box optimization models). The cost of the noise estimation procedure is usually a few extra function evaluations.

Default: 0

value	meaning
0	(no) Do not enable any noise estimation procedure for finite-difference gradients.
1	(yes) Enable noise estimation procedure for finite-difference gradients.
2	(withcurv) Enable noise estimation and curvature factor for finite-difference gradients.

findiff_relstepsize *(real)*: Specifies the relative stepsize used for finite-difference gradients during the optimization ↩

Specifies the relative stepsize used for finite-difference gradients during the optimization. This option sets the stepsize for all variables. Note that this option has no affect on the finite-difference derivatives computed for the derivative checker (default values are always used here). It is only used for the finite-difference derivatives computed during the optimization.

Range: [-∞, ∞]

Default: 0

findiff_terminate *(boolean)*: This option specifies the termination criteria when using finite-difference gradients ↩

This option specifies the termination criteria when using finite-difference gradients. The optimality (or KKT) conditions for nonlinear optimization depend on gradient values of the nonlinear objective and constraint functions . When using finite-difference gradients (e.g. gradopt > 1), there will typically be small errors in the computed gradients that will limit the precision in the solution (and the ability to satisfy the optimality conditions). By default,

Knitro will try to estimate these finite-difference gradient errors and terminate when it seems that no more accuracy in the solution is possible. The solution will be treated as optimal as long as it is feasible and the optimality conditions are satisfied either by the optimality tolerances (opttol and opttol_abs) or the error estimates. On some problems, the error estimates may result in extra function evaluations on some iterations, but will often prevent extra iterations that produce no significant improvement in the solution. This special termination can be disabled by setting findiff_terminate = 0 (none).

Default: 1

value	meaning
0	(none) No special criteria; use the standard stopping conditions.
1	(errest) Allow termination based on estimates of the finite-difference error (when no more significant progress is likely).

fstopval *(real)*: Used to implement a custom stopping condition based on the objective function value ↩

Used to implement a custom stopping condition based on the objective function value. Knitro will stop and declare that a satisfactory solution was found if a feasible objective function value at least as good as the value specified by fstopval is achieved. This stopping condition is only active when the absolute value of fstopval is less than objrange.

Range: [-∞, ∞]

Default: maxdouble

ftol *(real)*: The optimization process will terminate if the relative change in the objective function is less than ftol for ftol_iters consecutive feasible iterations ↩

Range: [0, ∞]

Default: 1e-15

ftol_iters *(integer)*: The optimization process will terminate if the relative change in the objective function is less than ftol for ftol_iters consecutive feasible iterations ↩

Range: {1, ..., ∞}

Default: 5

gradopt *(integer)*: Specifies how to compute the gradients of the objective and constraint functions ↩

Default: 1

value	meaning
1	(exact) User provides a routine for computing the exact gradients.
2	(forward) Knitro computes gradients by forward finite differences.
3	(central) Knitro computes gradients by central finite differences.

hessopt *(integer)*: Specifies how to compute the (approximate) Hessian of the Lagrangian ↩

Default: 0

value	meaning
0	(auto)
1	(exact) User provides a routine for computing the exact Hessian.
2	(bfgs) Knitro computes a (dense) quasi-Newton BFGS Hessian.
3	(sr1) Knitro computes a (dense) quasi-Newton SR1 Hessian.
4	(product_findiff) Knitro computes Hessian-vector products using finite-differences.
5	(product) User provides a routine to compute the Hessian-vector products.
6	(lbfgs) Knitro computes a limited-memory quasi-Newton BFGS Hessian (its size is determined by the option lmsize).
7	(gauss_newton) Knitro computes a Gauss-Newton approximation of the hessian (available for least-squares only, and default value for least-squares)

honorbnds *(integer)*: Indicates whether or not to enforce satisfaction of simple variable bounds throughout the optimization ↩

Indicates whether or not to enforce satisfaction of simple variable bounds throughout the optimization. This option and the bar_feasible option may be useful in applications where functions are undefined outside the region defined by inequalities.

Default: `-1`

value	meaning
-1	(auto) Knitro automatically determine the best setting.
0	(no) Knitro does not require that the bounds on the variables be satisfied at intermediate iterates.
1	(always) Knitro enforces that the initial point and all subsequent solution estimates satisfy the bounds on the variables.
2	(initpt) Knitro enforces that the initial point satisfies the bounds on the variables.

infeastol *(real)*: Specifies the (relative) tolerance used for declaring infeasibility of a model ↩

Specifies the (relative) tolerance used for declaring infeasibility of a model. Smaller values of infeastol make it more difficult to satisfy the conditions Knitro uses for detecting infeasible models. If you believe Knitro incorrectly declares a model to be infeasible, then you should try a smaller value for infeastol.

Range: [0, ∞]

Default: `1e-08`

infeastol_iters *(integer)*: The optimization process will terminate if the relative change in the feasibility error is less than infeastol for infeastol_iters consecutive infeasible iterations ↩

Range: {1, ..., ∞}

Default: `50`

initpenalty *(real)*: Specifies the initial penalty parameter used in the Knitro merit functions ↩

Specifies the initial penalty parameter used in the Knitro merit functions. The Knitro merit functions are used to balance improvements in the objective function versus improvements in feasibility. A larger initial penalty value places more weight initially on feasibility in the merit function.

Range: [0, ∞]

Default: 10

initvalues *(integer)*: Enable use of initial guess for levels and marginals (subsequent solves) ↩

In case of a MI(NL)P, the initial levels are added as primal initial point for the root relaxation and as primal MIP initial guess. From the primal MIP initial guess Knitro will try to construct a feasible solution.

Default: 1

value	meaning
0	Using no initial values
1	Using all initial values
2	Using only non-default initial values

initvalues0 *(integer)*: Enable use of initial guess for levels and marginals (first solve) ↩

In case of a MI(NL)P, the initial levels are added as primal initial point for the root relaxation and as primal MIP initial guess. From the primal MIP initial guess Knitro will try to construct a feasible solution.

Default: 2

value	meaning
0	Using no initial values
1	Using all initial values
2	Using only non-default initial values

linesearch *(integer)*: Indicates which linesearch strategy to use for the Interior/Direct or SQP algorithm to search for a new acceptable iterate ↩

Indicates which linesearch strategy to use for the Interior/Direct or SQP algorithm to search for a new acceptable iterate. This option has no effect on the Interior/CG or Active Set algorithm.

Default: 0

value	meaning
0	(auto) Let Knitro automatically choose the strategy.
1	(backtrack) Use a simple backtracking scheme.
2	(interpolate) Use a cubic interpolation scheme.
3	(weakwolfe) Use a linesearch that satisfies the weak Wolfe conditions (unconstrained only).

linesearch_maxtrials *(integer)*: Indicates the maximum allowable number of trial points during the

linesearch of the Interior/Direct or SQP algorithm before treating the linesearch step as a failure and generating a new step ↩

Indicates the maximum allowable number of trial points during the linesearch of the Interior/Direct or SQP algorithm before treating the linesearch step as a failure and generating a new step. This option has no effect on the Interior/CG or Active Set algorithm.

Range: {0, ..., ∞}

Default: 3

linsolver *(integer)*: Indicates which linear solver to use to solve linear systems arising in Knitro algorithms ↩

Default: 0

value	meaning
0	(auto) Let Knitro automatically choose the linear solver.
1	(internal) Not currently used; reserved for future use. Same as auto for now.
2	(hybrid) Use a hybrid approach where the solver chosen depends on the particular linear system which needs to be solved.
3	(qr) Use a dense QR method. This approach uses LAPACK QR routines. Since it uses a dense method, it is only efficient for small problems. It may often be the most efficient method for small problems with dense Jacobians or Hessian matrices.
4	(ma27) Use the HSL MA27 sparse symmetric indefinite solver.
5	(ma57) Use the HSL MA57 sparse symmetric indefinite solver.
6	(mklpardiso) Use the Intel MKL PARDISO (parallel, deterministic) sparse symmetric indefinite solver.
7	(ma97) Use the HSL MA97 (parallel, deterministic) sparse symmetric indefinite solver.
8	(ma86) Use the HSL MA86 (parallel, non-deterministic) sparse symmetric indefinite solver.

linsolver_maxitref *(integer)*: Indicates the maximum allowable number of iterative refinement steps applied when a linear system is solved inside Knitro ↩

Indicates the maximum allowable number of iterative refinement steps applied when a linear system is solved inside Knitro. Iterative refinement steps may be applied when there are significant errors (e.g. large residuals) in the linear system solves. Applying more iterative refinement steps may improve the numerical accuracy of the linear solves at extra cost.

Range: {0, ..., ∞}

Default: 2

linsolver_nodeamalg *(integer)*: Controls the node amalgamation setting for the MA57, MA86 and MA97 linear solvers ↩

Controls the node amalgamation setting for the MA57, MA86 and MA97 linear solvers. A value of 0 indicates that the default value should be used for the given linear solver, while a positive value sets the node amalgamation parameter for the linear solver to that specific value.

Range: {0, ..., ∞}

Default: 0

linsolver_numthreads *(integer)*: Specify the number of threads to use for linear system solve operations when linsolver = 6 ↩

Range: {0, ..., ∞}

Default: 0

linsolver_ooc *(integer)*: Indicates whether to use Intel MKL PARDISO out-of-core solve of linear systems when linsolver = mklpardiso ↩

Indicates whether to use Intel MKL PARDISO out-of-core solve of linear systems when linsolver = mklpardiso. This option is only active when linsolver = mklpardiso.

Default: 0

value	meaning
0	(no) Do not use Intel MKL PARDISO out-of-core option.
1	(maybe) Maybe solve out-of-core depending on how much space is needed.
2	(yes) Solve linear systems out-of-core when using Intel MKL PARDISO.

linsolver_ordering *(integer)*: Sets the ordering method used for the linear system solver ↩

Default: -1

value	meaning
-1	(auto) Let Knitro automatically choose the ordering strategy.
0	(best) Choose the best between AMD and METIS (try both).
1	(amd) Use AMD ordering (minimum degree for MKL PARDISO).
2	(metis) Use METIS ordering.

linsolver_pivottol *(real)*: Specifies the initial pivot threshold used in factorization routines ↩

Specifies the initial pivot threshold used in factorization routines. The value should be in the range [0, . . . , 0.5] with higher values resulting in more pivoting (more stable factorizations). Values less than 0 will be set to 0 and values larger than 0.5 will be set to 0.5. If linsolver_pivottol is non-positive, initially no pivoting will be performed. Smaller values may improve the speed of the code but higher values are recommended for more stability (for example, if the problem appears to be very ill-conditioned).

Range: [0, 0.5]

Default: 1e-08

linsolver_scaling *(integer)*: Enables scaling for the linear system solver ↩

Enables scaling for the linear system solver. Applying scaling may allow for more accuracy in the linear system solves, but will generally make the linear system solves more expensive.

Default: 0

value	meaning
0	(none) Do not apply scaling in the linear system solves.
1	(always) Always apply scaling in the linear system solves.

lmsize *(integer)*: Specifies the number of limited memory pairs stored when approximating the Hessian using the limited-memory quasi-Newton BFGS option ↩

Specifies the number of limited memory pairs stored when approximating the Hessian using the limited-memory quasi-Newton BFGS option. The value must be between 1 and 100 and is only used with hessopt = 6. Larger values may give a more accurate, but more expensive, Hessian approximation. Smaller values may give a less accurate, but faster, Hessian approximation. When using the limited memory BFGS approach it is recommended to experiment with different values of this parameter.

Range: {1, ..., 100}

Default: 10

maxfevals *(integer)*: Specifies the maximum number of function evaluations before termination ↩

Specifies the maximum number of function evaluations before termination. Values less than zero imply no limit.

Range: {-∞, ..., ∞}

Default: -1

maxit *(integer)*: Specifies the maximum number of iterations before termination ↩

Synonym: iterlim

Default: GAMS iterlim

value	meaning
0	Let Knitro automatically choose a value based on the problem type. Currently Knitro sets this value to 10000 for LPs/NLPs and 3000 for MIP problems.
n	At most n>0 iterations may be performed before terminating.

maxtime_cpu *(real)*: Specifies, in seconds, the maximum allowable CPU time before termination ↩

Range: [0, 100000000]

Default: 100000000

maxtime_real *(real)*: Specifies, in seconds, the maximum allowable real time before termination ↩

Synonym: reslim

Range: [0, 100000000]

Default: GAMS reslim

ma_maxtime_cpu *(real)*: Specifies, in seconds, the maximum allowable CPU time before termination for the multi-algorithm ("MA") procedure (algorithm=5) ↩

Range: [0, ∞]

Default: 100000000

ma_maxtime_real *(real)*: Specifies, in seconds, the maximum allowable real time before termination for the multi-algorithm ("MA") procedure (algorithm=5) ↩

Range: [0, ∞]

Default: 100000000

ma_outsub *(boolean)*: Enable writing algorithm output to files for the multi-algorithm (algorithm=5) procedure ↩

Default: 0

value	meaning
0	Do not write detailed algorithm output to files.
1	Write detailed algorithm output to files named knitro_ma_*.log.

ma_terminate *(integer)*: Define the termination condition for the multi-algorithm (algorithm=5) procedure ↩

Default: 1

value	meaning
0	Terminate after all algorithms have completed.
1	Terminate at first locally optimal solution.
2	Terminate at first feasible solution estimate.
3	Terminate at first solution estimate of any type.

mip_branchrule *(integer)*: Specifies which branching rule to use for MIP branch and bound procedure ↩

Default: 0

value	meaning
0	(auto) Let Knitro automatically choose the branching rule.
1	(most_frac) Use most fractional (most infeasible) branching.
2	(pseudcost) Use pseudo-cost branching.
3	(strong) Use strong branching (see options mip_strong_candlim, mip_strong_level and mip_strong_maxit for further control of strong branching procedure).

mip_clique *(integer)*: Specifies rules for adding clique cuts ↩

Default: -1

value	meaning
-1	(auto) Automatically determine whether to add clique cuts.
0	(none) Do not add clique cuts.
1	(root) Add clique cuts derived from the root node only.
2	(tree) Add clique cuts derived at every node depending on the solution of the relaxation and the cut generation strategy.

mip_cutfactor *(real)*: This value specifies a limit on the number of cuts added to a node subproblem ↩

This value specifies a limit on the number of cuts added to a node subproblem. If non-negative, a maximum of mip_cutfactor times the number of constraints is possibly appended.

Range: [0, ∞]

Default: 1

mip_cutoff *(real)*: This value specifies the objective cutoff value for MIP ↩

Range: [-∞, ∞]

Default: `maxdouble`

mip_cutting_plane *(boolean)*: Specifies when to apply the cutting plane procedure ↩

Default: `1`

value	meaning
`0`	(none) No cutting plane procedure enabled.
`1`	(root) Perform cutting plane procedure at the root node only.

mip_cut_flowcover *(integer)*: Specifies rules for adding flow cover cuts ↩

Default: `-1`

value	meaning
`-1`	(auto) Automatically determine whether to add flow cover cuts.
`0`	(none) Do not add flow cover cuts.
`1`	(root) Add flow cover cuts derived from the root node only.
`2`	(tree) Add flow cover cuts derived at every node depending on the solution of the relaxation and the cut generation strategy.

mip_cut_probing *(integer)*: Specifies rules for adding probing cuts ↩

Default: `-1`

value	meaning
`-1`	(auto) Automatically determine whether to add probing cuts.
`0`	(none) Do not add probing cuts.
`1`	(root) Add probing cuts derived from the root node only.
`2`	(tree) Add probing cuts derived at every node depending on the solution of the relaxation and the cut generation strategy.

mip_gomory *(integer)*: Specifies rules for adding Gomory mixed-integer cuts ↩

Default: `-1`

value	meaning
`-1`	(auto) Automatically determine whether to add Gomory cuts.
`0`	(none) Do not add Gomory cuts.
`1`	(root) Add Gomory cuts derived from the root node only.
`2`	(tree) Add Gomory cuts derived at every node depending on the solution of the relaxation and the cut generation strategy.

mip_gub_branch *(boolean)*: Specifies whether or not to branch on generalized upper bounds (GUBs) ↩

Default: `0`

value	meaning
0	(no) Do not branch on GUBs.
1	(yes) Allow branching on GUBs.

mip_heuristic_diving *(integer)*: Specifies whether or not to enable the MIP diving heuristic ↩

Specifies whether or not to enable the MIP diving heuristic. This option is a bit-valued option where various diving heuristics can be enabled by activating the corresponding bit value as described below. Setting this option to -1 will use an automatic setting and setting the value to 0 will disable all diving heuristics. Otherwise, set this parameter value to the sum of the values for the individual diving heuristics you wish to enable. For example, to enable only the "fractional" and "linesearch" diving heuristics, you would set this option value to 9 (summing 1 for fractional and 8 for linesearch).

Default: -1

value	meaning
-1	(auto) Let Knitro determine automaticallyfrom mip_heuristic_strategy.
1	(bit 0) Enable fractional diving heuristic.
2	(bit 1) Enable vector length diving heuristic.
4	(bit 2) Enable coefficient diving heuristic.
8	(bit 3) Enable linesearch diving heuristic.
16	(bit 4) Enable guided diving heuristic.

mip_heuristic_feaspump *(integer)*: Specifies whether or not to enable the MIP feasibility pump heuristic ↩

Default: -1

value	meaning
-1	(auto) Let Knitro determine automatically from mip_heuristic_strategy.
0	(off) Feasibility pump heuristic is not applied.
1	(on) Feasibility pump heuristic is enabled.

mip_heuristic_lns *(integer)*: Specifies whether or not to enable the MIP large neighborhood search (LNS) heuristics ↩

Specifies whether or not to enable the MIP large neighborhood search (LNS) heuristics. This option is a bit-valued option where various LNS heuristics can be enabled by activating the corresponding bit value as described below. Setting this option to -1 will use an automatic setting and setting the value to 0 will disable all LNS heuristics. Otherwise, set this parameter value to the sum of the values for the individual LNS heuristics you wish to enable. For example, to enable both the "RENS" and "RINS" LNS heuristics, you would set this option value to 3 (summing 1 for RENS and 2 for RINS).

Default: -1

value	meaning
-1	(auto) Let Knitro determine automatically from mip_heuristic_strategy.
1	(bit 0) Enable relaxation enforced neighborhood search (RENS) heuristic.
2	(bit 1) Enable relaxation induced neighborhood search (RINS) heuristic.

mip_heuristic_localsearch *(integer)*: Specifies whether or not to enable the MIP local search heuristic ↩

Default: `-1`

value	meaning
`-1`	(auto) Let Knitro determine automatically from mip_heuristic_strategy.
`0`	(off) Local search heuristic is not applied.
`1`	(on) Local search heuristic is enabled.

mip_heuristic_maxit *(integer)*: Specifies the maximum number of iterations to allow for MIP heuristic, if one is enabled ↩

Range: {`0`, ..., ∞}

Default: `100`

mip_heuristic_misqp *(integer)*: Specifies whether or not to enable the MIP MISQP heuristic ↩

Default: `-1`

value	meaning
`-1`	(auto) Let Knitro determine automatically from mip_heuristic_strategy.
`0`	(off) MISQP heuristic is not applied.
`1`	(on) MISQP heuristic is enabled.

mip_heuristic_mpec *(integer)*: Specifies whether or not to enable the MIP MPEC heuristic ↩

Default: `-1`

value	meaning
`-1`	(auto) Let Knitro determine automatically from mip_heuristic_strategy.
`0`	(off) MPEC heuristic is not applied.
`1`	(on) MPEC heuristic is enabled.

mip_heuristic_strategy *(integer)*: Specifies the level of effort applied for the MIP heuristic search used to try to find an initial integer feasible point ↩

Default: `-1`

value	meaning
`-1`	(auto) Let Knitro choose the heuristic strategy to apply (if any).
`0`	(none) No heuristic search applied.
`1`	(basic) Apply basic heuristics.
`2`	(advanced) Apply more advanced heuristics.
`3`	(extensive) Apply most extensive heuristics.

mip_heuristic_terminate *(integer)*: Specifies the condition for terminating the MIP heuristic ↩

Default: `1`

value	meaning
1	(feasible) Terminate at first feasible point or iteration limit (whichever comes first).
2	(limit) Always run to the iteration limit.

mip_implications *(boolean)*: Specifies whether or not to add constraints to the MIP derived from logical implications ↩

Default: 1

value	meaning
0	(no) Do not add constraints from logical implications.
1	(yes) Knitro adds constraints from logical implications.

mip_integer_tol *(real)*: This value specifies the threshold for deciding whether or not a variable is determined to be an integer ↩

Range: [0, 1]

Default: 1e-08

mip_intvar_strategy *(integer)*: Specifies how to handle integer variables ↩

Default: 0

value	meaning
0	(none) No special treatment applied.
1	(relax) Relax all integer variables.
2	(mpec) Convert all binary variables to complementarity constraints.

mip_knapsack *(integer)*: Specifies rules for adding MIP knapsack cuts ↩

Default: -1

value	meaning
-1	(auto) Automatically determine whether to add knapsack cuts.
0	(none) Do not add knapsack cuts.
1	(root) Add knapsack cuts derived from the root node only.
2	(tree) Add knapsack cuts derived at every node depending on the solution of the relaxation and the cut generation strategy.

mip_liftproject *(integer)*: Specifies rules for adding lift and project cuts ↩

Default: -1

value	meaning
-1	(auto) Automatically determine whether to add lift and project cuts.
0	(none) Do not add lift and project cuts.
1	(root) Add lift and project cuts at the root node only.

mip_lpalg *(integer)*: Specifies which algorithm to use for any linear programming (LP) subproblem solves that may occur in the MIP branch-and-bound procedure ↩

Specifies which algorithm to use for any linear programming (LP) subproblem solves that may occur in the MIP branch-and-bound procedure. LP subproblems may arise if the problem is a mixed integer linear program (MILP), or if using mip_method = HQG. (Nonlinear programming subproblems use the algorithm specified by the algorithm option.)

Default: `0`

value	meaning
`0`	(auto) Let Knitro automatically choose an algorithm, based on the problem characteristics.
`1`	(direct) Use the Interior/Direct (barrier) algorithm.
`2`	(cg) Use the Interior/CG (barrier) algorithm.
`3`	(active) Use the Active Set (simplex) algorithm.

mip_maxnodes *(integer)*: Specifies the maximum number of nodes explored (0 means no limit) ↩

Synonym: nodlim

Range: {`0`, ..., ∞}

Default: `GAMS nodlim`

mip_maxsolves *(integer)*: Specifies the maximum number of subproblem solves allowed (0 means no limit) ↩

Range: {`0`, ..., ∞}

Default: `0`

mip_maxtime_cpu *(real)*: Specifies the maximum allowable CPU time in seconds for the complete MIP solution ↩

Specifies the maximum allowable CPU time in seconds for the complete MIP solution. Use maxtime_cpu to additionally limit time spent per subproblem solve.

Range: [`0`, ∞]

Default: `100000000`

mip_maxtime_real *(real)*: Specifies the maximum allowable real time in seconds for the complete MIP solution ↩

Specifies the maximum allowable real time in seconds for the complete MIP solution. Use maxtime_real to additionally limit time spent per subproblem solve.

Range: [`0`, ∞]

Default: `GAMS reslim`

mip_method *(integer)*: Specifies which MIP method to use ↩

Default: `0`

value	meaning
0	(auto) Let Knitro automatically choose the method.
1	(BB) Use the standard branch-and-bound method.
2	(HQG) Use the hybrid Quesada-Grossman method (for convex, nonlinear problems only).
3	(MISQP) Use mixed-integer SQP method (allows for non-relaxable integer variables).

mip_mir *(integer)*: Specifies rules for adding mixed-integer rounding cuts ↩

Default: -1

value	meaning
-1	(auto) Let Knitro decide whether to add mixed-integer rounding cuts.
0	(none) Do not add mixed-integer rounding cuts.
1	(root) Add mixed-integer rounding cuts derived from the root node only.
2	(tree) Add mixed-integer rounding cuts derived at every node depending on the solution of the relaxation and the cut generation strategy.

mip_multistart *(boolean)*: Use to enable MIP multi-start at the branch-and-bound level ↩

Default: 0

value	meaning
0	(off) Do not enable MIP multi-start for branch-and-bound.
1	(on) Enable MIP multi-start for branch-and-bound.

mip_nodealg *(integer)*: Specifies which algorithm to use for standard node subproblem solves in MIP (same options as algorithm user option) ↩

Range: {0, ..., 5}

Default: 0

mip_numthreads *(integer)*: Specify the number of threads to use for MIP branch-and-bound (when mip_method = 1) ↩

Specify the number of threads to use for MIP branch-and-bound (when mip_method = 1). 0 Let Knitro choose the number of threads. n>0 Use n threads for the MIP branch-and-bound.

Range: {0, ..., ∞}

Default: 0

mip_opt_gap_abs *(real)*: The absolute optimality gap stop tolerance for MIP ↩

Synonym: optca

Range: [-1, ∞]

Default: GAMS optca

mip_opt_gap_rel *(real)*: The relative optimality gap stop tolerance for MIP ↩

Synonym: optcr

Range: [-1, ∞]

Default: `GAMS optcr`

mip_outinterval *(integer)*: Specifies node printing interval for mip_outlevel when mip_outlevel > 0 ↩

Default: `0`

value	meaning
0	Let Knitro decide.
1	Print output every node.
2	Print output every 2nd node.
n	Print output every Nth node.

mip_outlevel *(integer)*: Specifies how much MIP information to print ↩

Default: `2`

value	meaning
0	(none) Do not print any MIP node information.
1	(iters) Print one line of output for every node.
2	(iterstime) Also print accumulated time for every node.
3	(root) Also print detailed log from root node solve.

mip_pseudoinit *(integer)*: Specifies the method used to initialize pseudo-costs corresponding to variables that have not yet been branched on in the MIP method ↩

Default: `0`

value	meaning
0	Let Knitro automatically choose the method.
1	Initialize using the average value of computed pseudo-costs.
2	Initialize using strong branching.

mip_relaxable *(boolean)*: Specifies whether integer variables are relaxable ↩

Default: `1`

value	meaning
0	(none) Integer variables are not relaxable.
1	(all) All integer variables are relaxable.

mip_restart *(boolean)*: Specifies whether to enable the MIP restart procedure ↩

Default: `1`

value	meaning
0	(off) Do not enable the MIP restart procedure.
1	(on) Enable the MIP restart procedure.

mip_rootalg *(integer)*: Specifies which algorithm to use for the root node solve in MIP (same options as algorithm user option) ↩

Range: {0, ..., 5}

Default: 0

mip_rounding *(integer)*: Specifies the MIP rounding rule to apply ↩

Default: -1

value	meaning
-1	(auto) Let Knitro choose the rounding rule.
0	(none) No rounding heuristic is used.
2	(heur_only) Round using a fast heuristic only.
3	(nlp_sometimes) Round and solve a subproblem if likely to succeed.
4	(nlp_always) Always round and solve a subproblem.

mip_selectdir *(boolean)*: Specifies the MIP node selection direction rule (for tiebreakers) for choosing the next node in the branch-and-bound tree ↩

Default: 0

value	meaning
0	(down) Choose the down (i.e. <=) node first.
1	(up) Choose the up (i.e. >=) node first.

mip_selectrule *(integer)*: Specifies the MIP select rule for choosing the next node in the branch-and-bound tree ↩

Default: 0

value	meaning
0	(auto) Let Knitro choose the node selection rule.
1	(depth_first) Search the tree using a depth first procedure.
2	(best_bound) Select the node with the best relaxation bound.
3	(combo_1) Use depth first unless pruned, then best bound.

mip_strong_candlim *(integer)*: Specifies the maximum number of candidates to explore for MIP strong branching ↩

Range: {-∞, ..., ∞}

Default: 128

mip_strong_level *(integer)*: Specifies the maximum number of tree levels on which to perform MIP strong branching ↩

Range: {-∞, ..., ∞}

Default: `10`

mip_strong_maxit *(integer)*: Specifies the maximum number of iterations to allow for MIP strong branching solves ↩

Range: {-∞, ..., ∞}

Default: `1000`

mip_terminate *(integer)*: Specifies conditions for terminating the MIP algorithm ↩

Default: `0`

value	meaning
`0`	(optimal) Terminate at optimum.
`1`	(feasible) Terminate at first integer feasible point.

mip_zerohalf *(integer)*: Specifies rules for adding zero-half cuts ↩

Default: `-1`

value	meaning
`-1`	(auto) Automatically determine whether to add zero-half cuts.
`0`	(none) Do not add zero-half cuts.
`1`	(root) Add zero-half cuts derived from the root node only.
`2`	(tree) Add zero-half cuts derived at every node depending on the solution of the relaxation and the cut generation strategy.

ms_deterministic *(boolean)*: Indicates whether Knitro multi-start procedure will be deterministic (when ms_terminate = 0) ↩

Default: `1`

value	meaning
`0`	(no) multithreaded multi-start is non-deterministic.
`1`	(yes) multithreaded multi-start is deterministic (when ms_terminate = 0).

ms_enable *(boolean)*: Indicates whether Knitro will solve from multiple start points to find a better local minimum ↩

Default: `0`

value	meaning
`0`	(no) Knitro solves from a single initial point.
`1`	(yes) Knitro solves using multiple start points.

ms_initpt_cluster *(boolean)*: The strategy for clustering initial points in multi-start ↩

Default: 0

value	meaning
0	(none) Do not apply clustering.
1	(sl) Apply single linkage based clustering.

ms_maxbndrange *(real)*: Specifies the maximum range that an unbounded variable can take when determining new start points ↩

Specifies the maximum range that an unbounded variable can take when determining new start points. If a variable is unbounded in one or both directions, then new start point values are restricted by the option. If x_i is such a variable, then all initial values satisfy $\max\{b_i^L, x_i^0 - \texttt{ms\_maxbndrange}/2\} \leq x_i \leq \min\{b_i^U, x_i^0 + \texttt{ms\_maxbndrange}/2\}$, where x_i^0 is the initial value of x_i provided by the user, and b_i^L and b_i^U are the variable bounds (possibly infinite) on x_i . This option has no effect unless ms_enable = yes.

Range: [-∞, ∞]

Default: 1000

ms_maxsolves *(integer)*: Specifies how many start points to try in multi-start ↩

Specifies how many start points to try in multi-start. This option has no effect unless ms_enable = yes.

Default: 0

value	meaning
0	Let Knitro automatically choose a value based on the problem size and context.
n	Try n>0 start points.

ms_maxtime_cpu *(real)*: Specifies, in seconds, the maximum allowable CPU time before termination ↩

Specifies, in seconds, the maximum allowable CPU time before termination. The limit applies to the operation of Knitro since multi-start began; in contrast, the value of maxtime_cpu limits how long Knitro iterates from a single start point. Therefore, ms_maxtime_cpu should be greater than maxtime_cpu. This option has no effect unless ms_enable = yes.

Range: [-∞, ∞]

Default: 100000000

ms_maxtime_real *(real)*: Specifies, in seconds, the maximum allowable real time before termination ↩

Specifies, in seconds, the maximum allowable real time before termination. The limit applies to the operation of Knitro since multi-start began; in contrast, the value of maxtime_real limits how long Knitro iterates from a single start point. Therefore, ms_maxtime_real should be greater than maxtime_real. This option has no effect unless ms_enable = yes.

Range: [-∞, ∞]

Default: 100000000

ms_numthreads *(integer)*: Specify the number of threads to use for multi-start (when ms_enable = 1) ↩

Specify the number of threads to use for multi-start (when ms_enable = 1). 0 Let Knitro choose the number of threads (currently sets ms_numthreads based on numthreads). n>0 Use n threads for the multi-start (solve n problems in parallel).

Range: {0, ..., ∞}

Default: 0

ms_num_to_save *(integer)*: Specifies the number of distinct feasible points to save in a file named knitro_mspoints ↩

Specifies the number of distinct feasible points to save in a file named knitro_mspoints.log. Each point results from a Knitro solve from a different starting point, and must satisfy the absolute and relative feasibility tolerances. The file stores points in order from best objective to worst. Points are distinct if they differ in objective value or some component by the value of ms_savetol using a relative tolerance test. This option has no effect unless ms_enable = yes.

Range: {0, ..., ∞}

Default: 0

ms_outsub *(boolean)*: Enable writing algorithm output to files for the parallel multi-start procedure ↩

Default: 0

value	meaning
0	Do not write detailed algorithm output to files.
1	Write detailed algorithm output to files named knitro_ms_∗.log.

ms_savetol *(real)*: Specifies the tolerance for deciding if two feasible points are distinct ↩

Specifies the tolerance for deciding if two feasible points are distinct. Points are distinct if they differ in objective value or some component by the value of ms_savetol using a relative tolerance test. A large value can cause the saved feasible points in the file knitro_mspoints.log to cluster around more widely separated points. This option has no effect unless ms_enable = yes. and ms_num_to_save is positive.

Range: [-∞, ∞]

Default: 1e-06

ms_seed *(integer)*: Seed value used to generate random initial points in multi-start; should be a non-negative integer ↩

Range: {0, ..., ∞}

Default: 0

ms_startptrange *(real)*: Specifies the maximum range that each variable can take when determining new start points ↩

Specifies the maximum range that each variable can take when determining new start points. If a variable has upper and lower bounds and the difference between them is less than or equal to ms_startptrange, then new start point values for the variable can be any number between its upper and lower bounds. If the variable is unbounded in one or both directions, or the difference between bounds is greater than ms_startptrange, then new start point values are restricted by the option. If x_i is such a variable, then all initial values satisfy $\max\{b_i^L, x_i^0 - \tau\} \leq x_i \leq \min\{b_i^U, x_i^0 + \tau\}, \tau = \min\{\texttt{ms\_startptrange}/2, \texttt{ms\_maxbndrange}/2\}$ where x_i^0 is the initial value of x_i provided by the user, and b_i^L and b_i^U are the variable bounds (possibly infinite) on x_i . This option has no effect unless ms_enable = yes.

Range: [-∞, ∞]

Default: `1e+20`

ms_terminate *(integer)*: Specifies the condition for terminating multi-start ↩

Specifies the condition for terminating multi-start. This option has no effect unless ms_enable = yes.

Default: `0`

value	meaning
`0`	(maxsolves) Terminate after ms_maxsolves.
`1`	(optimal) Terminate after the first local optimal solution is found or ms_maxsolves, whichever comes first.
`2`	(feasible) Terminate after the first feasible solution estimate is found or ms_maxsolves, whichever comes first.
`3`	(any) Terminate after the first solution estimate of any type is found or ms_maxsolves, whichever comes first.
`4`	(rulebased) Terminate using rules that estimate when the probability of finding new local solutions is low.

names *(boolean)*: Enable to pass variable and equation names to Knitro ↩

Default: `0`

ncvx_qcqp_init *(integer)*: Specifies the initialization strategy used for non-convex QPs and QCQPs ↩

Specifies the initialization strategy used for non-convex QPs and QCQPs. In particular, these strategies may be more likely to cause Knitro to find global or better local solutions on non-convex quadratic programs (QPs) or non-convex quadratically constrained quadratic programs (QCQPs).

Default: `-1`

value	meaning
`-1`	(auto) Knitro will automatically determine the strategy.
`0`	(none) No special initialization strategy is used.
`1`	(linear) Initialize by solving a linear relaxation.
`2`	(hybrid) Initialize by solving a hybrid formulation.
`3`	(penalty) Initialize by solving a penalty formulation.
`4`	(cvxquad) Initialize by solving a convex quadratic relaxation.

newpoint *(integer)*: Specifies additional action to take after every iteration in a solve of a continuous problem, or after every new incumbent of the NLPBB algorithm ↩

Specifies additional action to take after every iteration in a solve of a continuous problem, or after every new incumbent of the NLPBB algorithm. For a continuous problem, an iteration of Knitro results in a new point that is closer to a solution. The new point includes values of x and Lagrange multipliers lambda. For the NLPBB algorithm, the new incumbent includes values of x.

Default: `0`

value	meaning
`0`	(none) Knitro takes no additional action.
`1`	(saveone) Knitro writes x and lambda to the file knitro_newpoint.log. Previous contents of the file are overwritten.
`2`	(saveall) Knitro appends x and lambda to the file knitro_newpoint.log. Warning: this option can generate a very large file. All iterates, including the start point, crossover points, and the final solution are saved.

objrange *(real)*: Specifies the extreme limits of the objective function for purposes of determining unboundedness ↩

Specifies the extreme limits of the objective function for purposes of determining unboundedness. If the magnitude of the objective function becomes greater than objrange for a feasible iterate, then the problem is determined to be unbounded and Knitro proceeds no further.

Range: [0, ∞]

Default: `1e+20`

option_file *(string)*: additional option file name - read only by KNITRO solver lib ↩

opttol *(real)*: Specifies the final relative stopping tolerance for the KKT (optimality) error ↩

Specifies the final relative stopping tolerance for the KKT (optimality) error. Smaller values of opttol result in a higher degree of accuracy in the solution with respect to optimality.

Range: [0, ∞]

Default: `1e-06`

opttolabs *(real)*: Specifies the final absolute stopping tolerance for the KKT (optimality) error ↩

Synonym: opttol_abs

Specifies the final absolute stopping tolerance for the KKT (optimality) error. Smaller values of opttol_abs result in a higher degree of accuracy in the solution with respect to optimality.

Range: [0, ∞]

Default: `0.001`

outappend *(boolean)*: Specifies whether output should be started in a new file, or appended to existing files ↩

Specifies whether output should be started in a new file, or appended to existing files. The option affects knitro.log. It does not affect knitro_newpoint.log, which is controlled by option newpoint.

Default: `0`

value	meaning
0	(no) Erase any existing files when opening for output.
1	(yes) Append output to any existing files.

outdir *(string)*: Specifies a single directory as the location to write all output files ↩

Specifies a single directory as the location to write all output files. The option should be a full pathname to the directory, and the directory must already exist.

outlev *(integer)*: Controls the level of output produced by Knitro ↩

Default: 2

value	meaning
0	(none) Printing of all output is suppressed.
1	(summary) Print only summary information.
2	(iter_10) Print basic information every 10 iterations.
3	(iter) Print basic information at each iteration.
4	(iter_verbose) Print basic information and the function count at each iteration.
5	(iter_x) Print all the above, and the values of the solution vector x.
6	(all) Print all the above, and the values of the constraints c at x and the Lagrange multipliers lambda.

outmode *(integer)*: Specifies where to direct the output from Knitro ↩

Default: 0

value	meaning
0	(screen) Output is directed to standard out (e.g., screen).
1	(file) Output is sent to a file named knitro.log.
2	(both) Output is directed to both the screen and file knitro.log.

outname *(string)*: Use to specify a custom filename when output is written to a file using outmode ↩

output_time *(boolean)*: print output on where time is used ↩

Default: 0

out_csvinfo *(boolean)*: Controls whether or not to generates a file knitro_solve ↩

Controls whether or not to generates a file knitro_solve.csv containing solve information in comma separated format.

Default: 0

value	meaning
0	(no) No solution information file is generated.
1	(yes) The knitro_solve.csv solution file is generated.

out_csvname *(string)*: Use to specify a custom csv filename when using out_csvinfo ↩

out_hints *(boolean)*: Specifies whether to print diagnostic hints (e.g. about user option settings) after solving ↩

Default: 1

value	meaning
0	(no) Do not print any hints.
1	(yes) Print diagnostic hints on occasion.

presolve *(boolean)*: Determine whether or not to use the Knitro presolver to try to simplify the model by removing variables or constraints ↩

Default: 1

value	meaning
0	(no) Do not use the Knitro presolver.
1	(yes) Enable the Knitro presolver.

presolveop_redundant *(integer)*: Determine whether or not to enable the Knitro presolve operation to detect and remove redundant constraints ↩

Default: 1

value	meaning
0	(none) Do not remove redundant constraints.
1	(dupcon) Detect and remove duplicate constraints.
2	(depcon) Detect and remove linearly dependent constraints.

presolveop_substitution *(integer)*: Determine whether or not to enable the Knitro presolve operation to substitute out variables when possible ↩

Default: -1

value	meaning
-1	(auto) Automatically determined (may depend on the algorithm).
0	(none) Do not perform any variable substitution.
1	(simple) Enable simple substitutions involving doubleton equality constraints.
2	(all) Enable all possible variable substitutions.

presolveop_substitution_tol *(real)*: Tolerance for applying a substitution ↩

Tolerance for applying a substitution. This is a relative tolerance on coefficients involved with the substituted variable. Higher values mean that less reductions will be applied (potentially improving numerical focus). Zero value means all possible substitutions are applied.

Range: [0, ∞]

Default: `0.01`

presolveop_tighten *(integer)*: Determine whether or not to enable the Knitro presolve operation to tighten variable bounds or coefficients ↩

Default: `-1`

value	meaning
`-1`	(auto) Automatically determined (may depend on the algorithm).
`0`	(none) Do not tighten variable bounds (unless it removes a constraint).
`1`	(varbnd) Enable tightening variable bounds always.
`2`	(coef) Enable tightening coefficients in linear constraints.
`3`	(all) Enable tightening variable bounds and coefficients.

presolve_initpt *(integer)*: Control whether the Knitro presolver can shift a user-supplied initial point ↩

Default: `-1`

value	meaning
`-1`	(auto) Let Knitro automatically choose whether to allow shifting.
`0`	(noshift) Do not allow presolver to shift user-supplied initial point.
`1`	(linshift) Allow presolver to shift user-supplied initial point if it only appears in linear constraints.
`2`	(anyshift) Allow presolver to shift any user-supplied initial point.

presolve_level *(integer)*: Set the level of presolve operations to enable through the Knitro presolver ↩

Set the level of presolve operations to enable through the Knitro presolver. A higher presolve level enables more complex presolve operations.

Default: `-1`

value	meaning
`-1`	(auto) Let Knitro automatically choose the presolve level.
`1`	(level1) Enable the most basic presolve operations.
`2`	(level2) Enable more advanced presolve operations.

presolve_passes *(integer)*: Set a maximum limit on the number of passes through the Knitro presolve operations ↩

Range: {`0`, ..., ∞}

Default: `10`

presolve_tol *(real)*: Determines the tolerance used by the Knitro presolver to remove variables and constraints from the model ↩

Determines the tolerance used by the Knitro presolver to remove variables and constraints from the model. If you believe the Knitro presolver is incorrectly modifying the model, use a smaller value for this tolerance (or turn the presolver off).

Range: [`0`, ∞]

Default: `1e-06`

qextractalg *(integer)*: quadratic extraction algorithm in GAMS interface ↩

Default: `0`

value	meaning
0	Automatic
1	ThreePass: Uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage.
2	DoubleForward: Uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.
3	Concurrent: Uses ThreePass and DoubleForward in parallel. As soon as one finishes, the other one stops.

restarts *(integer)*: Specifies whether or not to enable automatic restarts in Knitro ↩

Specifies whether or not to enable automatic restarts in Knitro. When enabled, if a Knitro algorithm seems to be converging slowly or not converging, the algorithm will automatically restart, which may help with convergence.

Default: `-1`

value	meaning
0	No automatic restarts allowed.
n	At most n>0 automatic restarts may be performed.

restarts_maxit *(integer)*: When restarts are enabled, this option can be used to specify a maximum number of iterations before enforcing a restart ↩

Default: `0`

value	meaning
0	No iteration limit on restarts enforced.
n	At most n>0 iterations are allowed without convergence before enforcing an automatic restart, if restarts are enabled.

scale *(integer)*: Specifies whether to perform problem scaling of the objective function, constraint functions, or possibly variables ↩

Specifies whether to perform problem scaling of the objective function, constraint functions, or possibly variables. If scaling is performed, internal computations, including some aspects of the optimality tests, are based on the scaled values, though the feasibility error is always computed in terms of the original, unscaled values.

Default: `1`

value	meaning
0	(no) No scaling is performed.
1	(user_internal) User provided scaling is used if defined, otherwise Knitro internal scaling is applied.
2	(user_none) User provided scaling is used if defined, otherwise no scaling is applied.
3	(internal) Knitro internal scaling is applied.

scale_vars *(boolean)*: Specifies the strategy for scaling variables ↩

Specifies the strategy for scaling variables. If scaling is performed, internal computations, including some aspects of the optimality tests, are based on the scaled values, though the feasibility error is always computed in terms of the original, unscaled values.

Default: `0`

value	meaning
`0`	(none) No variable scaling is performed.
`1`	(bnds) Scaling of variables is applied based on their bound values.

soc *(integer)*: Specifies whether or not to try second order corrections (SOC) ↩

Specifies whether or not to try second order corrections (SOC). A second order correction may be beneficial for problems with highly nonlinear constraints.

Default: `1`

value	meaning
`0`	(no) No second order correction steps are attempted.
`1`	(maybe) Second order correction steps may be attempted on some iterations.
`2`	(yes) Second order correction steps are always attempted if the original step is rejected and there are nonlinear constraints.

strat_warm_start *(boolean)*: Specifies whether or not to invoke a warm-start strategy ↩

Specifies whether or not to invoke a warm-start strategy. A warm-start strategy may be beneficial when an initial point close to the solution can be provided. For example, this may occur when solving a sequence of closely related problems, and the solution from one problem can be used to initialize (or warm-start) the next problem in the sequence. The Knitro warm-start strategy will use this information to tune the algorithms to try to converge more quickly in this case. If the initial point is not sufficiently close to the solution, or is too infeasible, the warm-start strategy may not be helpful. This option is currently only used for the Knitro barrier/interior-point algorithms. In this case it may also be useful to experiment with different (smaller than default) values for the initial barrier parameter bar_initmu. In general, the closer the initial point is to the solution, the smaller this value should be (Knitro will try by default to initialize this to a good value when applying a warm-start strategy).

Default: `0`

value	meaning
`0`	(no) No warm-start strategy is applied.
`1`	(yes) Knitro will apply a warm-start strategy with special tunings.

threads *(integer)*: default thread count ↩

Synonyms: gthreads numthreads

Controls the number of threads to use. Non-positive values are interpreted as the number of cores to leave free so setting `threads` to 0 uses all available cores while setting `threads` to -1 leaves one core free for other tasks.

By default, Knitro decides automatically how to use the threads specified. For example, if the multi-start method is enabled, it can run in parallel threads. If the multi-algorithm `algorithm=5` is selected, the different algorithms can be run in parallel threads if threads are allocated. In case only one algorithm is running, multiple threads can be allocated to the MKL BLAS library or to the PARDISO linear solver via the `threads` option: see the `blasoption`, `par_blasnumthreads`, and `par_lsnumthreads` options for details.

Range: {-∞, ..., ∞}

Default: `GAMS threads`

tuner *(boolean)*: Indicates whether to invoke the Knitro-Tuner ↩

Default: `0`

value	meaning
`0`	(off) Do not invoke the Knitro-Tuner.
`1`	(on) Invoke the Knitro-Tuner.

tuner_maxtime_cpu *(real)*: Specifies, in seconds, the maximum allowable CPU time before terminating the Knitro-Tuner ↩

Specifies, in seconds, the maximum allowable CPU time before terminating the Knitro-Tuner. The limit applies to the operation of Knitro since the Knitro- Tuner began. In contrast, the value of maxtime_cpu places a time limit on each individual Knitro-Tuner solve for a particular option setting. Therefore, tuner_maxtime_cpu should be greater than maxtime_cpu. This option has no effect unless tuner = on.

Range: [0, ∞]

Default: `100000000`

tuner_maxtime_real *(real)*: Specifies, in seconds, the maximum allowable real time before terminating the Knitro-Tuner ↩

Specifies, in seconds, the maximum allowable real time before terminating the Knitro-Tuner. The limit applies to the operation of Knitro since the Knitro- Tuner began. In contrast, the value of maxtime_real places a time limit on each individual Knitro-Tuner solve for a particular option setting. Therefore, tuner_maxtime_real should be greater than maxtime_real. This option has no effect unless tuner = on.

Range: [0, ∞]

Default: `100000000`

tuner_optionsfile *(string)*: Can be used to specify the location of a Tuner options file ↩

tuner_outsub *(integer)*: Enable writing additional Tuner subproblem solve output to files for the Knitro-Tuner procedure (tuner=1) ↩

Default: `0`

value	meaning
`0`	Do not write detailed solve output to files.
`1`	Write summary solve output to a file named knitro_tuner_summary.log.
`2`	Write detailed individual solve output to files named knitro_tuner_*.log.

tuner_terminate *(integer)*: Define the termination condition for the Knitro-Tuner procedure (tuner=1) ↩

Default: `0`

value	meaning
`0`	Terminate after all solves have completed.
`1`	Terminate at first locally optimal solution.
`2`	Terminate at first feasible solution estimate.
`3`	Terminate at first solution estimate of any type.

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

The optimization process will terminate if the relative change in all components of the solution point estimate is less than xtol for xtol_iters. consecutive iterations. If using the Interior/Direct or Interior/CG algorithm and the barrier parameter is still large, Knitro will first try decreasing the barrier parameter before terminating.

Range: [`0`, ∞]

Default: `1e-12`

xtol_iters *(integer)*: The optimization process will terminate if the relative change in the solution estimate is less than xtol for xtol_iters consecutive iterations ↩

The optimization process will terminate if the relative change in the solution estimate is less than xtol for xtol_iters consecutive iterations. If set to 0, Knitro chooses this value based on the solver and context. Currently Knitro sets this value to 3 unless the MISQP algorithm is being used, in which case the value is set to 1 by default.

Range: {`0`, ..., ∞}

Default: `0`

5.23.6 Knitro Termination Test and Optimality

5.23.6.1 Continuous problems

The first-order conditions for identifying a locally optimal solution of the problem (1) are:

$$\begin{aligned}
\nabla_x \mathcal{L}(x,\lambda) = \nabla f(x) + \sum_{i=1\ldots m} \lambda_i^c \nabla c_i(x) + \sum_{j=1\ldots n} \lambda_j^b \quad &= 0 \\
\lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))] \quad &= 0, \quad i = 1\ldots m \\
\lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)] \quad &= 0, \quad j = 1\ldots n \\
c_i^L \leq c_i(x) \quad &\leq c_i^U, \quad i = 1\ldots m \\
b_j^L \leq x_j \quad &\leq b_j^U, \quad j = 1\ldots n \\
\lambda_i^c \quad &\geq 0, \quad i \in \mathcal{I},\ c_i^L = -\infty,\ c_i^U \text{ finite} \\
\lambda_i^c \quad &\leq 0, \quad i \in \mathcal{I},\ c_i^U = +\infty,\ c_i^L \text{ finite} \\
\lambda_j^b \quad &\geq 0, \quad j \in \mathcal{B},\ b_j^L = -\infty,\ b_j^U \text{ finite} \\
\lambda_j^b \quad &\leq 0, \quad j \in \mathcal{B},\ b_j^U = +\infty,\ b_j^L \text{ finite}
\end{aligned} \tag{3}$$

where $\mathcal{I}$ and $\mathcal{B}$ represent the sets of indices corresponding to the general inequality constraints and non-fixed variable bound constraints respectively, λ_i^c is the Lagrange multiplier corresponding to constraint

$c_i(x)$, and λ_j^b is the Lagrange multiplier corresponding to the simple bounds on x_j. There is exactly one Lagrange multiplier for each constraint and variable. The Lagrange multiplier may be restricted to take on a particular sign depending on the corresponding constraint or variable bounds.

In Knitro we define the feasibility error (`FeasErr`) at a point x^k to be the maximum violation of the constraints of (1), i.e.,

$$\text{FeasErr} = \max_{i=1\ldots m,\ j=1\ldots n}(0, (c_i^L - c_i(x^k)), (c_i(x^k) - c_i^U), (b_j^L - x_j^k), (x_j^k - b_j^U)),$$

while the optimality error (`OptErr`) is defined as the maximum violation of the first three conditions of (3)

$$\text{OptErr} = \max_{i=1\ldots m,\ j=1\ldots n}(\|\nabla_x \mathcal{L}(x^k, \lambda^k)\|_\infty, \lambda_i^c \min[(c_i(x) - c_i^L), (c_i^U - c_i(x))], \lambda_j^b \min[(x_j - b_j^L), (b_j^U - x_j)]).$$

The remaining conditions on the sign of the multipliers are enforced explicitly throughout the optimization. In order to take into account problem scaling in the termination test, the following scaling factors are defined In order to take into account problem scaling in the termination test, the following scaling factors are defined

$$\tau_1 = \max(1, (c_i^L - c_i(x^0)), (c_i(x^0) - c_i^U), (b_j^L - x_j^0), , (x_j^0 - b_j^U))),$$
$$\tau_2 = \begin{cases} \max(1, \|\nabla f(x^k)\|_\infty), & \text{for constrained problems,} \\ \max(1, \min(|f(x^k)|, \|\nabla f(x^0)\|_\infty)), & \text{for unconstrained problems,} \end{cases}$$

where x^0 represents the initial point. The special treatment for unconstraints problems is necessary, as for these problems, $\|\nabla f(x^k)\|_\infty \to 0$ as a solution is approached, thus $\max(1, \|\nabla f(x^k)\|_\infty)$ would not be effective.

Knitro stops and declares `Locally optimal solution found` if the following stopping conditions are satisfied:

$$\text{FeasErr} \leq \max(\tau_1 * \text{feastol}, \text{feastolabs}) \tag{4}$$

$$\text{OptErr} \leq \max(\tau_2 * \text{opttol}, \text{opttolabs}) \tag{5}$$

where feastol, opttol, feastolabs and opttolabs are user-defined options (see section Usage).

This stopping test is designed to give the user much flexibility in deciding when the solution returned by Knitro is accurate enough. One can use a purely scaled stopping test (which is the recommended and default option) by setting feastolabs and opttolabs equal to `0.0e0`. Likewise, an absolute stopping test can be enforced by setting feastol and opttol equal to `0.0e0`.

Unbounded problems

Since by default Knitro uses a relative/scaled stopping test it is possible for the optimality conditions to be satisfied for an unbounded problem. For example, if $\tau_2 \to \infty$ while the optimality error `OptErr` stays bounded, condition (5) will eventually be satisfied for some `opttol`>0. If you suspect that your problem may be unbounded, using an absolute stopping test will allow Knitro to detect this.

5.23.6.2 Discrete problems

Algorithms for solving versions of (1) where one or more of the variables are restricted to take on only discrete values, proceed by solving a sequence of continuous relaxations, where the discrete variables are *relaxed* such that they can take on any continuous value. The *global* solutions $f(x_R)$ of these relaxed problems provide a lower bound on the optimal objective value for problem (1) (upper bound if maximizing). If a feasible point is found for problem (1) that satisfies the discrete restrictions on the variables, then this provides an upper bound on the optimal objective value of problem (1) (lower bound if maximizing). We will refer to these feasible points as *incumbent* points and denote the objective value at an incumbent point by $f(x_I)$. Assuming all the continuous subproblems have been solved to global optimality (if the problem is convex, all local solutions are global solutions), an optimal solution of problem (1) is verified when the lower bound and upper bound are equal.

Knitro declares optimality for a discrete problem when the gap between the best (i.e., largest) lower bound $f(x_R)$ and the best (i.e., smallest) upper bound $f(x_I)$ is less than a threshold determined by the user options mip_opt_gap_abs and mip_opt_gap_rel. Specifically, Knitro declares optimality when either

$$f(x_I) - f(x_R) \leq mip\_integral\_gap\_abs$$

or

$$f(x_I) - f(x_R) \leq mip\_integral\_gap\_rel \cdot \max(1, |f(x_I)|),$$

where mip_opt_gap_abs and mip_opt_gap_rel are typically small positive numbers. Since these termination conditions assume that the continuous subproblems are solved to global optimality and Knitro only finds local solutions of nonconvex, continuous optimization problems, they are only reliable when solving convex, mixed integer problems. The integrality gap $f(x_I) - f(x_R)$ should be non-negative although it may become slightly negative from roundoff error, or if the continuous subproblems are not solved to sufficient accuracy. If the integrality gap becomes largely negative, this may be an indication that the model is nonconvex, in which case `Knitro` may not converge to the optimal solution, and will be unable to verify optimality (even if it claims otherwise).

Note that the default values for mip_opt_gap_abs and mip_opt_gap_rel are taken from the GAMS options optCA and optCR, but an explicit setting of mip_opt_gap_abs or mip_opt_gap_rel will override those.

5.23.7 Knitro Output

If `outlev=0` then all printing of output is suppressed. The description below assumes the default output level (`outlev=2`) except where indicated:

Nondefault Options:

This output lists all user options (see section Usage) which are different from their default values. If nothing is listed in this section then all user options are set to their default values.

Problem Characteristics:

The output begins with a description of the problem characteristics.

Iteration Information - Continuous Problems:

An iteration, in the context of Knitro, is defined as a step which generates a new solution estimate (i.e., a successful step). The columns of the iteration log are as follows:

- `Iter` Iteration number.

- `fCount` The cumulative number of function evaluations, only included if (`outlev>3`)
- `Objective` Gives the value of the objective function at the current iterate.
- `FeasErr` Gives a measure of the feasibility violation at the current iterate.
- `OptErr` Gives a measure of the violation of the Karush-Kuhn-Tucker (KKT) (first-order) optimality conditions (not including feasibility) at the current iterate.
- ||`Step`|| The 2-norm length of the step (i.e., the distance between the new iterate and the previous iterate).
- `CG its` The number of Projected Conjugate Gradient (CG) iterations required to compute the step.

If `outlev=2`, information is printed every 10 major iterations. If `outlev=3` information is printed at each major iteration. If `outlev>4` addition information is included in the log.

Iteration Information - Discrete Problems:

By default, the GAMS/Knitro link prints a log line at every 10'th node. This frequency can be changed via the mip_outinterval option. To turn off the node log completely, set the mip_outlevel option to 0. The columns of the iteration log for discrete models are as follows:

- `Node` The node number. If an integer feasible point was found at a given node, it is marked with a $*$
- `Left` The current number of active nodes left in the branch and bound tree.
- `Iinf` The current number of active nodes left in the branch and bound tree.
- `Objective` Gives the value of the objective function at the solution of the relaxed subproblem solved at the current node. If the subproblem was infeasible or failed, this is indicated. Additional symbols may be printed at some nodes if the node was pruned (`pr`), integer feasible (`f`), or an integer feasible point was found through rounding (`r`).
- `Best relaxatn` The value of the current best relaxation (lower bound on the solution if minimizing).
- `Best incumbent` The value of the current best integer feasible point (upper bound on the solution if minimizing).

Termination Message: At the end of the run a termination message is printed indicating whether or not the optimal solution was found and if not, why the solver terminated. Below is a list of some possible termination messages.

- `EXIT: Locally optimal solution found.`

  Knitro found a locally optimal point which satisfies the stopping criterion (see section Knitro Termination Test and Optimality) for more detail on

  how this is defined). If the problem is convex (for example, a linear

  program), then this point corresponds to a globally optimal solution.

- `EXIT: Iteration limit reached.`

  The iteration limit was reached before being able to satisfy the required stopping criteria.

- `EXIT: Convergence to an infeasible point. Problem appears to be locally infeasible.`

  The algorithm has converged to an infeasible point from which it cannot further decrease the infeasibility measure. This happens when the problem is infeasible, but may also occur on occasion for feasible problems with nonlinear constraints or badly scaled problems. It is recommended to try various initial points. If this occurs for a variety of initial points, it is likely the problem is infeasible.

- `EXIT: Problem appears to be unbounded.`

  The objective function appears to be decreasing without bound, while satisfying the constraints.

- `EXIT: Current point cannot be improved.`

  No more progress can be made. If the current point is feasible it is likely it may be optimal, however the stopping tests cannot be satisfied perhaps because of degeneracy, ill-conditioning or bad scaling).

- `EXIT: Current point cannot be improved. Point appears to be optimal, but desired accuracy could not be achieved.`

  No more progress can be made, but the stopping tests are close to being satisfied (within a factor of 100) and so the current approximate solution is believed to be optimal.

- `EXIT: Time limit reached.`

  The time limit was reached before being able to satisfy the required stopping criteria.

- `EXIT: Evaluation error.`

  This termination value indicates that an evaluation error occurred (e.g., divide by 0, taking the square root of a negative number), preventing the optimization from continuing.

- `EXIT: Not enough memory available to solve problem.`

  This termination value indicates that there was not enough memory available to solve the problem.

Final Statistics:

Following the termination message some final statistics on the run are printed. Both relative and absolute error values are printed.

Solution Vector/Constraints:

If `outlev=5`, the values of the solution vector are printed after the final statistics. If `outlev=6`, the final constraint values are also printed before the solution vector and the values of the Lagrange multipliers (or dual variables) are printed next to their corresponding constraint or bound.

5.23.8 Algorithm Options

5.23.8.1 Automatic

By default, `Knitro` will automatically try to choose the best optimizer for the given problem based on the problem characteristics.

5.23.8.2 Interior/Direct

If the Hessian of the Lagrangian is ill-conditioned or the problem does not have a large-dense Hessian, it may be advisable to compute a step by directly factoring the KKT (primal-dual) matrix rather than using an iterative approach to solve this system. Knitro offers the Interior/Direct optimizer which allows the algorithm to take direct steps by setting `algorithm=1`. This option will try to take a direct step at each iteration and will only fall back on the iterative step if the direct step is suspected to be of poor quality, or if negative curvature is detected.

Using the Interior/Direct optimizer may result in substantial improvements over Interior/CG when the problem is ill-conditioned (as evidenced by Interior/CG taking a large number of Conjugate Gradient iterations). We encourage the user to try both options as it is difficult to predict in advance which one will be more effective on a given problem. In each case, also experiment with the bar_murule option, as it is difficult to predict which update rule will work best.

NOTE: Since the Interior/Direct algorithm in Knitro requires the explicit storage of a Hessian matrix, this version can only be used with Hessian options, `hessopt=1, 2, 3` or `6`. It may not be used with Hessian options, `hessopt=4` or `5`, which only provide Hessian-vector products. Both the Interior/Direct and Interior/CG methods can be used with the bar_feasible option.

5.23.8.3 Interior/CG

Since Knitro was designed with the idea of solving large problems, the Interior/CG optimizer in Knitro offers an iterative Conjugate Gradient approach to compute the step at each iteration. This approach has proven to be efficient in most cases and allows Knitro to handle problems with large, dense Hessians, since it does not require factorization of the Hessian matrix. The Interior/CG algorithm can be chosen by setting `algorithm=2`. It can use any of the Hessian options as well as the bar_feasible option.

5.23.8.4 Active Set

Knitro includes an active-set Sequential Linear-Quadratic Programing (SLQP) optimizer. This optimizer is particular advantageous when "warm starting" (i.e., when the user can provide a good initial solution estimate, for example, when solving a sequence of closely related problems). This algorithm is also the preferred algorithm for detecting infeasible problems quickly. The Active Set algorithm can be chosen by setting `algorithm=3`. It can use any of the Hessian options.

5.23.9 Other Knitro special features

This section describes in more detail some of the more important special features of Knitro and provides some guidance on how use them so that Knitro runs most efficiently for the problem at hand.

5.23.9.1 Second derivative options

The default version of Knitro assumes that exact second derivatives of the objective function and constraint functions can be computed. If this is possible and the cost of computing the second derivatives is not overly expensive, it is highly recommended to use exact second derivatives. However, Knitro also offers other options which are described in detail below.

(Dense) Quasi-Newton BFGS
The quasi-Newton BFGS option uses gradient information to compute a symmetric, *positive-definite* approximation to the Hessian matrix. Typically this method requires more iterations to converge than the exact Hessian version. However, since it is only computing gradients rather than Hessians, this approach may be more efficient in many cases. This option stores a *dense* quasi-Newton Hessian approximation so it is only recommended for small to medium problems ($n < 1000$). The quasi-Newton BFGS option can be chosen by setting options value `hessopt=2`.

(Dense) Quasi-Newton SR1
As with the BFGS approach, the quasi-Newton SR1 approach builds an approximate Hessian using gradient information. However, unlike the BFGS approximation, the SR1 Hessian approximation is not restricted to be positive-definite. Therefore the quasi-Newton SR1 approximation may be a better approach, compared to the BFGS method, if there is a lot of negative curvature in the problem since it may be able to maintain a better approximation to the true Hessian in this case. The quasi-Newton SR1 approximation maintains a *dense* Hessian approximation and so is only recommended for small to medium problems ($n < 1000$). The quasi-Newton SR1 option can be chosen by setting options value `hessopt=3`.

Finite-difference Hessian-vector product option
If the problem is large and gradient evaluations are not the dominate cost, then Knitro can internally compute Hessian-vector products using finite-differences. Each Hessian-vector product in this case requires one additional gradient evaluation. This option can be chosen by setting options value `hessopt=4`. This option is generally only recommended if the exact gradients are provided.

NOTE: This option may not be used when `algorithm=1`.

Exact Hessian-vector products
In some cases the problem which the user wishes to solve may have a large, dense Hessian which makes it impractical to store or work with the Hessian directly.

The performance of this option should be nearly identical to the exact Hessian option but requires much less storage. This option can be chosen by setting options value `hessopt=5`.

NOTE: This option may not be used when `algorithm=1`.

Limited-memory Quasi-Newton BFGS
The limited-memory quasi-Newton BFGS option is similar to the dense quasi-Newton BFGS option described above. However, it is better suited for large-scale problems since, instead of storing a dense Hessian approximation, it only stores a limited number of gradient vectors used to approximate the Hessian. In general it requires more iterations to converge than the dense quasi-Newton BFGS approach but will be much more efficient on large-scale problems. This option can be chosen by setting options value `hessopt=6`.

5.23.9.2 Feasible version

Knitro offers the user the option of forcing intermediate iterates to stay feasible with respect to the *inequality* constraints (it does not enforce feasibility with respect to the *equality* constraints however). Given an initial point which is *sufficiently* feasible with respect to all inequality constraints and selecting `bar_feasible = 1`, forces all the iterates to strictly satisfy the inequality constraints throughout the solution process. For the feasible mode to become active the iterate x must satisfy

$$cl + tol \leq c(x) \leq cu - tol \tag{21}$$

for *all* inequality constraints (i.e., for $cl \neq cu$). The tolerance $tol > 0$ by which an iterate must be strictly feasible for entering the feasible mode is determined by the parameter `bar_feasmodetol` which is `1.0e-4` by default. If the initial point does not satisfy (21) then the default infeasible version of Knitro will run until it obtains a point which is sufficiently feasible with respect to all the inequality constraints. At this point it will switch to the feasible version of Knitro and all subsequent iterates will be forced to satisfy the inequality constraints.

For a detailed description of the feasible version of Knitro see [32] .

NOTE: This option may only be used when `algorithm=2`.

5.23.9.3 Honor Bounds

By default Knitro does not enforce that the simple bounds on the variables (1c) are satisfied throughout the optimization process. Rather, satisfaction of these bounds is only enforced at the solution. In some applications, however, the user may want to enforce that the initial point and all intermediate iterates satisfy the bounds $bl \leq x \leq bu$. This can be enforced by setting `honorbnds=1`.

5.23.9.4 Crossover

Interior-point (or barrier) methods are a powerful tool for solving large-scale optimization problems. However, one drawback of these methods is that they do not always provide a clear picture of which constraints are active at the solution. In general they return a less exact solution and less exact sensitivity information. For this reason, Knitro offers a crossover feature in which the interior-point method switches to the Active Set method at the interior-point solution estimate, in order to "clean up" the solution and provide more exact sensitivity and active set information. The crossover procedure is controlled by the bar_maxcrossit option. If this option is greater than 0, then Knitro will attempt to perform bar_maxcrossit Active Set crossover iterations after the interior-point method has finished, to see if it can provide a more exact solution. This can be viewed as a form of post-processing. If bar_maxcrossit is not positive, then no crossover iterations are attempted.

The crossover procedure will not always succeed in obtaining a more exact solution compared with the interior-point solution. If crossover is unable to improve the solution within bar_maxcrossit crossover iterations, then it will restore the interior-point solution estimate and terminate. By default, Knitro will then print a message indicating that it was unable to improve the solution within the iterations allowed. In this case, you may want to increase the value of bar_maxcrossit and try again. If Knitro determines that the crossover procedure will not succeed, no matter how many iterations are tried, then a message of the form `Crossover mode unable to improve solution.` will be printed.

The extra cost of performing crossover is problem dependent. In most small or medium scale problems, the crossover cost is a small fraction of the total solve cost. In these cases it may be worth using the crossover procedure to obtain a more exact solution. On some large scale or difficult degenerate problems, however, the cost of performing crossover may be significant. It is recommended to experiment with this option to see whether improvement in the exactness of the solution is worth the additional cost.

5.23.9.5 Tuner

The Knitro-Tuner can help you identify some non-default options settings that may improve performance on a particular model or set of models. The Knitro tuner is enabled with the tuner option and controlled via the tuner_ family of options. If you are unsure about what Knitro options should be tuned to try to improve performance, you can run the default Knitro-Tuner by simply setting the option tuner=1 when solving with Knitro. This will cause Knitro to run your model with a variety of automatically determined option settings, and report some statistics at the end. Any Knitro options that have been set in the usual way will remain fixed throughout the tuning procedure.

If you have some ideas about which Knitro options you want to tune, you can tell Knitro which options you want it to tune, as well as specify the values for particular options that you want Knitro to explore. This can be done by specifying a Tuner options file. A Tuner options file is a simple text file that is similar to a standard Knitro options file, with some important differences:

- You can define multiple values (separated by spaces) for each option. This tells Knitro the values you want it to explore.
- You can specify an option name without any values. This will tell Knitro to explore all possible option values for that option. This only works for options that have a finite set of possible option value settings.
- A Tuner options file is loaded via the tuner_optionsfile option.

All possible combinations of options/values specified in a Tuner options file will be explored by Knitro, while any Knitro options that have been set in the usual way will remain fixed throughout the tuning procedure.

5.23.9.6 Solving Systems of Nonlinear Equations

Knitro is quite effective at solving systems of nonlinear equations. To solve a square system of nonlinear equations using Knitro one should specify the nonlinear equations as equality constraints (i.e., constraints with $cl = cu$), and specify the objective function (1a) as zero (i.e., $f(x) = 0$).

5.23.9.7 Solving Least Squares Problems

There are two ways of using Knitro for solving problems in which the objective function is a sum of squares of the form

$$f(x) = \frac{1}{2}\sum_{j=1}^{q} r_j(x)^2.$$

If the value of the objective function at the solution is not close to zero (the large residual case), the least squares structure of f can be ignored and the problem can be solved as any other optimization problem. Any of the Knitro options can be used.

On the other hand, if the optimal objective function value is expected to be small (small residual case) then Knitro can implement the Gauss-Newton or Levenberg-Marquardt methods which only require first derivatives of the residual functions, $r_j(x)$, and yet converge rapidly. To do so, the user need only define the Hessian of f to be

$$\nabla^2 f(x) = J(x)^T J(x),$$

where

$$J(x) = \left[\frac{\partial r_j}{\partial x_i}\right] \begin{array}{l} j = 1, 2, \ldots, q \\ i = 1, 2, \ldots, n \end{array}.$$

The actual Hessian is given by

$$\nabla^2 f(x) = J(x)^T J(x) + \sum_{j=1}^{q} r_j(x)\nabla^2 r_j(x);$$

the Gauss-Newton and Levenberg-Marquardt approaches consist of ignoring the last term in the Hessian.

Knitro will behave like a Gauss-Newton method by setting `algorithm=1`, and will be very similar to the classical Levenberg-Marquardt method when `algorithm=2`. For a discussion of these methods see, for example, [143] .

5.24 LINDO and LINDOGlobal

Lindo Systems, Inc.

5.24.1 Introduction

GAMS/LINDO finds guaranteed globally optimal solutions to general nonlinear problems with continuous and/or discrete variables. GAMS/LINDO supports most mathematical functions, including functions that are nonsmooth, such as abs(x) and or even discontinuous, such as floor(x). Nonlinear solvers employing methods like successive linear programming (SLP) or generalized reduced gradient (GRG) return a local optimal solution to an NLP problem. However, many practical nonlinear models are non-convex and have more than one local optimal solution. In some applications, the user may want to find a global optimal solution.

The LINDO global optimization procedure(GOP) employs branch-and-cut methods to break an NLP model down into a list of subproblems. Each subproblem is analyzed and either a) is shown to not have

a feasible or optimal solution, or b) an optimal solution to the subproblem is found, e.g., because the subproblem is shown to be convex, or c) the subproblem is further split into two or more subproblems which are then placed on the list. Given appropriate tolerances, after a finite, though possibly large number of steps a solution provably global optimal to tolerances is returned. Traditional nonlinear solvers can get stuck at suboptimal, local solutions. This is no longer the case when using the global solver.

GAMS/LINDO can automatically linearize a number of nonlinear relationships, such as max(x,y), through the addition of constraints and integer variables, so the transformed linearized model is mathematically equivalent to the original nonlinear model. Keep in mind, however, that each of these strategies will require additional computation time. Thus, formulating models, so they are convex and contain a single extremum, is desirable. In order to decrease required computing power and time it is also possible to disable the global solver and use GAMS/LINDO like a regular nonlinear solver.

GAMS/LINDO has a multistart feature that restarts the standard (non-global) nonlinear solver from a number of intelligently generated points. This allows the solver to find a number of locally optimal points and report the best one found. This alternative can be used when global optimization is costly. A user adjustable parameter controls the maximum number of multistarts to be performed.

LINDO automatically detects problem type and uses an appropriate solver, e.g., if you submit an LP model to LINDO, it will be solved as an LP at LP speed, regardless of what you said in the "solve using" statement. With the NLP parameter *NLP_QUADCHK* turned on, LINDO can detect hidden quadratic expressions and automatically recognize convex QCPs, as well as second-order cones (SOCP), like in Value-at-Risk models, allowing dramatically faster solution times via the barrier solver. When such models have integer variables, LINDO would use the barrier solver to solve all subproblems leading to significantly improved solution times when compared to the case with the standard NLP solver.

5.24.1.1 Licensing and software requirements

In order to use GAMS/LINDOGlobal, two licenses are required: a GAMS/LINDOGlobal license and a GAMS/CONOPT license. The additional CONOPT license requirement exists because LINDOGlobal uses CONOPT to solve the nonlinear subproblems. The GAMS/LINDOGlobal license places upper limits on the model size of 3,000 variables and 2,000 constraints.

To use GAMS/LINDO, only a GAMS/LINDO license is required. It imposes no upper limit on the model size and includes the capability to solve stochastic models (see section Stochastic Programming (SP) in GAMS/Lindo).

Neither the GAMS/LINDO nor the GAMS/LINDOGlobal license includes the Barrier solver option. The Barrier option is enabled via a separate license for the GAMS/MOSEK barrier solver.

5.24.1.2 Running GAMS/LINDO

GAMS/LINDO is capable of solving models of the following types: EMP (stochastic), LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP and MINLP. If GAMS/LINDO is not specified as the default solver for these models, it can be invoked by issuing one of the following command before the solve statement:

```
option xxx=lindo;
option xxx=lindoglobal;
```

where xxx is one of: EMP, LP, MIP, RMIP, NLP, DNLP, QCP, MIQCP, RMINLP, or MINLP.

You can also find global optima to math programs with equilibrium or complementarity constraints, type MPEC, by using the GAMS/NLPEC translator in conjunction with LINDO. You use NLPEC to translate complementarities into standard mathematical statements, e.g. $h*y = 0$, and then use LINDO as the DNLP(Discontinuous Nonlinear) solver to solve the translated model. The following little GAMS model illustrates:

```
$TITLE simple mpec example
variable f, x1, x2, y1, y2; positive
variable y1; y2.lo = -1; y2.up = 1;

equations cost, g, h1, h2;

cost..  f =E= x1 + x2;
   g..  sqr(x1) + sqr(x2) =L= 1;
  h1..  x1 =G= y1 - y2 + 1;
  h2..  x2 + y2 =N= 0;

* declare h and y complementary
model example / cost, g, h1.y1, h2.y2 /;

option mpec=nlpec;
option dnlp=lindo;
solve example using mpec min f;
```

5.24.2 Supported nonlinear functions

GAMS/LINDO supports most nonlinear functions in global mode, including +, -, ∗, /, floor, modulo, sign, min, max, sqr, exp, power, ln, log, sqrt, abs, cos, sin, tan, cosh, sinh, tanh, arccos, arcsin, arctan and logic expressions AND, OR, NOT, and IF. Be aware that using highly nonconvex functions may lead to long solve times.

5.24.3 Diagnosis of Infeasible or Unbounded Models

GAMS/LINDO offers two diagnostic tools, that can help users debug infeasible or unbounded optimization models. These tools can be called after the solver reports an infeasible or unbounded status for the model. When activating IIS Lindo finds an irreducible infeasible set (IIS) of constraints, whereas setting IUS, makes Lindo find an irreducible unbounded set (IUS) of variables. An IIS is a set of constraints that are infeasible taken together, but every strict subset is feasible. Similarly, an IUS is a set of unbounded variables such that fixing any one of them would make the model bounded. The IIS or IUS portion of the model will generally be much smaller than the original model. Thus, the user can track down formulation or data entry errors quickly. By isolating the source of infeasibility or unboundedness, the user can correct the model data such as right-hand side values, objective coefficients, senses of the constraints, and column bounds. Note that the IUS option is available for LPs only.

5.24.3.1 Infeasible Models

GAMS/Lindo's IIS option activates the IIS finder, after a model was tried to be solved and the solver returned a "no feasible solution" message. For an LP, if an infeasible basis is not resident in the solver, the IIS finder cannot initiate the process to isolate an IIS. This can occur if the infeasibility is detected in the pre-solver before a basis is created, or the barrier solver has terminated without performing a basis crossover. To obtain an IIS for such cases, the pre-solve option will be turned off automatically and the model gets optimized again.

The constraints and bounds in the IIS are further classified into two disjoint sets: a necessary set and a sufficient set. The *sufficient* set refers to a crucial subset of the IIS in the sense that removing any one of its members from the entire model renders the model feasible. Note that not all infeasible models have *sufficient* sets. The *necessary* set contains those constraints and bounds that are likely to contribute to the overall infeasibility of the entire model. Thus, the *necessary* set requires a correction in at least one member to make the original model feasible. A constraint that has been marked as *sufficient* has a high probability of containing an error. In fact, if the model contains only one bad coefficient, the constraint containing it will be marked as *sufficient*.

To control the level of analysis when locating an IIS, one can set the option IIS_ANALYZE_LEVEL.

5.24.3.2 Unbounded Linear Programs

GAMS/Lindo's IUS option is similar to the IIS option, except that it is used to track down the source of an unbounded solution in a linear program. This tool analyzes the model and isolates an "irreducibly unbounded set" (IUS) of variables. As in the infeasibility case, the IUS is partitioned into *sufficient* and *necessary* sets to indicate the role of the member variables for the unboundedness of the overall model.

The variables in the *sufficient* set are crucial in the sense that fixing any of these variables makes the overall model bounded. However, fixing the variables in the *necessary* set does not ensure that there are no other sets of unbounded variables that cause unboundedness for the overall model.

To control the level of analysis when locating an IUS, one can set the option IUS_ANALYZE_LEVEL.

5.24.4 GAMS/LINDO output

The log output below is obtained for the NLP model mhw4d.gms from the GAMS model library using LINDOs global solver.

```
LINDO                 24Nov11 23.8.0 WIN 30200.30202 VS8 x86/MS Windows

   LINDO Driver
   Lindo Systems Inc, www.lindo.com

Lindo API version 7.0.1.372 built on Nov  3 2011 21:49:01
Barrier Solver Version 6.0.0.114, Nonlinear Solver Version 3.15B
Platform Windows x86

Number of constraints:       3    le:       0, ge:       0, eq:       3, rn:
   0 (ne:0)
Number of variables  :       5    lb:       0, ub:       0, fr:       5, bx:
   0 (fx:0)
Number of nonzeroes  :       8    density=0.0053(%)

Nonlinear variables  :       5
Nonlinear constraints:       4
Nonlinear nonzeroes  :       5+5

Starting global optimization ...

Number of nonlinear functions/operators:  3
 EP_MULTIPLY  EP_POWER  EP_SQR

Starting GOP presolve ...
First Call Local Solver
Find local solution, objvalue =    27.871905

Pre-check unboundedness
Computing reduced bound...
Searching for a better solution...

Starting reformulation ...

Model                    Input     Operation    Atomic       Convex

Number of variables  :       5             6        20           20
```

```
Number of constraints:       3            4        18           46
integer variables    :       0            0         0            0
nonlinear variables  :       5            5         9            0

Starting global search ...
 Initial upper bound on objective: +2.931083e-002
 Initial lower bound on objective: -3.167052e+022


 #NODEs  BOXES   LOWER BOUND     UPPER BOUND      RGAP   TIME(s)

     1       1  -3.167052e+022  +2.931083e-002  1.0e+000      0 (*N)
    19      17  -2.136461e+000  +2.931083e-002  1.0e+000      0 (*I)
    22      20  -1.848574e-001  +2.931083e-002  2.1e-001      0 (*I)
    23      21  +2.416053e-003  +2.931083e-002  2.7e-002      0 (*F)


Terminating global search ...

 Global optimum found
 Objective value              :        0.0293108307216
 Best Bound                   :       0.00241605257558
 Factors (ok,stb)             :                    522 (100.00,99.81)
 Simplex iterations           :                   2503
 Barrier iterations           :                      0
 Nonlinear iterations         :                    433
 Box iterations               :                     23
 Total number of boxes        :                     21
 Max. Depth                   :                      5
 First solution time (sec.)   :                      0
 Best solution time (sec.)    :                      0
 Total time (sec.)            :                      0
```

After determining the different kinds of nonlinear operators LINDO tries to linearize these within the presolving. When a feasible starting point is found the optimization starts and the log provides information about the progress. At the end it is reported if an optimum could be found and then the results as well as the used resources are summarized.

The following flags can be seen in the progress log:

Flag	Description
(∗**FP**)	found a new MIP solution with feasibility pump
(∗**SBB**)	found a new MIP solution in tree reorder
(∗**SE**)	found a new MIP solution in simple enumeration
(∗**AB**)	found a new MIP solution in advanced branching
(∗**AH**)	found a new MIP solution with advanced heuristics
(∗**C**)	found a new MIP solution after cuts added
(∗**T**)	found a new MIP solution on the top
(∗**SRH**)	found a new MIP solution in simple rounding heuristics
(∗**SB**)	found a new MIP solution in strong branching
(∗**K**)	found a new MIP solution in knapsack enumerator
(∗)	found a new MIP solution normal branching
(∗**?-**)	found a new MIP solution with advanced heuristics (level$>$10)
(∗**N**)	found a new incumbent GOP solution

Flag	Description
(∗**I**)	stored a box with the incumbent solution into the GOP solution list
(∗**F**)	determined the final GOP status

5.24.5 The GAMS/LINDO Options

GAMS/LINDO offers a diverse range of user-adjustable parameters to control the behavior of its solvers. While the default values of these parameters work best for most purposes, there may be cases the users prefer to work with different settings for a subset of the available parameters. This section gives a list of available GAMS/LINDO parameters, categorized by type, along with their brief descriptions. A more detailed description is given in the section that follows.

5.24.5.1 GAMS/LINDO Options File

In order to set GAMS/LINDO options, you need to set up an option file *lindo.opt* or *lindoglobal.opt* in your GAMS project directory. You must indicate in the model that you want to use the option file by inserting before the solve statement, the line:

```
<modelname>.optfile = 1;
```

where

```
<modelname>
```

is the name of the model referenced in the model statement. The option file is in plain text format containing a single GAMS/LINDO option per line. Each option identifier is followed by its target value with space or tab characters separating them. The lines starting with ∗ character are treated as comments.

A sample option file *lindo.opt* looks like below

```
* Use(1) or Disable(0) global optimization for NLP/MINLP models
USEGOP               0

* Enable Multistart NLP solver
NLP_SOLVER           9

* Allow a maximum of 3 multistart attempts
NLP_MAXLOCALSEARCH   3

* Set an overall time limit of 200 secs.
SOLVER_TIMLMT      200
```

5.24.6 Summary of GAMS/Lindo Options

5.24.6.1 General Options

Option	Description	Default
DECOMPOSITION_TYPE	decomposition to be performed on a linear or mixed integer model	1
FIND_BLOCK	graph partitioning method to find block structures	0
FIND_SYMMETRY_LEVEL	specifies the symmetry finding level.	-1
FIND_SYMMETRY_PRINT_LEVEL	specifies print level for symmetry finding	0
INSTRUCT_SUBOUT	flag to specify how to deal with fixed variables in the instruction list	-1
MULTITHREAD_MODE	threading mode	-1
NUM_THREADS	number of parallel threads to be used	GAMS Threads
PROFILER_LEVEL	specifies the profiler level to break down the total cpu time into.	0
SOLVER_CONCURRENT_OPTMODE	controls if simplex and interior-point optimizers will run concurrently	0
SOLVER_CUTOFFVAL	solver will exit if optimal solution is worse than this	0
SOLVER_FEASTOL	feasibility tolerance	1e-7
SOLVER_IPMSOL	basis crossover flag for barrier solver	0
SOLVER_IUSOL	flag for computing basic solution for infeasible model	0
SOLVER_METHOD	specifies the method to use when generic solver is invoked	0
SOLVER_MODE	controls some of the advanced strategies when solving LPs	1
SOLVER_OPTTOL	dual feasibility tolerance	1e-7
SOLVER_PRE_ELIM_FILL	fill-in introduced by the eliminations during pre-solve	1000
SOLVER_RESTART	starting basis flag	0
SOLVER_TIMLMT	time limit in seconds for continous solver	GAMS ResLim
SOLVER_USECUTOFFVAL	flag for using cutoff value	0
TUNER_PRINT_LEVEL	specifies the amount of print to do during parameter tuning	1

5.24.6.2 LP Options

Option	Description	Default
LP_AIJ_ZEROTOL	coefficient matrix zero tolerance	2.22045e-16
LP_BIGM	big-M for phase-I	1e6
LP_BNDINF	big-M to truncate lower and upper bounds in single phase dual-simplex	1e15
LP_DPSWITCH	specifies whether LP primal-dual simplex switch is enabled or not	1
LP_DRATIO	controls the dual min-ratio strategy	1
LP_DYNOBJFACT	Dynamic obj factor	0.75

Option	Description	Default
LP_DYNOBJMODE	Dynamic obj mode	0
LP_ITRLMT	simplex iteration limit	infinity
LP_PIV_BIGTOL	simplex maximum pivot tolerance	1e-5
LP_PIV_ZEROTOL	simplex pivot zero tolerance	1e-8
LP_PPARTIAL	primal simplex partial pricing method	0
LP_PRELEVEL	controls the amount and type of LP pre-solving	126
LP_RATRANGE	controls the number of pivot-candidates to consider when searching for a stable pivot in LU decomposition	4
LP_SCALE	scaling flag	1
LP_SPRINT_COLFACT	maximum number of columns in Sprint as a factor of number of rows	10
LP_SPRINT_MAXPASS	maximum number of passes in Sprint method	100
LP_SPRINT_SUB	LP method for subproblem in Sprint method	0
PROB_TO_SOLVE	controls whether the explicit primal or dual form of the given LP problem will be solved	0
SPLEX_DPRICING	pricing option for dual simplex method	-1
SPLEX_DUAL_PHASE	controls the dual simplex strategy	0
SPLEX_PPRICING	pricing option for primal simplex method	-1
SPLEX_REFACFRQ	number of simplex iterations between two consecutive basis re-factorizations	100

5.24.6.3 IPM Options

Option	Description	Default
IPM_BASIS_REL_TOL_S	maximum relative dual bound violation allowed in an optimal basic solution	1e-12
IPM_BASIS_TOL_S	maximum absolute dual bound violation in an optimal basic solution	1e-7
IPM_BASIS_TOL_X	maximum absolute primal bound violation allowed in an optimal basic solution	1e-7
IPM_BI_LU_TOL_REL_PIV	relative pivot tolerance used in the LU factorization in the basis identification procedure	1e-2
IPM_CHECK_CONVEXITY	flag to check convexity of a quadratic program using barrier solver	1
IPM_CO_TOL_DFEAS	dual feasibility tolerance for Conic solver	1e-8
IPM_CO_TOL_INFEAS	maximum bound infeasibility tolerance for Conic solver	1e-12
IPM_CO_TOL_MU_RED	optimality tolerance for Conic solver	1e-8
IPM_CO_TOL_PFEAS	primal feasibility tolerance for Conic solver	1e-8
IPM_MAX_ITERATIONS	ipm iteration limit	1000
IPM_NUM_THREADS	number of threads to run the interiorpoint optimizer on	1
IPM_OFF_COL_TRH	extent for detecting the offending columns in the Jacobian of the constraint matrix	40
IPM_TOL_DFEAS	dual feasibility tolerance	1e-8

Option	Description	Default
IPM_TOL_DSAFE	controls the initial dual starting point	1
IPM_TOL_INFEAS	infeasibility tolerance	1e-10
IPM_TOL_MU_RED	relative complementarity gap tolerance	1e-16
IPM_TOL_PATH	how close to follow the central path	1e-8
IPM_TOL_PFEAS	primal feasibility tolerance	1e-8
IPM_TOL_PSAFE	controls the initial primal starting point	1
IPM_TOL_REL_STEP	relative step size to the boundary	0.9999

5.24.6.4 MIP Options

Option	Description	Default
MIP_ABSCUTTOL	MIP absolute cut tolerance	-1.0
MIP_ABSOPTTOL	MIP absolute optimality tolerance	GAMS OptCA
MIP_ADDCUTOBJTOL	required objective improvement to continue generating cuts	1.5625e-5
MIP_ADDCUTPER	percentage of constraint cuts that can be added	0.75
MIP_ADDCUTPER_TREE	percentage of constraint cuts that can be added at child nodes	0.5
MIP_AGGCUTLIM_TOP	max number of constraints involved in derivation of aggregation cut at root node	-1
MIP_AGGCUTLIM_TREE	max number of constraints involved in derivation of aggregation cut at tree nodes	-1
MIP_ANODES_SWITCH_DF	threshold on active nodes for switching to depth-first search	50000
MIP_AOPTTIMLIM	time in seconds beyond which the relative optimality tolerance will be applied	100
MIP_BIGM_FOR_INTTOL	threshold for which coefficient of a binary variable would be considered as big-M	1e8
MIP_BRANCHDIR	first branching direction	0
MIP_BRANCHRULE	rule for choosing the variable to branch	0
MIP_BRANCH_LIMIT	limit on the total number of branches to be created during branch and bound	-1
MIP_BRANCH_PRIO	controls how variable selection priorities are set and used	0
MIP_CONCURRENT_REOPTMODE	specifies the concurrent optimization mode with warm start	0
MIP_CONCURRENT_STRATEGY	controls the concurrent MIP strategy	-1
MIP_CONCURRENT_TOPOPTMODE	specifies the concurrent optimization mode with cold start	0
MIP_CUTDEPTH	threshold value for the depth of nodes in the branch and bound tree	8
MIP_CUTFREQ	frequency of invoking cut generation at child nodes	10
MIP_CUTLEVEL_TOP	combination of cut types to try at the root node when solving a MIP	57342

Option	Description	Default
MIP_CUTLEVEL_TREE	combination of cut types to try at child nodes in the branch and bound tree when solving a MIP	53246
MIP_CUTOFFOBJ	defines limit for branch and bound	1e30
MIP_CUTTIMLIM	time to be spent in cut generation	-1
MIP_DELTA	near-zero value used in linearizing nonlinear expressions	1e-6
MIP_DUAL_SOLUTION	flag for computing dual solution of LP relaxation	0
MIP_FP_HEU_MODE	specifies the feasibility-pump (FP) heuristic mode	0
MIP_FP_ITRLIM	iteration limit for feasibility pump heuristic	500
MIP_FP_MODE	mode for the feasibility pump heuristic	-1
MIP_FP_OPT_METHOD	optimization and reoptimization method for feasibility pump heuristic	0
MIP_FP_PROJECTION	type of objective function of LPs in projection step of the feasibility pump heuristic	0
MIP_FP_TIMLIM	time limit for feasibility pump heuristic	1800
MIP_FP_WEIGTH	weight of the objective function in the feasibility pump	1
MIP_GENERAL_MODE	general strategy in solving MIPs	0
MIP_HEULEVEL	specifies heuristic used to find integer solution	3
MIP_HEUMINTIMLIM	minimum time in seconds to be spent in finding heuristic solutions	0
MIP_HEU_DROP_OBJ	flag for whether to use without OBJ heuristic	0
MIP_HEU_MODE	heuristic used in MIP solver	0
MIP_INTTOL	absolute integer feasibility tolerance	1e-6
MIP_ITRLIM	iteration limit for branch and bound	infinity
MIP_KBEST_USE_GOP	specifies whether to use gop solver in MIP KBest	0
MIP_KEEPINMEM	flag for keeping LP bases in memory	1
MIP_LBIGM	Big-M value used in linearizing nonlinear expressions	10000
MIP_LSOLTIMLIM	time limit until finding a new integer solution	-1
MIP_MAKECUT_INACTIVE_COUNT	threshold for times a cut could remain active after successive reoptimization	10
MIP_MAXCUTPASS_TOP	number passes to generate cuts on the root node	200
MIP_MAXCUTPASS_TREE	number passes to generate cuts on the child nodes	2
MIP_MAXNONIMP_CUTPASS	number of passes allowed in cut-generation that does not improve current relaxation	3
MIP_MAXNUM_MIP_SOL_STORAGE	maximum number of k-best solutions to store	1

Option	Description	Default
MIP_MINABSOBJSTEP	value to update cutoff value each time a mixed integer solution is found	0
MIP_NODESELRULE	specifies the node selection rule	0
MIP_NUM_THREADS	number of parallel threads to use by the parallel MIP solver	1
MIP_PARA_FP	flag for whether to use parallelization on the feasibility pump heuristic	1
MIP_PARA_FP_MODE	flag for the mode of parallel feasibility pump	0
MIP_PARA_INIT_NODE	number of initial nodes for MIP parallelization	-1
MIP_PARA_ITR_MODE	flag for iteration mode in MIP parallelization	1
MIP_PARA_RND_ITRLMT	iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations	2.0
MIP_PARA_SUB	flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing	1
MIP_PEROPTTOL	MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds	1e-5
MIP_PERSPECTIVE_REFORM	flag for whether to use Perspective Reformulation	1
MIP_POLISH_ALPHA_TARGET	proportion solutions in the pool to initiate a polishing-task at the current node	0.6
MIP_POLISH_MAX_BRANCH_COUNT	maximum number of branches to polish	2000
MIP_POLISH_NUM_BRANCH_NEXT	number of branches to polish in the next round	4000
MIP_PREHEU_DFE_VSTLIM	limit for the variable visit in depth first enumeration	200
MIP_PREHEU_LEVEL	heuristic level for the prerelax solver	0
MIP_PREHEU_TC_ITERLIM	iteration limit for the two change heuristic	30000000
MIP_PREHEU_VAR_SEQ	sequence of the variable considered by the prerelax heuristic	-1
MIP_PRELEVEL	controls the amount and type of MIP presolving at root node	3070
MIP_PRELEVEL_TREE	amount and type of MIP pre-solving at tree nodes	1214
MIP_PRE_ELIM_FILL	controls fill-in introduced by eliminations during pre-solve	100
MIP_PSEUDOCOST_RULE	specifies the rule in pseudocost computations for variable selection	0
MIP_PSEUDOCOST_WEIGT	weight in pseudocost computations for variable selection	1.5625e-05
MIP_REDCOSTFIX_CUTOFF	cutoff value as a percentage of the reduced costs	0.9
MIP_REDCOSTFIX_CUTOFF_TREE	cutoff value as a percentage of the reduced costs at tree nodes	0.9
MIP_RELINTTOL	relative integer feasibility tolerance	8e-6
MIP_RELOPTTOL	MIP relative optimality tolerance	GAMS OptCR

Option	Description	Default
MIP_REOPT	optimization method to use when doing reoptimization	0
MIP_SCALING_BOUND	maximum difference between bounds of an integer variable for enabling scaling	10000
MIP_SOLLIM	integer solution limit for MIP solver	-1
MIP_SOLVERTYPE	optimization method to use when solving mixed-integer models	0
MIP_STRONGBRANCHDONUM	minimum number of variables to try the strong branching on	3
MIP_STRONGBRANCHLEVEL	depth from the root in which strong branching is used	10
MIP_SWITCHFAC_SIM_IPM_ITER	specifies the (positive) factor that multiplies the number of constraints to impose an iteration limit to simplex method and trigger a switch over to the barrier method	-1
MIP_SWITCHFAC_SIM_IPM_TIME	factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method	-1
MIP_SYMMETRY_MODE	specifies mip symmetry handling methods	0
MIP_SYMMETRY_NONZ	limit on number of nonzeros to look for symmetries	50000
MIP_TIMLIM	time limit in seconds for integer solver	GAMS ResLim
MIP_TOPOPT	optimization method to use when there is no previous basis	0
MIP_TREEREORDERLEVEL	tree reordering level	10
MIP_TREEREORDERMODE	tree reordering mode	1
MIP_USECUTOFFOBJ	flag for using branch and bound limit	1
MIP_USE_CUTS_HEU	controls if cut generation is enabled during MIP heuristics	-1
MIP_USE_ENUM_HEU	frequency of enumeration heuristic	4
MIP_USE_INT_ZERO_TOL	controls if all MIP calculations would be based on absolute integer feasibility tolarance	0

5.24.6.5 NLP Options

Option	Description	Default
NLP_AUTODERIV	defining type of computing derivatives	0
NLP_AUTOHESS	flag for using Second Order Automatic Differentiation for solving NLP	0
NLP_CONIC_REFORM	determines if to explore conic reformulation	1
NLP_CONOPT_VER	specifies the CONOPT version to be used in NLP optimizations	3
NLP_CUTOFFOBJ	as soon as any multi-start thread achieves this value all threads stop	-1e30
NLP_DERIV_DIFFTYPE	flag indicating the technique used in computing derivatives with finite differences	0

Option	Description	Default
NLP_FEASCHK	how to report results when solution satisfies tolerance of scaled but not original model	1
NLP_FEASTOL	feasibility tolerance for nonlinear constraints	1e-6
NLP_INF	numeric infinity for nonlinear models	1e30
NLP_IPM2GRG	switch from IPM solver to GRG solver when IPM fails due to numerical errors	1
NLP_ITERS_PER_LOGLINE	number of nonlinear iterations to elapse before next progress message	10
NLP_ITRLMT	nonlinear iteration limit	GAMS IterLim
NLP_LINEARZ	extent to which the solver will attempt to linearize nonlinear models	0
NLP_LINEARZ_WB_CONSISTENT	determines if linearization process is consistent with WB/excel calculation	0
NLP_MAXLOCALSEARCH	maximum number of local searches	5
NLP_MAXLOCALSEARCH_TREE	maximum number of multistarts	1
NLP_MAX_RETRY	maximum number refinement retries to purify the final NLP solution	5
NLP_MSW_EUCDIST_THRES	euclidean distance threshold in multistart search	0.001
NLP_MSW_FILTMODE	filtering mode to exclude certain domains during sampling in multistart search	-1
NLP_MSW_MAXNOIMP	maximum number of consecutive populations to generate without any improvements	-1
NLP_MSW_MAXPOP	maximum number of populations to generate in multistart search	-1
NLP_MSW_MAXREF	maximum number of reference points to generate trial points in multistart search	-1
NLP_MSW_NORM	norm to measure the distance between two points in multistart search	2
NLP_MSW_NUM_THREADS	number of parallel threads to be used when solving an NLP model with the multistart solver	1
NLP_MSW_OVERLAP_RATIO	rate of replacement in successive populations	0.1
NLP_MSW_POXDIST_THRES	penalty function neighborhood threshold in multistart search	0.01
NLP_MSW_PREPMODE	preprocessing strategies in multistart solver	-1
NLP_MSW_RG_SEED	random number generator seed for the multistart solver	1019
NLP_MSW_RMAPMODE	specifies the mode to map reference points in the unit cube into the original space	-1
NLP_MSW_SOLIDX	index of the multistart solution to be loaded	0
NLP_MSW_XKKTRAD_FACTOR	KKT solution neighborhood factor in multistart search	0.85
NLP_MSW_XNULRAD_FACTOR	initial solution neighborhood factor in multistart search	0.5
NLP_PRELEVEL	controls the amount and type of NLP presolving	126
NLP_PSTEP_FINITEDIFF	value of the step length in computing the derivatives using finite differences	5e-7

Option	Description	Default
NLP_QUADCHK	flag for checking if NLP is quadratic	1
NLP_REDGTOL	tolerance for the gradients of nonlinear functions	1e-7
NLP_SOLVER	type of nonlinear solver	7
NLP_SOLVE_AS_LP	flag indicating if the nonlinear model will be solved as an LP	0
NLP_STALL_ITRLMT	iteration limit before a sequence of non-improving NLP iterations is declared as stalling	100
NLP_STARTPOINT	flag for using initial starting solution for NLP	1
NLP_SUBSOLVER	type of nonlinear subsolver	1
NLP_USECUTOFFOBJ	flag to use parameter NLP_CUTOFFOBJ	0
NLP_USE_CRASH	flag for using simple crash routines for initial solution	0
NLP_USE_LINDO_CRASH	flag for using advanced crash routines for initial solution	1
NLP_USE_SDP	flag to use SDP solver for POSD constraint	1
NLP_USE_SELCONEVAL	flag for using selective constraint evaluations for solving NLP	1
NLP_USE_SLP	flag for using sequential linear programming step directions for updating solution	1
NLP_USE_STEEPEDGE	flag for using steepest edge directions for updating solution	0

5.24.6.6 Global Options

Option	Description	Default
GOP_ABSOPTTOL	absolute optimality tolerance	GAMS OptCA
GOP_ALGREFORMMD	algebraic reformulation rule for a GOP	18
GOP_BBSRCHMD	node selection rule in GOP branch-and-bound	1
GOP_BNDLIM	max magnitude of variable bounds used in GOP convexification	1e10
GOP_BOXTOL	minimal width of variable intervals	1e-6
GOP_BRANCHMD	direction to branch first when branching on a variable	5
GOP_BRANCH_LIMIT	limit on the total number of branches to be created in GOP tree	-1
GOP_CMINLP	flag indicating if GOP exploits convex MINLP model	0
GOP_CONIC_REFORM	flag indicating if GOP explore conic reformulation	1
GOP_CORELEVEL	strategy of GOP branch-and-bound	14
GOP_DECOMPPTMD	decomposition point selection rule in GOP branch-and-bound	1
GOP_DELTATOL	delta tolerance in GOP convexification	1e-7
GOP_FLTTOL	floating-point tolerance	1e-10
GOP_HEU_MODE	heuristic used in global solver	0

Option	Description	Default
GOP_ITRLIM	GOP iteration limit	infinity
GOP_ITRLIM_IPM	total barrier iteration limit summed over all branches in GOP	-1
GOP_ITRLIM_NLP	total nonlinear iteration limit summed over all branches in GOP	-1
GOP_ITRLIM_SIM	total simplex iteration limit summed over all branches in GOP	-1
GOP_LIM_MODE	flag indicating which heuristic limit on sub-solver in GOP is based	1
GOP_LINEARZ	flag indicating if GOP exploits linearizable model	1
GOP_LSOLBRANLIM	branch limit until finding a new nonlinear solution	-1
GOP_MAXWIDMD	maximum width flag for the global solution	0
GOP_MULTILINEAR	flag indicating if GOP exploits multi linear feature	1
GOP_NUM_THREADS	number of parallel threads to be used when solving a nonlinear model with the global optimization solver	1
GOP_OBJ_THRESHOLD	threshold of objective value in the GOP solver	-1e+30
GOP_OPTCHKMD	criterion used to certify the global optimality	2
GOP_OPT_MODE	mode for GOP optimization	1
GOP_POSTLEVEL	amount and type of GOP postsolving	6
GOP_PRELEVEL	amount and type of GOP presolving	30
GOP_QUADMD	flag indicating if GOP exploits quadratic feature	-1
GOP_QUAD_METHOD	specifies if the GOP solver should solve the model as a QP when applicable	-1
GOP_RELBRNDMD	reliable rounding in the GOP branch-and-bound	0
GOP_RELOPTTOL	relative optimality tolerance	GAMS OptCR
GOP_SOLLIM	integer solution limit for GOP branch-and-bound	-1
GOP_SUBOUT_MODE	substituting out fixed variables	1
GOP_TIMLIM	time limit in seconds for GOP branch-and-bound	GAMS ResLim
GOP_USEBNDLIM	max magnitude of variable bounds flag for GOP convexification	2
GOP_WIDTOL	maximal width of variable intervals	1e-4
USEGOP	use global optimization	1

5.24.6.7 SP Options

Option	Description	Default
CORE_ORDER_BY_STAGE	order nontemporal models or not	1
EMPINFOFILE	Path and name of file containing additional EMP-SP information as randvar, jrandvar, stage etc.	
REPORTEVSOL	solve and report the expected value solution	0
SAMP_CDSINC	correlation matrix diagonal shift increment	1e-6

Option	Description	Default
SAMP_NCM_CUTOBJ	objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem	1e-30
SAMP_NCM_DSTORAGE	flag to enable or disable sparse mode in NCM computations	-1
SAMP_NCM_ITERLIM	iteration limit for NCM method	100
SAMP_NCM_METHOD	bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem	5
SAMP_NCM_OPTTOL	optimality tolerance for NCM method	1e-7
SAMP_NUM_THREADS	specifies the number of parallel threads to be used when sampling	0
SAMP_RG_BUFFER_SIZE	specifies the buffer size for random number generators in running in parallel mode	0
SAMP_SCALE	flag to enable scaling of raw sample data	0
STOC_ABSOPTTOL	absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver	GAMS OptCA
STOC_ADD_MPI	flag to use add-instructions mode when building deteq	0
STOC_ALD_DUAL_FEASTOL	dual feasibility tolerance for ALD	1e-4
STOC_ALD_DUAL_STEPLEN	dual step length for ALD	0.9
STOC_ALD_INNER_ITER_LIM	inner loop iteration limit for ALD	1000
STOC_ALD_OUTER_ITER_LIM	outer loop iteration limit for ALD	200
STOC_ALD_PRIMAL_FEASTOL	primal feasibility tolerance for ALD	1e-4
STOC_ALD_PRIMAL_STEPLEN	primal step length for ALD	0.5
STOC_AUTOAGGR	flag to enable or disable autoaggregation	1
STOC_BENCHMARK_SCEN	benchmark scenario to compare EVPI and EVMU against	-2
STOC_BIGM	big-M value for linearization and penalty functions	1e7
STOC_BUCKET_SIZE	bucket size in Benders decomposition	-1
STOC_CALC_EVPI	flag to enable or disable calculation of EVPI	1
STOC_CORRELATION_TYPE	correlation type associated with correlation matrix	0
STOC_DEQOPT	method to solve the DETEQ problem	0
STOC_DETEQ_TYPE	type of deterministic equivalent	-1
STOC_DS_SUBFORM	subproblem formulation to use in DirectSearch	-1
STOC_ELIM_FXVAR	flag to enable elimination of fixed variables from deteq MPI	1
STOC_INFBND	value to truncate infinite bounds at non-leaf nodes	1e9
STOC_ITER_LIM	iteration limit for stochastic solver	infinity
STOC_MAP_MPI2LP	flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements	0

Option	Description	Default
STOC_MAX_NUMSCENS	maximum number of scenarios before forcing automatic sampling	40000
STOC_METHOD	stochastic optimization method to solve the model	-1
STOC_NAMEDATA_LEVEL	name data level	1
STOC_NODELP_PRELEVEL	presolve level solving node-models	0
STOC_NSAMPLE_PER_STAGE	list of sample sizes per stage (starting at stage 2)	
STOC_NSAMPLE_SPAR	common sample size per stochastic parameter	-1
STOC_NSAMPLE_STAGE	common sample size per stage	-1
STOC_NUM_THREADS	number of parallel threads	1
STOC_RELOPTTOL	relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver	GAMS OptCR
STOC_REL_DSTEPTOL	dual-step tolerance	1e-7
STOC_REL_PSTEPTOL	primal-step tolerance	1e-8
STOC_REOPT	reoptimization method to solve the node-models	0
STOC_RG_SEED	seed to initialize the random number generator	1031
STOC_SAMP_CONT_ONLY	flag to restrict sampling to continuous stochastic parameters only or not	1
STOC_SBD_MAXCUTS	max cuts to generate for master problem	-1
STOC_SBD_NUMCANDID	maximum number of candidate solutions to generate at SBD root	-1
STOC_SBD_OBJCUTFLAG	flag to enable objective cut in SBD master problem	1
STOC_SBD_OBJCUTVAL	RHS value of objective cut in SBD master problem	1e-30
STOC_SHARE_BEGSTAGE	stage beyond which node-models are shared	-1
STOC_TIME_LIM	time limit for stochastic solver	GAMS ResLim
STOC_TOPOPT	optimization method to solve the root problem	0
STOC_VARCONTROL_METHOD	sampling method for variance reduction	1
STOC_WSBAS	warm start basis for wait-see model	-1
SVR_LS_ANTITHETIC	Sample variance reduction map to Lindo Antithetic algorithm	
SVR_LS_LATINSQUARE	Sample variance reduction map to Lindo Latin Square algorithm	
SVR_LS_MONTECARLO	Sample variance reduction map to Lindo Montecarlo algorithm	

5.24.6.8 IIS and IUS Options

Option	Description	Default
IIS	run the IIS finder if the problem is infeasible	0

Option	Description	Default
IIS_ANALYZE_LEVEL	controls the level of analysis when locating an IIS	1
IIS_GETMODE	flag that controls whether variable bounds in the IIS should be retrieved or the integer restrictions	0
IIS_INFEAS_NORM	specifies the norm to measure infeasibilities in IIS search	0
IIS_ITER_LIMIT	the iteration limit for IIS search	-1
IIS_METHOD	specifies the method to use in analyzing infeasible models to locate an IIS	0
IIS_PRINT_LEVEL	specifies the amount of print to do during IIS search	2
IIS_REOPT	specifies which optimization method to use when starting from a given basis	0
IIS_TIME_LIMIT	the time limit for IIS search	-1
IIS_TOPOPT	specifies which optimization method to use when there is no previous basis	0
IIS_USE_EFILTER	flag that controls whether the Elastic Filter should be enabled as the supplementary filter in analyzing infeasible models	0
IIS_USE_GOP	flag that controls whether the global optimizer should be enabled in analyzing infeasible NLP models	0
IIS_USE_SFILTER	flag indicating is sensitivity filter will be used during IIS search	1
IUS	run the IUS finder if the problem is unbounded	0
IUS_ANALYZE_LEVEL	controls the level of analysis when locating an IUS	2

5.24.6.9 Link Options

Option	Description	Default
CHECKRANGE	calculate feasible range for variables	range.gdx
READPARAMS	read Lindo parameter file	
WRITEDEMPI	write deterministic equivalent in MPI format	
WRITEDEMPS	write deterministic equivalent in MPS format	
WRITEMPI	write (S)MPI file of processed model	
WRITEMPS	write (S)MPS file of processed model	

5.24.7 Detailed Descriptions of GAMS/Lindo Options

CHECKRANGE *(string)*: calculate feasible range for variables ↩

If this option is set, Lindo calculates the feasible range (determined by an upper and lower bound) for every variable in each equation while all other variables are fixed to their level. If set, the value of this option defines the name of the GDX file where the results are written to. For every combination of equation- and variable block there will be one symbol in the format *EquBlock_VarBlock(equ_Ind_1, ..., equ_Ind_M, var_Ind_1, ..., var_Ind_N, directions)*.

Default: `range.gdx`

CORE_ORDER_BY_STAGE *(integer)*: order nontemporal models or not ↩

Order nontemporal models or not.

Default: 1

DECOMPOSITION_TYPE *(integer)*: decomposition to be performed on a linear or mixed integer model ↩

This refers to the type of decomposition to be performed on a linear or mixed integer model.

Default: 1

value	meaning
0	Solver decides which type of decomposition to use
1	Solver does not perform any decompositions and uses the original model
2	Attempt total decomposition
3	Decomposed model will have dual angular structure
4	Decomposed model will have block angular structure
5	Decomposed model will have both dual and block angular structure

EMPINFOFILE *(string)*: Path and name of file containing additional EMP-SP information as randvar, jrandvar, stage etc. ↩

FIND_BLOCK *(integer)*: graph partitioning method to find block structures ↩

Specifies the graph partitioning method to find block structures.

Default: 0

value	meaning
0	Use an edge-weight minimizing graph partitioning heuristic
1	Use a vertex-weight minimizing graph partitioning heuristic

FIND_SYMMETRY_LEVEL *(integer)*: specifies the symmetry finding level. ↩

Default: -1

value	meaning
-1	Solver decides
0	Finding orbit only without MIP preprocessing
1	Finding orbit only with MIP preprocessing
2	Finding generators without MIP preprocessing
3	Finding generators with MIP preprocessing
4	Finding the first generator without MIP preprocessing
5	Finding the first generator with MIP preprocessing

FIND_SYMMETRY_PRINT_LEVEL *(integer)*: specifies print level for symmetry finding ↩

Default: 0

value	meaning
0	Nothing printed
+2	General information
+4	Time information
+8	Orbit information
+16	Partition information

GOP_ABSOPTTOL *(real)*: absolute optimality tolerance ↩

Synonym: ABSOPTTOL

This value is the GOP absolute optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

Default: GAMS OptCA

GOP_ALGREFORMMD *(integer)*: algebraic reformulation rule for a GOP ↩

Synonym: ALGREFORMMD

This controls the algebraic reformulation rule for a GOP. The algebraic reformulation and analysis is very crucial in building a tight convex envelope to enclose the nonlinear/nonconvex functions. A lower degree of overestimation on convex envelopes helps increase the convergence rate to the global optimum.

Default: 18

value	meaning
+2	Rearrange and collect terms
+4	Expand all parentheses
+8	Retain nonlinear functions
+16	Selectively expand parentheses

GOP_BBSRCHMD *(integer)*: node selection rule in GOP branch-and-bound ↩

Synonym: BBSRCHMD

This specifies the node selection rule for choosing between all active nodes in the GOP branch-and-bound tree when solving global optimization programs.

Default: 1

value	meaning
0	Depth first search
1	Choose node with worst bound

GOP_BNDLIM *(real)*: max magnitude of variable bounds used in GOP convexification ↩

Synonym: BNDLIM

This value specifies the maximum magnitude of variable bounds used in the GOP convexification. Any lower bound smaller than the negative of this value will be treated as the negative of this

value. Any upper bound greater than this value will be treated as this value. This helps the global solver focus on more productive domains.

Default: `1e10`

GOP_BOXTOL *(real)*: minimal width of variable intervals ↩

Synonym: BOXTOL

This value specifies the minimal width of variable intervals in a box allowed to branch.

Default: `1e-6`

GOP_BRANCHMD *(integer)*: direction to branch first when branching on a variable ↩

Synonym: BRANCHMD

This specifies the direction to branch first when branching on a variable. The branch variable is selected as the one that holds the largest magnitude in the measure.

Default: `5`

value	meaning
0	Absolute width
1	Locally relative width
2	Globally relative width
3	Globally relative distance from the convex minimum to the bounds
4	Absolute violation between the function and its convex envelope at the convex minimum
5	Relative violation between the function and its convex envelope at the convex minimum

GOP_BRANCH_LIMIT *(integer)*: limit on the total number of branches to be created in GOP tree ↩

Synonym: BRANCH_LIMIT

This is the limit on the total number of branches to be created during branch-and- bound in GOP tree. The default value is -1, which means no limit is imposed. If the branch limit is reached and a feasible solution was found, it will be installed as the incumbent (best known) solution.

Range: {-1, ..., ∞}

Default: `-1`

GOP_CMINLP *(integer)*: flag indicating if GOP exploits convex MINLP model ↩

Default: `0`

value	meaning
0	Off
1	On

GOP_CONIC_REFORM *(integer)*: flag indicating if GOP explore conic reformulation ↩

Default: `1`

value	meaning
`0`	Off
`1`	On

GOP_CORELEVEL *(integer)*: strategy of GOP branch-and-bound ↩

Synonym: CORELEVEL

This controls the strategy of GOP branch-and-bound procedure.

Default: `14`

value	meaning
`+2`	LP convex relaxation
`+4`	NLP solving
`+8`	Box Branching

GOP_DECOMPPTMD *(integer)*: decomposition point selection rule in GOP branch-and-bound ↩

Synonym: DECOMPPTMD

This specifies the decomposition point selection rule. In the branch step of GOP branch-and-bound, a branch point M is selected to decompose the selected variable interval [Lb, Ub] into two subintervals, [Lb, M] and [M, Ub].

Default: `1`

value	meaning
`0`	Mid-point
`1`	Local minimum or convex minimum

GOP_DELTATOL *(real)*: delta tolerance in GOP convexification ↩

Synonym: DELTATOL

This value is the delta tolerance in the GOP convexification. It is a measure of how closely the additional constraints added as part of convexification should be satisfied.

Default: `1e-7`

GOP_FLTTOL *(real)*: floating-point tolerance ↩

Synonym: FLTTOL

This value is the GOP floating-point tolerance. It specifies the maximum rounding errors in the floating-point computation.

Default: `1e-10`

GOP_HEU_MODE *(integer)*: heuristic used in global solver ↩

Synonym: HEU_MODE

This specifies the heuristic used in the global solver to find a good solution. Typically, if a heuristic is used, this will put more efforts in searching for good solutions, and less in bound tightening.

Default: `0`

value	meaning
0	No heuristic is used
1	A simple heuristic is used

GOP_ITRLIM *(real)*: GOP iteration limit ↩

Synonym: ITRLIM

This is the total iteration limit (including simplex, barrier and nonlinear iteration) summed over branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: [-1, ∞]

Default: `infinity`

GOP_ITRLIM_IPM *(real)*: total barrier iteration limit summed over all branches in GOP ↩

Synonym: ITRLIM_IPM

This is the total barrier iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: [-1, ∞]

Default: `-1`

GOP_ITRLIM_NLP *(real)*: total nonlinear iteration limit summed over all branches in GOP ↩

Synonym: ITRLIM_NLP

This is the total nonlinear iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: [-1, ∞]

Default: `-1`

GOP_ITRLIM_SIM *(real)*: total simplex iteration limit summed over all branches in GOP ↩

Synonym: ITRLIM_SIM

This is the total simplex iteration limit summed over all branches in GOP. The default value is -1, which means no iteration limit is imposed. If this limit is reached, GOP will stop.

Range: [-1, ∞]

Default: `-1`

GOP_LIM_MODE *(integer)*: flag indicating which heuristic limit on sub-solver in GOP is based ↩

Synonym: LIM_MODE

This is a flag indicating which heuristic limit on sub-solver in GOP is based.

Default: `1`

value	meaning
0	No limit
1	Time based limit
2	Iteration based limit
3	Both time and iteration based limit

GOP_LINEARZ *(integer)*: flag indicating if GOP exploits linearizable model ↩

This is a flag indicating if GOP exploits linearizable model.

Default: 1

value	meaning
0	Exploit lineariable model
1	Do not exploit lineariable model

GOP_LSOLBRANLIM *(integer)*: branch limit until finding a new nonlinear solution ↩

Synonym: LSOLBRANLIM

This value controls the branch limit until finding a new nonlinear solution since the last nonlinear solution is found. The default value is -1, which means no branch limit is imposed.

Range: {-1, ..., ∞}

Default: -1

GOP_MAXWIDMD *(integer)*: maximum width flag for the global solution ↩

Synonym: MAXWIDMD

This is the maximum width flag for the global solution. The GOP branch-and-bound may continue contracting a box with an incumbent solution until its maximum width is smaller than GOP_WIDTOL.

Default: 0

value	meaning
0	The maximum width criterion is suppressed
1	The maximum width criterion is performed

GOP_MULTILINEAR *(integer)*: flag indicating if GOP exploits multi linear feature ↩

This is a flag indicating if GOP exploits multi linear feature.

Default: 1

value	meaning
0	Off
1	On

GOP_NUM_THREADS *(integer)*: number of parallel threads to be used when solving a nonlinear model with the global optimization solver ↩

This value specifies the number of parallel threads to be used when solving a nonlinear model with the global optimization solver.

Default: `1`

GOP_OBJ_THRESHOLD *(real)*: threshold of objective value in the GOP solver ↩

This value specifies the threshold of objective value in the GOP solver. For min problem, if current incumbent solution is less than the threshold GOP solver will stop.

Range: [`-1e+30`, ∞]

Default: `-1e+30`

GOP_OPTCHKMD *(integer)*: criterion used to certify the global optimality ↩

Synonym: OPTCHKMD

This specifies the criterion used to certify the global optimality. When this value is 0, the absolute deviation of objective lower and upper bounds should be smaller than `GOP_ABSOPTTOL` at the global optimum. When its value is 1, the relative deviation of objective lower and upper bounds should be smaller than `GOP_RELOPTTOL` at the global optimum. 2 means either absolute or relative tolerance is satisfied at global optimum.

Default: `2`

GOP_OPT_MODE *(integer)*: mode for GOP optimization ↩

Synonym: OPT_MODE

This specifies the mode for GOP optimization.

Default: `1`

value	meaning
0	Global search for a feasible solution (thus a feasibility certificate)
1	Global search for an optimal solution
2	Global search for an unboundedness certificate

GOP_POSTLEVEL *(integer)*: amount and type of GOP postsolving ↩

Synonym: POSTLEVEL

This controls the amount and type of GOP post-solving. The default value is: 6 = 2+4 meaning to do both of the below options.

Default: `6`

value	meaning
+2	Apply LSgetBestBound
+4	Reoptimize variable bounds

GOP_PRELEVEL *(integer)*: amount and type of GOP presolving ↩

Synonym: PRELEVEL

This controls the amount and type of GOP pre-solving. The default value is: 30 = 2+4+8+16 meaning to do all of the below options.

Default: `30`

value	meaning
`+2`	Initial local optimization
`+4`	Initial linear constraint propagation
`+8`	Recursive linear constraint propagation
`+16`	Recursive nonlinear constraint propagation

GOP_QUADMD *(integer)*: flag indicating if GOP exploits quadratic feature ↩

Default: `-1`

value	meaning
`-1`	Solver decides
`0`	No
`1`	Yes

GOP_QUAD_METHOD *(integer)*: specifies if the GOP solver should solve the model as a QP when applicable ↩

Default: `-1`

value	meaning
`-1`	Solver decides
`0`	General GOP solver
`1`	Specified QP solver

GOP_RELBRNDMD *(integer)*: reliable rounding in the GOP branch-and-bound ↩

Synonym: RELBRNDMD

This controls the reliable rounding rule in the GOP branch-and-bound. The global solver applies many suboptimizations to estimate the lower and upper bounds on the global optimum. A rounding error or numerical instability could unintentionally cut off a good solution. A variety of reliable approaches are available to improve the precision.

Default: `0`

value	meaning
`+2`	Use smaller optimality or feasibility tolerances and appropriate presolving options
`+4`	Apply interval arithmetic to reverify the solution feasibility

GOP_RELOPTTOL *(real)*: relative optimality tolerance ↩

Synonyms: OPTTOL RELOPTTOL

This value is the GOP relative optimality tolerance. Solutions must beat the incumbent by at least this amount to become the new best solution.

Default: `GAMS OptCR`

GOP_SOLLIM *(integer)*: integer solution limit for GOP branch-and-bound ↩

Range: {`-1`, ..., ∞}

Default: `-1`

GOP_SUBOUT_MODE *(integer)*: substituting out fixed variables ↩

Synonym: SUBOUT_MODE

This is a flag indicating whether fixed variables are substituted out of the instruction list used in the global solver.

Default: `1`

value	meaning
0	Do not substitute out fixed variables
1	Substitute out fixed variables

GOP_TIMLIM *(integer)*: time limit in seconds for GOP branch-and-bound ↩

Synonym: TIMLIM

This is the time limit in seconds for GOP branch-and-bound.

Range: {`-1`, ..., ∞}

Default: `GAMS ResLim`

GOP_USEBNDLIM *(integer)*: max magnitude of variable bounds flag for GOP convexification ↩

Synonym: USEBNDLIM

This value is a flag for the parameter GOP_BNDLIM.

Default: `2`

value	meaning
0	Do not use the bound limit on the variables
1	Use the bound limit right at the beginning of global optimization
2	Use the bound limit after the initial local optimization if selected

GOP_WIDTOL *(real)*: maximal width of variable intervals ↩

Synonym: WIDTOL

This value specifies the maximal width of variable intervals for a box to be considered as an incumbent box containing an incumbent solution. It is used when GOP_MAXWIDMD is set at 1.

Default: `1e-4`

IIS *(boolean)*: run the IIS finder if the problem is infeasible ↩

Default: `0`

IIS_ANALYZE_LEVEL *(integer)*: controls the level of analysis when locating an IIS ↩

Default: `1`

value	meaning
+1	Search for necessary rows
+2	Search for necessary columns
+4	Search for sufficient rows
+8	Search for sufficient columns
+16	Consider integrality restrictions as the potential cause of infeasibilities and include it in the analysis
+32	Compute the underlying LTF matrix and use this as the basis of a ranking score to guide the IIS run
+64	If the underlying matrix is totally decomposable, rank blocks w.r.t their sizes and debug the smallest independent infeasible block
+128	Use the nonzero structure of the underlying matrix to compute a ranking score to guide the IIS run
+256	Treat iter/time limits as intractability

IIS_GETMODE *(integer)*: flag that controls whether variable bounds in the IIS should be retrieved or the integer restrictions ↩

Default: `0`

value	meaning
0	Variable bound
1	Integer restrictions

IIS_INFEAS_NORM *(integer)*: specifies the norm to measure infeasibilities in IIS search ↩

Default: `0`

value	meaning
0	Solver decides
1	Use L-1 norm
2	L-infinity norm

IIS_ITER_LIMIT *(integer)*: the iteration limit for IIS search ↩

Range: {-1, ..., ∞}

Default: `-1`

IIS_METHOD *(integer)*: specifies the method to use in analyzing infeasible models to locate an IIS ↩

Default: `0`

value	meaning
0	Default filter
1	Standard deletion filter
2	Standard additive filter
3	Generalized-binary-search filter
4	Depth-first-binary-search filter
5	Fast-scan filter
6	Standard elastic filter

IIS_PRINT_LEVEL *(integer)*: specifies the amount of print to do during IIS search ↩

Default: 2

IIS_REOPT *(integer)*: specifies which optimization method to use when starting from a given basis ↩

Default: 0

value	meaning
0	Free
1	Primal Simplex
2	Dual Simplex
3	Barrier
4	NLP

IIS_TIME_LIMIT *(integer)*: the time limit for IIS search ↩

Range: {-1, ..., ∞}

Default: -1

IIS_TOPOPT *(integer)*: specifies which optimization method to use when there is no previous basis ↩

Default: 0

value	meaning
0	Free
1	Primal Simplex
2	Dual Simplex
3	Barrier
4	NLP

IIS_USE_EFILTER *(integer)*: flag that controls whether the Elastic Filter should be enabled as the supplementary filter in analyzing infeasible models ↩

Default: 0

value	meaning
-1	Solver decides
0	Do not use elastic filter
1	Use elastic filter

IIS_USE_GOP *(integer)*: flag that controls whether the global optimizer should be enabled in analyzing infeasible NLP models ↩

Default: `0`

value	meaning
`-1`	Solver decides
`0`	Do not use GOP
`1`	Use GOP

IIS_USE_SFILTER *(integer)*: flag indicating is sensitivity filter will be used during IIS search ↩

Default: `1`

value	meaning
`-1`	Solver decides
`0`	Do not use sensitivity filter
`1`	Use sensitivity filter

INSTRUCT_SUBOUT *(integer)*: flag to specify how to deal with fixed variables in the instruction list ↩

This is a flag indicating whether 1) fixed variables are substituted out of the instruction list, 2) performing numerical calculation on constant numbers and replacing with the results.

Default: `-1`

value	meaning
`-1`	Solver decides
`0`	Substitutions will not be performed
`1`	Substitutions will be performed

IPM_BASIS_REL_TOL_S *(real)*: maximum relative dual bound violation allowed in an optimal basic solution ↩

Maximum relative dual bound violation allowed in an optimal basic solution.

Default: `1e-12`

IPM_BASIS_TOL_S *(real)*: maximum absolute dual bound violation in an optimal basic solution ↩

Maximum absolute dual bound violation in an optimal basic solution.

Default: `1e-7`

IPM_BASIS_TOL_X *(real)*: maximum absolute primal bound violation allowed in an optimal basic solution ↩

Maximum absolute primal bound violation allowed in an optimal basic solution.

Default: `1e-7`

IPM_BI_LU_TOL_REL_PIV *(real)*: relative pivot tolerance used in the LU factorization in the basis identification procedure ↩

Relative pivot tolerance used in the LU factorization in the basis identification procedure.

Range: [0, 0.999999]

Default: 1e-2

IPM_CHECK_CONVEXITY *(integer)*: flag to check convexity of a quadratic program using barrier solver ↩

This is a flag to check convexity of a quadratic program using barrier solver.

Default: 1

value	meaning
-1	Check convexity only without solving the model
0	Use barrier solver to check convexity
1	Do not use barrier solver to check convexity

IPM_CO_TOL_DFEAS *(real)*: dual feasibility tolerance for Conic solver ↩

Default: 1e-8

IPM_CO_TOL_INFEAS *(real)*: maximum bound infeasibility tolerance for Conic solver ↩

Maximum bound infeasibility tolerance for Conic solver.

Default: 1e-12

IPM_CO_TOL_MU_RED *(real)*: optimality tolerance for Conic solver ↩

Default: 1e-8

IPM_CO_TOL_PFEAS *(real)*: primal feasibility tolerance for Conic solver ↩

Default: 1e-8

IPM_MAX_ITERATIONS *(integer)*: ipm iteration limit ↩

Controls the maximum number of iterations allowed in the interior-point optimizer.

Default: 1000

IPM_NUM_THREADS *(integer)*: number of threads to run the interiorpoint optimizer on ↩

Number of threads to run the interiorpoint optimizer on. This value should be less than or equal to the actual number of processors or cores on a multi-core system.

Default: 1

IPM_OFF_COL_TRH *(integer)*: extent for detecting the offending columns in the Jacobian of the constraint matrix ↩

Controls the extent for detecting the offending columns in the Jacobian of the constraint matrix. 0 means no offending columns will be detected. 1 means offending columns will be detected. In general, increasing the parameter value beyond the default value of 40 does not improve the result.

Default: `40`

IPM_TOL_DFEAS *(real)*: dual feasibility tolerance ↩

Dual feasibility tolerance used for linear and quadratic optimization problems.

Default: `1e-8`

IPM_TOL_DSAFE *(real)*: controls the initial dual starting point ↩

Controls the initial dual starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the dual variables associated with constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: [`1e-4`, ∞]

Default: `1`

IPM_TOL_INFEAS *(real)*: infeasibility tolerance ↩

This is the tolerance to declare the model primal or dual infeasible using the interior-point optimizer. A smaller number means the optimizer gets more conservative about declaring the model infeasible.

Default: `1e-10`

IPM_TOL_MU_RED *(real)*: relative complementarity gap tolerance ↩

Relative complementarity gap tolerance.

Default: `1e-16`

IPM_TOL_PATH *(real)*: how close to follow the central path ↩

Controls how close the interior-point optimizer follows the central path. A large value of this parameter means the central path is followed very closely. For numerically unstable problems it might help to increase this parameter.

Range: [`0`, `0.5`]

Default: `1e-8`

IPM_TOL_PFEAS *(real)*: primal feasibility tolerance ↩

Primal feasibility tolerance used for linear and quadratic optimization problems.

Default: `1e-8`

IPM_TOL_PSAFE *(real)*: controls the initial primal starting point ↩

Controls the initial primal starting point used by the interior-point optimizer. If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: [`1e-2`, ∞]

Default: `1`

IPM_TOL_REL_STEP *(real)*: relative step size to the boundary ↩

Relative step size to the boundary for linear and quadratic optimization problems.

Range: [`0`, `0.999999`]

Default: `0.9999`

IUS *(boolean)*: run the IUS finder if the problem is unbounded ↩

Default: `0`

IUS_ANALYZE_LEVEL *(integer)*: controls the level of analysis when locating an IUS ↩

Default: `2`

value	meaning
+2	Search for necessary columns
+8	Search for sufficient columns

LP_AIJ_ZEROTOL *(real)*: coefficient matrix zero tolerance ↩

Default: 2.22045e-16

LP_BIGM *(real)*: big-M for phase-I ↩

Default: 1e6

LP_BNDINF *(real)*: big-M to truncate lower and upper bounds in single phase dual-simplex ↩

Default: 1e15

LP_DPSWITCH *(integer)*: specifies whether LP primal-dual simplex switch is enabled or not ↩

Range: {0, ..., 1}

Default: 1

LP_DRATIO *(integer)*: controls the dual min-ratio strategy ↩

Range: {0, ..., 2}

Default: 1

LP_DYNOBJFACT *(real)*: Dynamic obj factor ↩

Range: [0, 1]

Default: 0.75

LP_DYNOBJMODE *(integer)*: Dynamic obj mode ↩

Default: 0

LP_ITRLMT *(integer)*: simplex iteration limit ↩

Synonym: SPLEX_ITRLMT

Range: {-1, ..., ∞}

Default: infinity

LP_PIV_BIGTOL *(real)*: simplex maximum pivot tolerance ↩

Default: 1e-5

LP_PIV_ZEROTOL *(real)*: simplex pivot zero tolerance ↩

Default: 1e-8

LP_PPARTIAL *(integer)*: primal simplex partial pricing method ↩

Range: {0, ..., 3}

Default: 0

LP_PRELEVEL *(integer)*: controls the amount and type of LP pre-solving ↩

This controls the amount and type of LP pre-solving to be used.

Default: 126

value	meaning
+2	Simple pre-solving
+4	Probing
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+512	Maximum pass

LP_RATRANGE *(integer)*: controls the number of pivot-candidates to consider when searching for a stable pivot in LU decomposition ↩

Range: {1, ..., ∞}

Default: 4

LP_SCALE *(integer)*: scaling flag ↩

Synonym: SPLEX_SCALE

Default: 1

value	meaning
0	Scaling is suppressed
1	Scaling is performed

LP_SPRINT_COLFACT *(integer)*: maximum number of columns in Sprint as a factor of number of rows ↩

Range: {1, ..., ∞}

Default: 10

LP_SPRINT_MAXPASS *(integer)*: maximum number of passes in Sprint method ↩

Range: {1, ..., ∞}

Default: 100

LP_SPRINT_SUB *(integer)*: LP method for subproblem in Sprint method ↩

Default: 0

MIP_ABSCUTTOL *(real)*: MIP absolute cut tolerance ↩

This is the MIP absolute cut tolerance. If the value is less than or equal to zero, it will use the internal decided tolerace, otherwise it will use this value as the abolute tolerace for adding cuts.

Range: [-1.0, ∞]

Default: -1.0

MIP_ABSOPTTOL *(real)*: MIP absolute optimality tolerance ↩

This is the MIP absolute optimality tolerance. Solutions must beat the incumbent by at least this absolute amount to become the new, best solution.

Default: `GAMS OptCA`

MIP_ADDCUTOBJTOL *(real)*: required objective improvement to continue generating cuts ↩

This specifies the minimum required improvement in the objective function for the cut generation phase to continue generating cuts.

Default: `1.5625e-5`

MIP_ADDCUTPER *(real)*: percentage of constraint cuts that can be added ↩

This determines how many constraint cuts can be added as a percentage of the number of original rows in an integer programming model.

Default: `0.75`

MIP_ADDCUTPER_TREE *(real)*: percentage of constraint cuts that can be added at child nodes ↩

This determines how many constraint cuts can be added at child nodes as a percentage of the number of original rows in an integer programming model.

Default: `0.5`

MIP_AGGCUTLIM_TOP *(integer)*: max number of constraints involved in derivation of aggregation cut at root node ↩

This specifies an upper limit on the number of constraints to be involved in the derivation of an aggregation cut at the root node. The default is .1, which means that the solver will decide.

Range: {`-1`, ..., ∞}

Default: `-1`

MIP_AGGCUTLIM_TREE *(integer)*: max number of constraints involved in derivation of aggregation cut at tree nodes ↩

This specifies an upper limit on the number of constraints to be involved in the derivation of an aggregation cut at the tree nodes. The default is .1, which means that the solver will decide.

Range: {`-1`, ..., ∞}

Default: `-1`

MIP_ANODES_SWITCH_DF *(integer)*: threshold on active nodes for switching to depth-first search ↩

This specifies the threshold on active nodes for switching to depth-first search rule.

Default: `50000`

MIP_AOPTTIMLIM *(integer)*: time in seconds beyond which the relative optimality tolerance will be applied ↩

This is the time in seconds beyond which the relative optimality tolerance, `MIP_PEROPTTOL` will be applied.

Default: `100`

MIP_BIGM_FOR_INTTOL *(real)*: threshold for which coefficient of a binary variable would be considered as big-M ↩

This value specifies the threshold for which the coefficient of a binary variable would be considered as big-M (when applicable).

Default: `1e8`

MIP_BRANCHDIR *(integer)*: first branching direction ↩

This specifies the direction to branch first when branching on a variable.

Default: `0`

value	meaning
0	Solver decides
1	Always branch up first
2	Always branch down first

MIP_BRANCHRULE *(integer)*: rule for choosing the variable to branch ↩

This specifies the rule for choosing the variable to branch on at the selected node.

Default: 0

value	meaning
0	Solver decides
1	Basis rounding with pseudo reduced costs
2	Maximum infeasibility
3	Pseudo reduced costs only
4	Maximum coefficient only
5	Previous branching only

MIP_BRANCH_LIMIT *(integer)*: limit on the total number of branches to be created during branch and bound ↩

This is the limit on the total number of branches to be created during branch-and- bound. The default value is -1, which means no limit is imposed. If the branch limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

Range: {-1, ..., ∞}

Default: -1

MIP_BRANCH_PRIO *(integer)*: controls how variable selection priorities are set and used ↩

This controls how variable selection priorities are set and used.

Default: 0

value	meaning
0	If the user has specified priorities then use them Otherwise let LINDO API decide
1	If user has specified priorities then use them Overwrite users choices if necessary
2	If user has specified priorities then use them Otherwise do not use any priorities
3	Let LINDO API set the priorities and ignore any user specified priorities
4	Binaries always have higher priority over general integers

MIP_CONCURRENT_REOPTMODE *(integer)*: specifies the concurrent optimization mode with warm start ↩

Default: 0

value	meaning
0	no concurrent runs
1	run concurrently if at least 2 threads exist
2	run concurrently

MIP_CONCURRENT_STRATEGY *(integer)*: controls the concurrent MIP strategy ↩

Default: `-1`

value	meaning
`-1`	Solver decides
`1`	Defines built-in priority lists for each thread
`3`	Defines heuristic based strategies for each thread

MIP_CONCURRENT_TOPOPTMODE *(integer)*: specifies the concurrent optimization mode with cold start ↩

Default: `0`

value	meaning
`0`	no concurrent runs
`1`	run concurrently if at least 2 threads exist
`2`	run concurrently

MIP_CUTDEPTH *(integer)*: threshold value for the depth of nodes in the branch and bound tree ↩

This controls a threshold value for the depth of nodes in the B&B tree, so cut generation will be less likely at those nodes deeper than this threshold.

Default: `8`

MIP_CUTFREQ *(integer)*: frequency of invoking cut generation at child nodes ↩

This controls the frequency of invoking cut generation at child nodes. The default value is 10, indicating that the MIP solver will try to generate cuts at every 10 nodes.

Default: `10`

MIP_CUTLEVEL_TOP *(integer)*: combination of cut types to try at the root node when solving a MIP ↩

This controls the combination of cut types to try at the root node when solving a MIP. Bit settings are used to enable the various cut types.

Default: `57342`

value	meaning
`+2`	GUB cover
`+4`	Flow cover
`+8`	Lifting
`+16`	Plant location
`+32`	Disaggregation
`+64`	Knapsack cover
`+128`	Lattice
`+256`	Gomory
`+512`	Coefficient reduction
`+1024`	GCD
`+2048`	Obj integrality
`+4096`	Basis Cuts
`+8192`	Cardinality Cuts
`+16384`	Disjunk Cuts
`+32768`	Soft Knapsack Cuts

MIP_CUTLEVEL_TREE *(integer)*: combination of cut types to try at child nodes in the branch and bound tree when solving a MIP ↩

This controls the combination of cut types to try at child nodes in the B&B tree when solving a MIP.

Default: `53246`

value	meaning
+2	GUB cover
+4	Flow cover
+8	Lifting
+16	Plant location
+32	Disaggregation
+64	Knapsack cover
+128	Lattice
+256	Gomory
+512	Coefficient reduction
+1024	GCD
+2048	Obj integrality
+4096	Basis Cuts
+8192	Cardinality Cuts
+16384	Disjunk Cuts
+32768	Soft Knapsack Cuts

MIP_CUTOFFOBJ *(real)*: defines limit for branch and bound ↩

If this is specified, then any part of the branch-and-bound tree that has a bound worse than this value will not be considered. This can be used to reduce the running time if a good bound is known.

Default: `1e30`

MIP_CUTTIMLIM *(integer)*: time to be spent in cut generation ↩

This controls the total time to be spent in cut generation throughout the solution of a MIP. The default value is -1, indicating that no time limits will be imposed when generating cuts.

Range: {`-1`, ..., ∞}

Default: `-1`

MIP_DELTA *(real)*: near-zero value used in linearizing nonlinear expressions ↩

This refers to a near-zero value used in linearizing nonlinear expressions.

Default: `1e-6`

MIP_DUAL_SOLUTION *(integer)*: flag for computing dual solution of LP relaxation ↩

This flag controls whether the dual solution to the LP relaxation that yielded the optimal MIP solution will be computed or not.

Default: `0`

value	meaning
0	Do not calculate dual solution for LP relaxation
1	Calculate dual solution for LP relaxation

MIP_FP_HEU_MODE *(integer)*: specifies the feasibility-pump (FP) heuristic mode ↩

Default: 0

value	meaning
0	FP is disabled
1	Solver decides
2	Enable FP if no cutoff value or initial mip solution was defined
3	Enable FP independent of cutoff values and initial mip solutions
4	Same as 2 but also enable FP on child nodes in branch-bound tree
5	Same as 3 but also enable FP on child nodes in branch-bound tree

MIP_FP_ITRLIM *(integer)*: iteration limit for feasibility pump heuristic ↩

This is the iteration limit in seconds for feasibility pump heuristic. A value of -1 means no iteration limit is imposed.

Default: 500

MIP_FP_MODE *(integer)*: mode for the feasibility pump heuristic ↩

Controls the mode for the feasibility pump heuristic.

Default: -1

value	meaning
-1	Solver decides
0	Off
1	On until the first solution
2	Try to get more than one solution

MIP_FP_OPT_METHOD *(integer)*: optimization and reoptimization method for feasibility pump heuristic ↩

This specifies optimization and reoptimization method for feasibility pump heuristic.

Default: 0

value	meaning
0	Solver decides
1	Primal simplex
2	Dual simplex
3	Barrier

MIP_FP_PROJECTION *(integer)*: type of objective function of LPs in projection step of the feasibility pump heuristic ↩

Range: {0, ..., 1}

Default: 0

MIP_FP_TIMLIM *(real)*: time limit for feasibility pump heuristic ↩

This is the time limit in seconds for feasibility pump heuristic. A value of -1 implies no time limit is imposed.

Default: 1800

MIP_FP_WEIGTH *(real)*: weight of the objective function in the feasibility pump ↩

Controls the weight of the objective function in the feasibility pump.

Range: [0, 1]

Default: 1

MIP_GENERAL_MODE *(integer)*: general strategy in solving MIPs ↩

This value specifies the general strategy in solving MIPs.

Default: 0

value	meaning
0	Solver decides
+2	Disable all time-driven events for reproducibility of runs
+16	Disable cut generation before branching

MIP_HEULEVEL *(integer)*: specifies heuristic used to find integer solution ↩

This specifies the heuristic used to find the integer solution. Possible values are: 0: No heuristic is used. 1: A simple heuristic is used. Typically, this will find integer solutions only on problems with a certain structure. However, it tends to be fast. 2: This is an advanced heuristic that tries to find a "good" integer solution fast. In general, a value of 2 seems to not increase the total solution time and will find an integer solution fast on many problems. A higher value may find an integer solution faster, or an integer solution where none would have been found with a lower level. Try level 3 or 4 on "difficult" problems where 2 does not help. Higher values cause more time to be spent in the heuristic. The value may be set arbitrarily high. However, >20 is probably not worthwhile. MIP_HEUMINTIMLIM controls the time to be spent in searching heuristic solutions.

Default: 3

MIP_HEUMINTIMLIM *(integer)*: minimum time in seconds to be spent in finding heuristic solutions ↩

This specifies the minimum time in seconds to be spent in finding heuristic solutions to the MIP model. MIP_HEULEVEL controls the heuristic used to find the integer solution.

Default: 0

MIP_HEU_DROP_OBJ *(integer)*: flag for whether to use without OBJ heuristic ↩

This is a flag for whether to use without OBJ heuristic.

Default: 0

value	meaning
0	Off
1	On

MIP_HEU_MODE *(integer)*: heuristic used in MIP solver ↩

This controls the MIP heuristic mode.

Default: 0

value	meaning
0	Solver decides when to stop the heuristic
1	Solver uses a pre-specified time limit to stop the heuristic.
2	Solver uses a pre-specified iteration limit to stop the heuristic

MIP_INTTOL *(real)*: absolute integer feasibility tolerance ↩

An integer variable is considered integer feasible if the absolute difference from the nearest integer is smaller than this.

Default: 1e-6

MIP_ITRLIM *(real)*: iteration limit for branch and bound ↩

This is the total LP iteration limit summed over all branches for branch-and-bound. Range for The default value is -1, which means no iteration limit is imposed. If this iteration limit is reached, branch-and-bound will stop and the best feasible integer solution found will be installed as the incumbent (best known) solution.

Range: [-1, ∞]

Default: infinity

MIP_KBEST_USE_GOP *(integer)*: specifies whether to use gop solver in MIP KBest ↩

Default: 0

value	meaning
0	No
1	Yes

MIP_KEEPINMEM *(integer)*: flag for keeping LP bases in memory ↩

If this is set to 1, the integer pre-solver will try to keep LP bases in memory. This typically gives faster solution times, but uses more memory. Setting this parameter to 0 causes the pre-solver to erase bases from memory.

Default: 1

value	meaning
0	Do not keep LP bases in memory
1	Keep LP bases in memory

MIP_LBIGM *(real)*: Big-M value used in linearizing nonlinear expressions ↩

This refers to the Big-M value used in linearizing nonlinear expressions.

Default: `10000`

MIP_LSOLTIMLIM *(integer)*: time limit until finding a new integer solution ↩

Range: {`-1`, ..., ∞}

Default: `-1`

MIP_MAKECUT_INACTIVE_COUNT *(integer)*: threshold for times a cut could remain active after successive reoptimization ↩

This value specifies the threshold for the times a cut could remain active after successive reoptimization during branch-and-bound. If the count is larger than the specified level the solver will inactive the cut.

Default: `10`

MIP_MAXCUTPASS_TOP *(integer)*: number passes to generate cuts on the root node ↩

This controls the number passes to generate cuts on the root node. Each of these passes will be followed by a reoptimization and a new batch of cuts will be generated at the new solution.

Default: `200`

MIP_MAXCUTPASS_TREE *(integer)*: number passes to generate cuts on the child nodes ↩

This controls the number passes to generate cuts on the child nodes. Each of these passes will be followed by a reoptimization and a new batch of cuts will be generated at the new solution.

Default: `2`

MIP_MAXNONIMP_CUTPASS *(integer)*: number of passes allowed in cut-generation that does not improve current relaxation ↩

This controls the maximum number of passes allowed in cut-generation that does not improve the current relaxation.

Default: `3`

MIP_MAXNUM_MIP_SOL_STORAGE *(integer)*: maximum number of k-best solutions to store ↩

This specifies the maximum number of k-best solutions to store. Possible values are positive integers.

Default: `1`

MIP_MINABSOBJSTEP *(real)*: value to update cutoff value each time a mixed integer solution is found ↩

This specifies the value to update the cutoff value each time a mixed integer solution is found.

Default: `0`

MIP_NODESELRULE *(integer)*: specifies the node selection rule ↩

This specifies the node selection rule for choosing between all active nodes in the branch-and-bound tree when solving integer programs.Possible selections are: 0: Solver decides (default). 1: Depth first search. 2: Choose node with worst bound. 3: Choose node with best bound. 4: Start with best bound. If no improvement in the gap between best bound and best integer solution is obtained for some time, switch to: if (number of active nodes<10000) Best estimate node selection (5). else Worst bound node selection (2). 5: Choose the node with the best estimate, where the new objective estimate is obtained using pseudo costs. 6: Same as (4), but start with the best estimate.

Default: `0`

value	meaning
0	Solver decides
1	Depth first search
2	Choose node with worst bound
3	Choose node with best bound
4	Start with best bound
5	Choose the node with the best estimate
6	Same as 4 but start with the best estimate

MIP_NUM_THREADS *(integer)*: number of parallel threads to use by the parallel MIP solver ↩

This parameter specifies the number of parallel threads to use by the parallel MIP solver. Possible values are positive integers. The default is 1 implying that the parallel solver is disabled.

Range: {1, ..., ∞}

Default: 1

MIP_PARA_FP *(integer)*: flag for whether to use parallelization on the feasibility pump heuristic ↩

This is a flag for whether to use parallelization on the feasibility pump heuristic.

Default: 1

value	meaning
0	Off
1	On

MIP_PARA_FP_MODE *(integer)*: flag for the mode of parallel feasibility pump ↩

This is a flag for the mode of parallel feasibility pump.

Default: 0

value	meaning
0	Terminate when all threads finish
1	Terminate as soon as the master thread finishes

MIP_PARA_INIT_NODE *(real)*: number of initial nodes for MIP parallelization ↩

This value specifies the number of initial nodes for MIP parallelization.

Range: [-1, ∞]

Default: -1

MIP_PARA_ITR_MODE *(integer)*: flag for iteration mode in MIP parallelization ↩

This is a flag for iteration mode in MIP parallelization.

Default: 1

value	meaning
0	Each thread terminates as soon as arrives iteration limit
1	Each thread terminates until all threads get iteration limit

MIP_PARA_RND_ITRLMT *(real)*: iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations ↩

This value specifies the iteration limit of each round in MIP parallelization, it is a weighted combination of simplex and barrier iterations.

Default: 2.0

MIP_PARA_SUB *(integer)*: flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing ↩

This is a flag for whether to use MIP parallelization on subproblems solved in MIP preprocessing.

Default: 1

value	meaning
0	Off
1	On

MIP_PEROPTTOL *(real)*: MIP relative optimality tolerance in effect after MIP_AOPTTIMLIM seconds ↩

This is the MIP relative optimality tolerance that will be in effect after T seconds following the start. The value T should be specified using the MIP_AOPTTIMLIM parameter.

Default: 1e-5

MIP_PERSPECTIVE_REFORM *(integer)*: flag for whether to use Perspective Reformulation ↩

This is the flag for wether to use Persective Reformulation.

Default: 1

value	meaning
0	Off
1	On

MIP_POLISH_ALPHA_TARGET *(real)*: proportion solutions in the pool to initiate a polishing-task at the current node ↩

This value specifies the proportion solutions in the pool to initiate a polishing-task at the current node.

Range: [0.01, 0.99]

Default: 0.6

MIP_POLISH_MAX_BRANCH_COUNT *(integer)*: maximum number of branches to polish ↩

This value specifies the maximum number of branches to polish.

Default: 2000

MIP_POLISH_NUM_BRANCH_NEXT *(integer)*: number of branches to polish in the next round ↩

This value specifies the number of branches to polish in the next round.

Default: 4000

MIP_PREHEU_DFE_VSTLIM *(integer)*: limit for the variable visit in depth first enumeration ↩

Limit for the variable visit in depth first enumeration.

Default: 200

MIP_PREHEU_LEVEL *(integer)*: heuristic level for the prerelax solver ↩

The heuristic level for the prerelax solver.

Default: 0

value	meaning
0	Nothing
1	One-change
2	One-change and two-change
3	Depth first enumeration

MIP_PREHEU_TC_ITERLIM *(integer)*: iteration limit for the two change heuristic ↩

Iteration limit for the two change heuristic.

Default: 30000000

MIP_PREHEU_VAR_SEQ *(integer)*: sequence of the variable considered by the prerelax heuristic ↩

The sequence of the variable considered by the prerelax heuristic.

Default: -1

value	meaning
-1	Backward
1	Forward

MIP_PRELEVEL *(integer)*: controls the amount and type of MIP pre-solving at root node ↩

This controls the amount and type of MIP pre-solving at root node.

Default: 3070

value	meaning
+2	Simple pre-solving
+4	Probing

value	meaning
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+128	Binary row presolving
+256	Row aggregation
+512	Coef Probe Lifting
+1024	Maximum pass
+2048	Similar row

MIP_PRELEVEL_TREE *(integer)*: amount and type of MIP pre-solving at tree nodes ↩

This controls the amount and type of MIP pre-solving at tree nodes.

Default: 1214

value	meaning
+2	Simple pre-solving
+4	Probing
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+128	Binary row presolving
+256	Row aggregation
+512	Coef Probe Lifting
+1024	Maximum pass

MIP_PRE_ELIM_FILL *(integer)*: controls fill-in introduced by eliminations during pre-solve ↩

This is a nonnegative value that controls the fill-in introduced by the eliminations during pre-solve. Smaller values could help when the total nonzeros in the presolved model is significantly more than the original model.

Default: 100

MIP_PSEUDOCOST_RULE *(integer)*: specifies the rule in pseudocost computations for variable selection ↩

This specifies the rule in pseudocost computations for variable selection.

Default: 0

value	meaning
0	Solver decides
1	Only use min pseudo cost
2	Only use max pseudo cost
3	Use quadratic score function and the pseudo cost weigth
4	Same as 3 without quadratic score

MIP_PSEUDOCOST_WEIGT *(real)*: weight in pseudocost computations for variable selection ↩

This specifies the weight in pseudocost computations for variable selection.

Default: `1.5625e-05`

MIP_REDCOSTFIX_CUTOFF *(real)*: cutoff value as a percentage of the reduced costs ↩

This specifies the cutoff value as a percentage of the reduced costs to be used in fixing variables when using the reduced cost fixing heuristic.

Default: `0.9`

MIP_REDCOSTFIX_CUTOFF_TREE *(real)*: cutoff value as a percentage of the reduced costs at tree nodes ↩

This specifies the cutoff value as a percentage of the reduced costs to be used in fixing variables when using the reduced cost fixing heuristic at tree nodes.

Default: `0.9`

MIP_RELINTTOL *(real)*: relative integer feasibility tolerance ↩

An integer variable is considered integer feasible if the difference between its value and the nearest integer value divided by the value of the nearest integer is less than this.

Default: `8e-6`

MIP_RELOPTTOL *(real)*: MIP relative optimality tolerance ↩

This is the MIP relative optimality tolerance. Solutions must beat the incumbent by at least this relative amount to become the new, best solution.

Default: `GAMS OptCR`

MIP_REOPT *(integer)*: optimization method to use when doing reoptimization ↩

This specifies which optimization method to use when doing reoptimization from a given basis.

Default: `0`

value	meaning
0	Solver decides
1	Use primal method
2	Use dual simplex
3	Use barrier solver

MIP_SCALING_BOUND *(integer)*: maximum difference between bounds of an integer variable for enabling scaling ↩

This controls the maximum difference between the upper and lower bounds of an integer variable that will enable the scaling in the simplex solver when solving a subproblem in the branch-and-bound tree.

Default: `10000`

MIP_SOLLIM *(integer)*: integer solution limit for MIP solver ↩

Range: {-1, ..., ∞}

Default: -1

MIP_SOLVERTYPE *(integer)*: optimization method to use when solving mixed-integer models ↩

This specifies the optimization method to use when solving mixed-integer models.

Default: 0

value	meaning
0	Solver decides
1	Use Branch and Bound only
2	Use Enumeration and Knapsack solver only

MIP_STRONGBRANCHDONUM *(integer)*: minimum number of variables to try the strong branching on ↩

This value specifies the minimum number of variables, among all the candidates, to try the strong branching on.

Default: 3

MIP_STRONGBRANCHLEVEL *(integer)*: depth from the root in which strong branching is used ↩

This specifies the depth from the root in which strong branching is used. The default value of 10 means that strong branching is used on a level of 1 to 10 measured from the root. Strong branching finds the real bound for branching on a given variable, which, in most cases, requires a solution of a linear program and may therefore also be quite expensive in computing time. However, if used on nodes close to the root node of the tree, it also gives a much better bound for that part of the tree and can therefore reduce the size of the branch-and-bound tree.

Default: 10

MIP_SWITCHFAC_SIM_IPM_ITER *(integer)*: specifies the (positive) factor that multiplies the number of constraints to impose an iteration limit to simplex method and trigger a switch over to the barrier method ↩

Range: {-1, ..., ∞}

Default: -1

MIP_SWITCHFAC_SIM_IPM_TIME *(real)*: factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method ↩

This specifies the (positive) factor that multiplies the number of constraints to impose a time limit to simplex method and trigger a switch over to the barrier method. A value of -1.0 means that no time limit is imposed.

Range: [-1, ∞]

Default: -1

MIP_SYMMETRY_MODE *(integer)*: specifies mip symmetry handling methods ↩

Default: 0

value	meaning
0	Do not use symmetries
1	Adding symmetry breaking cuts
2	Orbital fixing

MIP_SYMMETRY_NONZ *(integer)*: limit on number of nonzeros to look for symmetries ↩

Range: {0, ..., ∞}

Default: 50000

MIP_TIMLIM *(integer)*: time limit in seconds for integer solver ↩

This is the time limit in seconds for branch-and-bound. The default value is -1, which means no time limit is imposed. However, the value of SOLVER_TIMLMT will be applied to each continuous subproblem solve. If the value of this parameter is greater than 0, then the value of SOLVER_TIMLMT will be disregarded. If this time limit is reached and a feasible integer solution was found, it will be installed as the incumbent (best known) solution.

Range: {-1, ..., ∞}

Default: GAMS ResLim

MIP_TOPOPT *(integer)*: optimization method to use when there is no previous basis ↩

This specifies which optimization method to use when there is no previous basis.

Default: 0

value	meaning
0	Solver decides
1	Use primal method
2	Use dual simplex
3	Use barrier solver

MIP_TREEREORDERLEVEL *(integer)*: tree reordering level ↩

This specifies the tree reordering level.

Default: 10

MIP_TREEREORDERMODE *(integer)*: tree reordering mode ↩

This specifies the tree reordering mode.

Default: 1

value	meaning
1	Use tree reordering only for subproblems
2	Use tree reordering for subproblems and the main bnb loop only when LP status is infeasible
3	Not use tree reordering
4	Use tree reordering based on MIP_TREEREORDERLEVEL

MIP_USECUTOFFOBJ *(integer)*: flag for using branch and bound limit ↩

This is a flag for the parameter MIP_CUTOFFOBJ. If you do not want to lose the value of the parameter MIP_CUTOFFOBJ, this provides an alternative to disabling the cutoff objective.

Default: 1

value	meaning
0	Do not use current cutoff value
1	Use current cutoff value

MIP_USE_CUTS_HEU *(integer)*: controls if cut generation is enabled during MIP heuristics ↩

This flag controls if cut generation is enabled during MIP heuristics. The default is -1 (i.e. the solver decides).

Default: -1

value	meaning
-1	Solver decides
0	Do not use cut heuristic
1	Use cut heuristic

MIP_USE_ENUM_HEU *(integer)*: frequency of enumeration heuristic ↩

This specifies the frequency of enumeration heuristic.

Default: 4

value	meaning
0	Off
1	Only at top (root) node without cuts
2	Both at top (root) and tree nodes without cuts
3	Same as 1 with cuts
4	Same as 2 with cuts

MIP_USE_INT_ZERO_TOL *(integer)*: controls if all MIP calculations would be based on absolute integer feasibility tolarance ↩

This flag controls if all MIP calculations would be based on the integrality tolarance specified by MIP_INTTOL.

Default: 0

value	meaning
0	Do not base MIP calculations on MIP_INTTOL
1	Base MIP calculations on MIP_INTTOL

MULTITHREAD_MODE *(integer)*: threading mode ↩

This parameter controls the threading mode for solvers with multithreading support.

Default: `-1`

value	meaning
-1	Solver decides
1	Try parallel mode but if it is not available try concurrent mode
2	Try parallel mode only
3	Try concurrent mode but if it is not available try parallel mode
4	Try concurrent mode only

NLP_AUTODERIV *(integer)*: defining type of computing derivatives ↩

This is a flag to indicate if automatic differentiation is the method of choice for computing derivatives and select the type of differentiation.

Default: 0

value	meaning
0	Finite Differences approach will be used
1	Forward type of Automatic Differentiation will be used
2	Backward type of Automatic Differentiation will be used

NLP_AUTOHESS *(integer)*: flag for using Second Order Automatic Differentiation for solving NLP ↩

This is a flag to indicate if Second Order Automatic Differentiation will be performed in solving a nonlinear model. The second order derivatives provide an exact/precise Hessian matrix to the SQP algorithm, which may lead to less iterations and better solutions, but may also be quite expensive in computing time for some cases.

Default: 0

value	meaning
0	Do not use Second Order Automatic Differentiation
1	Use Second Order Automatic Differentiation

NLP_CONIC_REFORM *(integer)*: determines if to explore conic reformulation ↩

Default: 1

value	meaning
0	No
1	Yes

NLP_CONOPT_VER *(integer)*: specifies the CONOPT version to be used in NLP optimizations ↩

Range: {3, ..., 4}

Default: 3

NLP_CUTOFFOBJ *(real)*: as soon as any multi-start thread achieves this value all threads stop ↩

If the current best objective of the NLP being solved in a multistart run is better than this value, the solver will terminate early without exhausting the maximum number of multistarts. This is a way of saving computer time if the current best solution is sufficiently attractive.

Range: [-1e30, ∞]

Default: -1e30

NLP_DERIV_DIFFTYPE *(integer)*: flag indicating the technique used in computing derivatives with finite differences ↩

This is a flag indicating the technique used in computing derivatives with Finite Differences.

Default: 0

value	meaning
0	The solver decides
1	Use forward differencing method
2	Use backward differencing method
3	Use center differencing method

NLP_FEASCHK *(integer)*: how to report results when solution satisfies tolerance of scaled but not original model ↩

This input parameter specifies how the NLP solver reports the results when an optimal or local-optimal solution satisfies the feasibililty tolerance (NLP_FEASTOL) of the scaled model but not the original (descaled) one.

Default: 1

value	meaning
0	Perform no action accept the final solution
1	Declare the model status as FEASIBLE if maximum violation in the unscaled model is not higher than 10 times NLP_FEASTOL
2	Declare the model status as UNKNOWN if maximum violation in the unscaled model is higher than NLP_FEASTOL

NLP_FEASTOL *(real)*: feasibility tolerance for nonlinear constraints ↩

This is the feasibility tolerance for nonlinear constraints. A constraint is considered violated if the artificial, slack, or surplus variable associated with the constraint violates its lower or upper bounds by the feasibility tolerance.

Default: 1e-6

NLP_INF *(real)*: numeric infinity for nonlinear models ↩

Specifies the numeric infinity for nonlinear models. Possible values are positive real numbers. Smaller values could cause numerical problems.

nlp_ipm2grg This is a flag to switch from IPM solver to the standard NLP (GRG) solver when IPM fails due to numerical errors.

Default: 1e30

NLP_IPM2GRG *(integer)*: switch from IPM solver to GRG solver when IPM fails due to numerical errors ↩

Default: 1

value	meaning
0	Do not switch
1	Switch

NLP_ITERS_PER_LOGLINE *(integer)*: number of nonlinear iterations to elapse before next progress message ↩

Number of nonlinear iterations to elapse before next progress message.

Range: {1, ..., ∞}

Default: 10

NLP_ITRLMT *(integer)*: nonlinear iteration limit ↩

This controls the iteration limit on the number of nonlinear iterations performed.

Range: {-1, ..., ∞}

Default: GAMS IterLim

NLP_LINEARZ *(integer)*: extent to which the solver will attempt to linearize nonlinear models ↩

This determines the extent to which the solver will attempt to linearize nonlinear models.

Default: 0

value	meaning
0	Solver decides
1	No linearization occurs
2	Linearize ABS MAX and MIN functions
3	Same as option 2 plus IF AND OR NOT and all logical operators are linearized

NLP_LINEARZ_WB_CONSISTENT *(integer)*: determines if linearization process is consistent with WB/excel calculation ↩

Default: 0

value	meaning
0	No
1	Yes

NLP_MAXLOCALSEARCH *(integer)*: maximum number of local searches ↩

This controls the maximum number of local searches (multistarts) when solving a NLP using the multistart solver.

Default: 5

NLP_MAXLOCALSEARCH_TREE *(integer)*: maximum number of multistarts ↩

Maximum number of multistarts (at tree nodes)

Default: 1

NLP_MAX_RETRY *(integer)*: maximum number refinement retries to purify the final NLP solution ↩

Maximum number refinement retries to purify the final NLP solution.

Range: {-1, ..., ∞}

Default: 5

NLP_MSW_EUCDIST_THRES *(real)*: euclidean distance threshold in multistart search ↩

Euclidean distance threshold in multistart search

Default: 0.001

NLP_MSW_FILTMODE *(integer)*: filtering mode to exclude certain domains during sampling in multistart search ↩

Filtering mode to exclude certain domains during sampling in multistart search.

Default: -1

value	meaning
-1	Solver decides
+1	Filter-out the points around known KKT or feasible points previously visited
+2	Filter-out the points whose p are in the vicinity of p(x)
+4	Filter-out the points in the vicinity of x where x are initial points of all previous local optimizations
+8	Filter-out the points whose p(.) values are below a dynamic threshold tolerance

NLP_MSW_MAXNOIMP *(integer)*: maximum number of consecutive populations to generate without any improvements ↩

Maximum number of consecutive populations to generate without any improvements.

Range: {-1, ..., ∞}

Default: -1

NLP_MSW_MAXPOP *(integer)*: maximum number of populations to generate in multistart search ↩

Maximum number of populations to generate in multistart search.

Range: {-1, ..., ∞}

Default: -1

NLP_MSW_MAXREF *(integer)*: maximum number of reference points to generate trial points in multistart search ↩

Maximum number of reference points in the solution space to generate trial points in multistart search.

Range: {-1, ..., ∞}

Default: -1

NLP_MSW_NORM *(integer)*: norm to measure the distance between two points in multistart search ↩

Norm to measure the distance between two points in multistart search.

Range: {-1, ..., ∞}

Default: 2

NLP_MSW_NUM_THREADS *(integer)*: number of parallel threads to be used when solving an NLP model with the multistart solver ↩

This value specifies the number of parallel threads to be used when solving an NLP model with the multistart solver.

Default: 1

NLP_MSW_OVERLAP_RATIO *(real)*: rate of replacement in successive populations ↩

This value specifies the rate of replacement in successive populations. Higher values favors survival of points in the parent population.

Range: [0.0, 1.0]

Default: 0.1

NLP_MSW_POXDIST_THRES *(real)*: penalty function neighborhood threshold in multistart search ↩

Penalty function neighborhood threshold in multistart search

Default: 0.01

NLP_MSW_PREPMODE *(integer)*: preprocessing strategies in multistart solver ↩

This value specifies the preprocessing strategies in multistart solver.

Default: -1

value	meaning
-1	Solver decides
+1	Truncate free variables
+2	Scale reference points to origin
+4	Enable expansive scaling of radius[k] by hit[k]
+8	Skewed sampling allowing values in the vicinity of origin.
+16	Get best bounds by presolver
+32	Get best bounds using GOP
+64	Enable sampling of free variables (not recommended)
+128	Collect sufficiently many trial points prior to local solves
+256	Enable power solver, trying several different local strategies

NLP_MSW_RG_SEED *(integer)*: random number generator seed for the multistart solver ↩

This value specified the random number generator seed for the multistart solver.

Default: 1019

NLP_MSW_RMAPMODE *(integer)*: specifies the mode to map reference points in the unit cube into the original space ↩

Default: `-1`

value	meaning
`-1`	Solver decides
`0`	Use original variable bounds
`1`	Use min-max values over all sample points per each dimension
`2`	Use min-max values over all sample points over all dimensions

NLP_MSW_SOLIDX *(integer)*: index of the multistart solution to be loaded ↩

Index of the multistart solution to be loaded main solution structures.

Default: `0`

NLP_MSW_XKKTRAD_FACTOR *(real)*: KKT solution neighborhood factor in multistart search ↩

KKT solution neighborhood factor in multistart search

Default: `0.85`

NLP_MSW_XNULRAD_FACTOR *(real)*: initial solution neighborhood factor in multistart search ↩

Initial solution neighborhood factor in multistart search

Default: `0.5`

NLP_PRELEVEL *(integer)*: controls the amount and type of NLP pre-solving ↩

This controls the amount and type of NLP pre-solving.

Default: `126`

value	meaning
`+2`	Simple pre-solving
`+4`	Probing
`+8`	Coefficient reduction
`+16`	Elimination
`+32`	Dual reductions
`+64`	Use dual information
`+512`	Maximum pass

NLP_PSTEP_FINITEDIFF *(real)*: value of the step length in computing the derivatives using finite differences ↩

This controls the value of the step length in computing the derivatives using finite differences.

Default: `5e-7`

NLP_QUADCHK *(integer)*: flag for checking if NLP is quadratic ↩

This is a flag indicating if the nonlinear model should be examined to check if it is a quadratic model.

Default: `1`

value	meaning
0	Do not check if NLP is quadratic
1	Check if NLP is quadratic

NLP_REDGTOL *(real)*: tolerance for the gradients of nonlinear functions ↩

This is the tolerance for the gradients of nonlinear functions. The (projected) gradient of a function is considered to be the zero-vector if its norm is below this tolerance.

Default: 1e-7

NLP_SOLVER *(integer)*: type of nonlinear solver ↩

This value determines the type of nonlinear solver.

Default: 7

value	meaning
4	Solver decides
5	Uses Levenberg-Marquardt method to solve nonlinear least-squares problem
6	Uses Barrier solver for convex QCP models
7	Uses CONOPTs reduced gradient solver
8	Uses SLP solver
9	Uses CONOPT with multistart feature enabled

NLP_SOLVE_AS_LP *(integer)*: flag indicating if the nonlinear model will be solved as an LP ↩

This is a flag indicating if the nonlinear model will be solved as an LP. 1 means that an LP using first order approximations of the nonlinear terms in the model will be used when optimizing the model with the LSoptimize() function.

Default: 0

value	meaning
0	NLP will not be solved as LP
1	NLP will be solved as LP

NLP_STALL_ITRLMT *(integer)*: iteration limit before a sequence of non-improving NLP iterations is declared as stalling ↩

This specifies the iteration limit before a sequence of non-improving NLP iterations is declared as stalling, thus causing the solver to terminate.

Default: 100

NLP_STARTPOINT *(integer)*: flag for using initial starting solution for NLP ↩

This is a flag indicating if the nonlinear solver should accept initial starting solutions.

Default: 1

value	meaning
0	Do not use initial starting solution for NLP
1	Use initial starting solution for NLP

NLP_SUBSOLVER *(integer)*: type of nonlinear subsolver ↩

This controls the type of linear solver to be used for solving linear subproblems when solving nonlinear models.

Default: 1

value	meaning
1	Primal simplex method
2	Dual simplex method
3	Barrier solver with or without crossover

NLP_USECUTOFFOBJ *(integer)*: flag to use parameter NLP_CUTOFFOBJ ↩

This is a flag for the parameter NLP_CUTOFFOBJ. The value of 0 means NLP_CUTOFFOBJ will be ignored, else it will be used as specified.

Default: 0

value	meaning
-1	Solver decides
0	No
1	Yes

NLP_USE_CRASH *(integer)*: flag for using simple crash routines for initial solution ↩

This is a flag indicating if an initial solution will be computed using simple crash routines.

Default: 0

value	meaning
0	Do not use simple crash routines
1	Use simple crash routines

NLP_USE_LINDO_CRASH *(integer)*: flag for using advanced crash routines for initial solution ↩

This is a flag indicating if an initial solution will be computed using advanced crash routines.

Default: 1

value	meaning
0	Do not use advanced crash routines
1	Use advanced crash routines

NLP_USE_SDP *(integer)*: flag to use SDP solver for POSD constraint ↩

Default: 1

value	meaning
0	No
1	Yes

NLP_USE_SELCONEVAL *(integer)*: flag for using selective constraint evaluations for solving NLP ↩

This is a flag indicating if selective constraint evaluations will be performed in solving a nonlinear model.

Default: 1

value	meaning
0	Do not use selective constraint evaluations
1	Use selective constraint evaluations

NLP_USE_SLP *(integer)*: flag for using sequential linear programming step directions for updating solution ↩

This is a flag indicating if sequential linear programming step directions should be used in updating the solution.

Default: 1

value	meaning
-1	Solver decides
0	Do not use sequential linear programming step directions
1	Use sequential linear programming step directions

NLP_USE_STEEPEDGE *(integer)*: flag for using steepest edge directions for updating solution ↩

This is a flag indicating if steepest edge directions should be used in updating the solution.

Default: 0

value	meaning
0	Do not use steepest edge directions
1	Use steepest edge directions

NUM_THREADS *(integer)*: number of parallel threads to be used ↩

Synonym: gthreads

Number of threads to use in the solver routine to be called. It is a solver-independent parameter which internally sets solver-specific threading parameters automatically. If the GAMS `threads` parameter is set to 0, the Lindo default will be used, which is 1.

Range: {1, ..., ∞}

Default: `GAMS Threads`

PROB_TO_SOLVE *(integer)*: controls whether the explicit primal or dual form of the given LP problem will be solved ↩

This flag controls whether the explicit primal or dual form of the given LP problem will be solved.

Default: `0`

value	meaning
0	Solver decides
1	Explicit primal form
2	Explicit dual form

PROFILER_LEVEL *(integer)*: specifies the profiler level to break down the total cpu time into. ↩

Specifies the profiler level to break down the total cpu time into.

Default: `0`

value	meaning
0	Profiler is off
+1	Enable for simplex solver
+2	Enable for integer solver
+4	Enable for multistart solver
+8	Enable for global solver

READPARAMS *(string)*: read Lindo parameter file ↩

REPORTEVSOL *(no value)*: solve and report the expected value solution ↩

Default: `0`

SAMP_CDSINC *(real)*: correlation matrix diagonal shift increment ↩

Correlation matrix diagonal shift increment.

Default: `1e-6`

SAMP_NCM_CUTOBJ *(real)*: objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem ↩

Objective cutoff (target) value to stop the nearest correlation matrix (NCM) subproblem.

Default: `1e-30`

SAMP_NCM_DSTORAGE *(integer)*: flag to enable or disable sparse mode in NCM computations ↩

Flag to enable/disable sparse mode in NCM computations.

Range: {`-1`, ..., ∞}

Default: `-1`

SAMP_NCM_ITERLIM *(integer)*: iteration limit for NCM method ↩

Iteration limit for NCM method.

Default: `100`

SAMP_NCM_METHOD *(integer)*: bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem ↩

Bitmask to enable methods for solving the nearest correlation matrix (NCM) subproblem.

Default: `5`

SAMP_NCM_OPTTOL *(real)*: optimality tolerance for NCM method ↩

Optimality tolerance for NCM method.

Default: `1e-7`

SAMP_NUM_THREADS *(integer)*: specifies the number of parallel threads to be used when sampling ↩

Default: `0`

SAMP_RG_BUFFER_SIZE *(integer)*: specifies the buffer size for random number generators in running in parallel mode ↩

Default: `0`

SAMP_SCALE *(integer)*: flag to enable scaling of raw sample data ↩

Flag to enable scaling of raw sample data.

Default: `0`

SOLVER_CONCURRENT_OPTMODE *(integer)*: controls if simplex and interior-point optimizers will run concurrently ↩

Controls if simplex and interior-point optimizers will run concurrently, 0 means no concurrent runs will be performed, 1 means both optimizers will run concurrently if at least two threads exist in system, 2 means both optimizers will run concurrently.

Default: `0`

value	meaning
0	no concurrent runs
1	run concurrently if at least 2 threads exist
2	run concurrently

SOLVER_CUTOFFVAL *(real)*: solver will exit if optimal solution is worse than this ↩

If the optimal objective value of the LP being solved is shown to be worse than this (e.g., if the dual simplex method is being used), then the solver will exit without finding a feasible solution. This is a way of saving computer time if there is no sufficiently attractive solution. SOLVER_USECUTOFFVAL needs to be set to 1 to activate this value.

Default: `0`

SOLVER_FEASTOL *(real)*: feasibility tolerance ↩

This is the feasibility tolerance. A constraint is considered violated if the artificial, slack, or surplus variable associated with the constraint violates its lower or upper bounds by the feasibility tolerance.

Default: `1e-7`

SOLVER_IPMSOL *(integer)*: basis crossover flag for barrier solver ↩

This flag controls whether a basis crossover will be performed when solving LPs with the barrier solver. A value of 0 indicates that a crossover to a basic solution will be performed. If the value is 1, then the barrier solution will be left intact. For example, if alternate optima exist, the barrier method will return a solution that is, loosely speaking, the average of all alternate optima.

Default: `0`

value	meaning
0	Perform crossover to basis solution
1	Leave barrier solution intact

SOLVER_IUSOL *(integer)*: flag for computing basic solution for infeasible model ↩

This is a flag that, when set to 1, will force the solver to compute a basic solution to an infeasible model that minimizes the sum of infeasibilities and a basic feasible solution to an unbounded problem from which an extreme direction originates. When set to the default of 0, the solver will return with an appopriate status flag as soon as infeasibility or unboundedness is detected. If infeasibility or unboundedness is declared with presolver's determination, no solution will be computed.

Default: `0`

value	meaning
0	Return appropriate status if infeasibility is encountered
1	Force the solver to compute a basic solution to an infeasible model

SOLVER_METHOD *(integer)*: specifies the method to use when generic solver is invoked ↩

Default: `0`

value	meaning
0	FREE
1	PSIMPLEX
2	DSIMPLEX
3	BARRIER
4	NLP
5	MIP
6	MULTIS
7	GOP
8	IIS
9	IUS
10	SBD
11	SPRINT
12	GA

SOLVER_MODE *(integer)*: controls some of the advanced strategies when solving LPs ↩

Default: `1`

value	meaning
`+1`	add distinct basic solutions to the pool of alternate optimal solutions
`+2`	add edge/nonbasic solutions to the pool of alternate optimal solutions
`+4`	favor basic solutions with integer values when choosing solutions to add to the pool of alternate optimal solutions

SOLVER_OPTTOL *(real)*: dual feasibility tolerance ↩

This is the optimality tolerance. It is also referred to as the dual feasibility tolerance. A dual slack (reduced cost) is considered violated if it violates its lower bound by the optimality tolerance.

Default: `1e-7`

SOLVER_PRE_ELIM_FILL *(integer)*: fill-in introduced by the eliminations during pre-solve ↩

This is a nonnegative value that controls the fill-in introduced by the eliminations during pre-solve. Smaller values could help when the total nonzeros in the presolved model is significantly more than the original model.

Default: `1000`

SOLVER_RESTART *(integer)*: starting basis flag ↩

This is the starting basis flag. 1 means LINDO API will perform warm starts using any basis currently in memory. 0 means LINDO API will perform cold starts discarding any basis in memory and starting from scratch.

Default: `0`

value	meaning
`0`	Perform cold start
`1`	Perform warm start

SOLVER_TIMLMT *(integer)*: time limit in seconds for continous solver ↩

This is a time limit in seconds for the LP solver. The default value of -1 imposes no time limit.

Range: {`-1`, ..., ∞}

Default: `GAMS ResLim`

SOLVER_USECUTOFFVAL *(integer)*: flag for using cutoff value ↩

This is a flag for the parameter SOLVER_CUTOFFVAL

Default: `0`

value	meaning
`0`	Do not use cutoff value
`1`	Use cutoff value

SPLEX_DPRICING *(integer)*: pricing option for dual simplex method ↩

This is the pricing option to be used by the dual simplex method.

Default: `-1`

value	meaning
`-1`	Solver decides the dual pricing method
`0`	Partial pricing
`1`	Steepest edge

SPLEX_DUAL_PHASE *(integer)*: controls the dual simplex strategy ↩

This controls the dual simplex strategy, single-phase versus two-phase.

Default: `0`

value	meaning
`0`	Solver decides
`1`	Single-phase
`2`	Two-phase

SPLEX_PPRICING *(integer)*: pricing option for primal simplex method ↩

This is the pricing option to be used by the primal simplex method.

Default: `-1`

value	meaning
`-1`	Solver decides the primal pricing method
`0`	Partial pricing
`1`	Devex

SPLEX_REFACFRQ *(integer)*: number of simplex iterations between two consecutive basis re-factorizations ↩

This is a positive integer scalar referring to the simplex iterations between two consecutive basis re-factorizations. For numerically unstable models, setting this parameter to smaller values may help.

Default: `100`

STOC_ABSOPTTOL *(real)*: absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver ↩

Absolute optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. . Possible values are reals in (0,1) interval.

Default: `GAMS OptCA`

STOC_ADD_MPI *(integer)*: flag to use add-instructions mode when building deteq ↩

Flag to use add-instructions mode when building deteq.

Default: `0`

STOC_ALD_DUAL_FEASTOL *(real)*: dual feasibility tolerance for ALD ↩

Dual feasibility tolerance for ALD.

Default: `1e-4`

STOC_ALD_DUAL_STEPLEN *(real)*: dual step length for ALD ↩

Dual step length for ALD.

Default: `0.9`

STOC_ALD_INNER_ITER_LIM *(integer)*: inner loop iteration limit for ALD ↩

Inner loop iteration limit for ALD.

Default: `1000`

STOC_ALD_OUTER_ITER_LIM *(integer)*: outer loop iteration limit for ALD ↩

Outer loop iteration limit for ALD.

Default: `200`

STOC_ALD_PRIMAL_FEASTOL *(real)*: primal feasibility tolerance for ALD ↩

Primal feasibility tolerance for ALD.

Default: `1e-4`

STOC_ALD_PRIMAL_STEPLEN *(real)*: primal step length for ALD ↩

Primal step length for ALD.

Default: `0.5`

STOC_AUTOAGGR *(integer)*: flag to enable or disable autoaggregation ↩

Flag to enable or disable autoaggregation.

Default: `1`

STOC_BENCHMARK_SCEN *(integer)*: benchmark scenario to compare EVPI and EVMU against ↩

Benchmark scenario to compare EVPI and EVMU against.

Range: {`-2`, ..., ∞}

Default: `-2`

STOC_BIGM *(real)*: big-M value for linearization and penalty functions ↩

Big-M value for linearization and penalty functions.

Default: `1e7`

STOC_BUCKET_SIZE *(integer)*: bucket size in Benders decomposition ↩

Bucket size in Benders decomposition. Possible values are positive integers or (-1) for solver decides.

Range: {`-1`, ..., ∞}

Default: `-1`

STOC_CALC_EVPI *(integer)*: flag to enable or disable calculation of EVPI ↩

Flag to enable/disable calculation of lower bounds on EVPI.

Default: `1`

value	meaning
0	disable
1	enable

STOC_CORRELATION_TYPE *(integer)*: correlation type associated with correlation matrix ↩

Correlation type associated with the correlation matrix.

Default: 0

value	meaning
-1	Target correlation
0	Pearson correlation
1	Kendall correlation
2	Spearman correlation

STOC_DEQOPT *(integer)*: method to solve the DETEQ problem ↩

This specifies the method to use when solving the deterministic equivalent.

Default: 0

value	meaning
0	Solver decides
10	Use simple Benders Decomposition

STOC_DETEQ_TYPE *(integer)*: type of deterministic equivalent ↩

Type of deterministic equivalent to be used by the solver. Implicit determinisitc equivalent is valid for linear and integer models only.

Default: -1

value	meaning
-1	Solver decides
0	Implicit determinisitc equivalent
1	Explicit determinisitc equivalent

STOC_DS_SUBFORM *(integer)*: subproblem formulation to use in DirectSearch ↩

This parameter specifies the type of subproblem formulation to be used in heuristic search.

Default: -1

value	meaning
-1	Solver decides
0	Perform heuristic search in the original solution space
1	Perform heuristic search in the space of discrete variables coupled with optimizations in the linear space

STOC_ELIM_FXVAR *(integer)*: flag to enable elimination of fixed variables from deteq MPI ↩

Flag to enable elimination of fixed variables from deteq MPI.

Default: `1`

STOC_INFBND *(real)*: value to truncate infinite bounds at non-leaf nodes ↩

Value to truncate infinite bounds at nonleaf nodes.

Default: `1e9`

STOC_ITER_LIM *(integer)*: iteration limit for stochastic solver ↩

Iteration limit for stochastic solver. Possible values are positive integers or (-1) no limit.

Range: {`-1`, ..., ∞}

Default: `infinity`

STOC_MAP_MPI2LP *(integer)*: flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements ↩

Flag to specify whether stochastic parameters in MPI will be mapped as LP matrix elements.

Default: `0`

STOC_MAX_NUMSCENS *(integer)*: maximum number of scenarios before forcing automatic sampling ↩

Maximum number of scenarios before forcing automatic sampling. Possible values are positive integers.

Default: `40000`

STOC_METHOD *(integer)*: stochastic optimization method to solve the model ↩

Stochastic optimization method to solve the model.

Default: `-1`

value	meaning
`-1`	Solve with the method chosen by the solver
`0`	Solve the deterministic equivalent (DETEQ)
`1`	Solve with the Nested Benders Decomposition (NBD) method

STOC_NAMEDATA_LEVEL *(integer)*: name data level ↩

Name data level.

Default: `1`

STOC_NODELP_PRELEVEL *(integer)*: presolve level solving node-models ↩

Presolve level solving node-models.

Default: `0`

value	meaning
+2	Simple pre-solving
+4	Probing
+8	Coefficient reduction
+16	Elimination
+32	Dual reductions
+64	Use dual information
+512	Maximum pass

STOC_NSAMPLE_PER_STAGE *(string)*: list of sample sizes per stage (starting at stage 2) ↩

Comma separated list of sample sizes per stage. The sample size of stage 1 is assumed to be 1 so that this list starts with stage stage 2.

STOC_NSAMPLE_SPAR *(integer)*: common sample size per stochastic parameter ↩

Common sample size per stochastic parameter. Possible values are positive integers.

Range: {-1, ..., ∞}

Default: -1

STOC_NSAMPLE_STAGE *(integer)*: common sample size per stage ↩

Common sample size per stage.

Range: {-1, ..., ∞}

Default: -1

STOC_NUM_THREADS *(integer)*: number of parallel threads ↩

This value specifies the number of parallel threads to be used when solving a stochastic programming model.

Default: 1

STOC_RELOPTTOL *(real)*: relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver ↩

Relative optimality tolerance (w.r.t lower and upper bounds on the true objective) to stop the solver. Possible values are reals in (0,1) interval.

Default: GAMS OptCR

STOC_REL_DSTEPTOL *(real)*: dual-step tolerance ↩

This value specifies the dual-step tolerance in decomposition based algorithms.

Default: 1e-7

STOC_REL_PSTEPTOL *(real)*: primal-step tolerance ↩

This value specifies the primal-step tolerance in decomposition based algorithms.

Default: 1e-8

STOC_REOPT *(integer)*: reoptimization method to solve the node-models ↩

Reoptimization method to solve the node-models.

Default: 0

value	meaning
0	Solver decides
1	Use primal method
2	Use dual simplex
3	Use barrier solver
4	Use NLP solver

STOC_RG_SEED *(integer)*: seed to initialize the random number generator ↩

Seed to initialize the random number generator. Possible values are positive integers.

Default: 1031

STOC_SAMP_CONT_ONLY *(integer)*: flag to restrict sampling to continuous stochastic parameters only or not ↩

Flag to restrict sampling to continuous stochastic parameters only or not.

Default: 1

value	meaning
0	disable
1	enable

STOC_SBD_MAXCUTS *(integer)*: max cuts to generate for master problem ↩

Max cuts to generate for master problem.

Range: {-1, ..., ∞}

Default: -1

STOC_SBD_NUMCANDID *(integer)*: maximum number of candidate solutions to generate at SBD root ↩

Maximum number of candidate solutions to generate at SBD root.

Range: {-1, ..., ∞}

Default: -1

STOC_SBD_OBJCUTFLAG *(integer)*: flag to enable objective cut in SBD master problem ↩

Flag to enable objective cut in SBD master problem.

Default: 1

STOC_SBD_OBJCUTVAL *(real)*: RHS value of objective cut in SBD master problem ↩

RHS value of objective cut in SBD master problem.

Default: 1e-30

STOC_SHARE_BEGSTAGE *(integer)*: stage beyond which node-models are shared ↩

Stage beyond which node-models share the same model structure. Possible values are positive integers less than or equal to number of stages in the model or (-1) for solver decides.

Range: {-1, ..., ∞}

Default: -1

STOC_TIME_LIM *(real)*: time limit for stochastic solver ↩

Time limit for stochastic solver. Possible values are nonnegative real numbers or -1 for solver decides.

Range: [-1, ∞]

Default: GAMS ResLim

STOC_TOPOPT *(integer)*: optimization method to solve the root problem ↩

Optimization method to solve the root problem.

Default: 0

value	meaning
0	Solver decides
1	Use primal method
2	Use dual simplex
3	Use barrier solver
4	Use NLP solver
6	Use multi-start solver
7	Use global solver

STOC_VARCONTROL_METHOD *(integer)*: sampling method for variance reduction ↩

Sampling method for variance reduction.

Default: 1

value	meaning
0	Montecarlo sampling
1	Latinsquare sampling
2	Antithetic sampling

STOC_WSBAS *(integer)*: warm start basis for wait-see model ↩

Warm start basis for wait-see model .

Range: {-1, ..., ∞}

Default: -1

SVR_LS_ANTITHETIC *(string)*: Sample variance reduction map to Lindo Antithetic algorithm ↩

SVR_LS_LATINSQUARE *(string)*: Sample variance reduction map to Lindo Latin Square algorithm ↩

SVR_LS_MONTECARLO *(string)*: Sample variance reduction map to Lindo Montecarlo algorithm ↩

TUNER_PRINT_LEVEL *(integer)*: specifies the amount of print to do during parameter tuning ↩

Default: 1

value	meaning
0	Do not print anything default
>0	Print more information

USEGOP *(integer)*: use global optimization ↩

This value determines whether the global optimization will be used.

Default: 1

value	meaning
0	Do not use global optimization
1	Use global optimization

WRITEDEMPI *(string)*: write deterministic equivalent in MPI format ↩

WRITEDEMPS *(string)*: write deterministic equivalent in MPS format ↩

WRITEMPI *(string)*: write (S)MPI file of processed model ↩

If this option is set, Lindo write an MPI file of processed model. If set, the value of this option defines the name of the MPI file.

WRITEMPS *(string)*: write (S)MPS file of processed model ↩

5.24.8 Stochastic Programming (SP) in GAMS/Lindo

GAMS/Lindo can also solve stochastic programming models. The syntax to set up an SP problem in GAMS is explained in the chapter Stochastic Programming (SP) with EMP. The options to control LINDOs stochastic solver are described in the subsection SP Options.

5.25 MILES

Thomas F. Rutherford, University of Colorado

5.25.1 Abstract

MILES is a solver for nonlinear complementarity problems and nonlinear systems of equations. The solver can be accessed through GAMS to solve MPSGE or MCP models. This paper documents the solution algorithm, user options, and solver output. The purpose of the paper is to provide users of MILES an overview of how the solver works so that they can use it effectively.

5.25.2 Introduction

MILES is a Fortran-based solver for nonlinear complementarity problems and nonlinear systems of equations. The solution procedure is a generalized Newton method with a backtracking line search. This code is based on an algorithm investigated by Mathiesen (1985) who proposed a modeling format and sequential method for solving economic equilibrium models. The method is closely related to algorithms proposed by Robinson (1975), Hogan (1977), Eaves (1978) and Josephy (1979). In this implementation, subproblems are solved as linear complementarity problems (LCPs), using an extension of Lemke's almost-complementary pivoting scheme in which upper and lower bounds are represented implicitly. The linear solver employs the basis factorization package LUSOL, developed by Gill et al. (1991).

The class of problems for which MILES may be applied are referred to as "generalized" or "mixed" complementarity problems, which is defined as follows:

$$\begin{aligned} \text{Given:} \quad & F : R^n \to R^n \ , \quad \ell, u \in R^n \\ \text{Find:} \quad & z, w, v \in R^n \\ \text{such that} \quad & F(z) = w - v \\ & \ell \le z \le u, \quad w \ge 0, \quad v \ge 0 \\ & w^T(z - \ell) = 0 \ , \quad v^T(u - z) = 0. \end{aligned}$$

When $\ell = -\infty$ and $u = \infty$ MCP reduces to a nonlinear system of equations. When $\ell = 0$ and $u = +\infty$, the MCP is a nonlinear complementarity problem. Finite dimensional variational inequalities are also MCP. MCP includes inequality-constrained linear, quadratic and nonlinear programs as special cases, although for these problems standard optimization methods may be preferred. MCP models which are not optimization problems encompass a large class of interesting mathematical programs. Specific examples of MCP formulations are not provided here. See Rutherford (1992a) for MCP formulations arising in economics. Other examples are provided by Harker and Pang (1990) and Dirkse (1993).

There are two types of GAMS models which can be presented to MILES:

1. MILES may be used to solve computable general equilibrium models generated by MPSGE as a GAMS subsystem. In the MPSGE language, a model-builder specifies classes of nonlinear functions using a specialized tabular input format embedded within a GAMS program. Using benchmark quantities and prices, MPSGE automatically calibrates function coefficients and generates nonlinear equations and Jacobian matrices. Large, complicated systems of nonlinear equations may be implemented and analyzed very easily using this interface to MILES. An introduction to general equilibrium modeling with GAMS/MPSGE is provided by Rutherford (1992a).

2. MILES may be accessed as a GAMS solver for mixed complementarity problems (MCP). If more than one MCP solver is available [3], the statement `OPTION MCP = MILES;` tells GAMS to use MILES as the MCP solution system. When problems are presented to MILES using the MCP format, the user specifies nonlinear functions using GAMS matrix algebra and the GAMS compiler automatically generates the Jacobian functions. An introduction to the GAMS/MCP modeling format is provided by Rutherford (1992b).

The purpose of this document is to provide users of MILES with an overview of how the solver works so that they can use the program more effectively. Section The Newton Algorithm introduces the Newton algorithm. Section Lemke's Method with Implicit Bounds describes the implementation of Lemke's algorithm which is used to solve linear subproblems. Section The Options File defines switches and tolerances which may be specified using the options file. Section Log File Output interprets the run-time log file which is normally directed to the screen. Section Status File Output interprets the status file and the detailed iteration reports which may be generated. Section Termination Messages lists and suggests remedies for abnormal termination conditions.

5.25.3 The Newton Algorithm

The iterative procedure applied within MILES to solve nonlinear complementarity problems is closely related to the classical Newton algorithm for nonlinear equations. This first part of this section reviews the classical procedure. A thorough introduction to these ideas is provided by Dennis and Schnabel (1983). For a practical perspective, see Press et al. (1986).

Newton algorithms for nonlinear equations begin with a local (Taylor series) approximation of the system of nonlinear equations. For a point z in the neighborhood of $\hat{z}$, the system of nonlinear functions is linearized:

$$LF(z) = F(\bar{z}) + \nabla F(\bar{z})(z - \bar{z}).$$

Solving the linear system $LF(z) = 0$ provides the Newton direction from $\bar{z}$ which is given by $d = -\nabla F^{-1}\ F(\bar{z})$.

Newton iteration k begins at point z^k. First, the linear model formed at z^k is solved to determine the associated "Newton direction", d^k. Second, a line search in direction d^k determines the scalar steplength and the subsequent iterate: $z^{k+1} = z^k + \lambda d^k$. An Armijo or "back-tracking" line search initially considers $\lambda = 1$. If $\|F(z^z + \lambda d^k)\| \leq \|F(z^k)\|$, the step size λ is adopted,otherwise is multiplied by a positive factor α, $\alpha < 1$, and the convergence test is reapplied. This procedure is repeated until either an improvement results or $\lambda < \underline{\lambda}$. When $\underline{\lambda} = 0$, a positive step is taken provided that [4]

$$\frac{d}{d\lambda}\|F(z^k + \lambda d^k)\| < 0.$$

Convergence theory for this algorithm is quite well developed. See, for example, Ortega and Rheinbolt (1970) or Dennis and Schnabel (1983). The most attractive aspect of the Newton scheme with the backtracking line search is that in the neighborhood of a well-behaved fixed point, $\lambda = 1$ is the optimal step length and the rate of convergence can be quadratic. If this method finds a solution, it does so very quickly.

The application of Newton methods to nonlinear complementarity problems involves a modification of the search direction. Here, d solves a *linear complementarity problem* (LCP) rather than a linear system of equations. For iteration k, d solves:

[3] There is one other MCP solver available through GAMS: PATH (Ferris and Dirkse,1992)
[4] α and $\underline{\lambda}$ correspond to user-specified tolerances 'DMPFAC' and 'MINSTP', respectively

$$F(z^k) + \nabla F(z^k)d - w + v = 0$$

$$\ell \leq d + z^k \leq u, \quad w \geq 0, \quad v \geq 0$$

$$w^T(d + z^k - \ell) = v^T(u - d - z^k) = 0.$$

Conceptually, we are solving for d, but in practice MILES solves the linear problem in terms of the original variables $z = z^k + d$:

$$F(z^k) - \nabla F(z^k)z_k + \nabla F(z^k)z = w - v$$

$$\ell \leq z \leq u, \quad w \geq 0, \quad v \geq 0$$

$$w^T(z - \ell) = 0\ , \quad v^T(u - z) = 0.$$

After computing the solution z, MILES sets $d^k = z - z^k$.

The linear subproblem incorporates upper and lower bounds on any or all of the variables, assuring that the iterative sequence always remains within the bounds: ($\ell \leq z^k \leq u$). This can be helpful when, as is often the case, $F()$ is undefined for some $z \in R^n$.

Convergence of the Newton algorithm applied to MCP hinges on three questions:

1. Does the linearized problem always have a solution?
2. If the linearized problem has a solution, does Lemke's algorithm find it?
3. Is it possible to show that the computed direction d^k will provide an "improvement" in the solution?

Only for a limited class of functions $F()$ can all three questions be answered in the affirmative. For a much larger class of functions, the algorithm converges in practice but convergence is not "provable" [5]

The answer to question 3. depends on the choice of a norm by which an improvement is measured. The introduction of bounds and complementarity conditions makes the calculation of an error index more complicated. In MILES, the deviation associated with a candidate solution $z, \epsilon(z)$, is based on a measure of the extent to which z, w and v violate applicable upper and lower bounds and complementarity conditions.

[5] Kaneko (1978) provides some convergence theory for the linearized subproblem

5.25.3.1 Evaluating Convergence

Let δ_i^L and δ_i^U be indicator variables for whether z_i is off its lower or upper bound. These are defined as [6]

$$\delta_i^L = min(1, (z_i - \ell_i)^+) \quad \text{and} \quad \delta_i^U = min(1, (u_i - z_i)^+).$$

Given z, MILES uses the value of $F(z)$ to implicitly define the slack variables w and v:

$$w_i = F_i(z)^+ \ , \quad v_i = \Big(-F_i(z)\Big)^+ .$$

There are two components to the error term associated with index i, one corresponding to z_i's violation of upper and lower bounds:

$$\varepsilon_i^B = (z_i - u_i)^+ + (\ell_i - z_i)^+$$

and another corresponding to violations of complementarity conditions:

$$\varepsilon_i^C = \delta_i^L w_i + \delta_i^U v_i.$$

The error assigned to point z is then taken:

$$\varepsilon(z) = \|\varepsilon^B(z) + \varepsilon^C(z)\|_p$$

for a pre-specified value of $p = 1, 2$ or $+\infty$. [7]

5.25.4 Lemke's Method with Implicit Bounds

A mixed linear complementarity problem has the form:

$$\begin{array}{rl} \text{Given:} & M \in R^{n\times n}, \qquad q, \ell, u \in R^n \\ \text{Find:} & z, w, v \in R^n \\ \text{such that} & Mz + q = w - v, \\ & \ell \le z \le u, \quad w \ge 0, \quad v \ge 0, \\ & w^T(z - \ell) = 0 \ , \quad v^T(u - z) = 0. \end{array}$$

In the Newton subproblem at iteration k, the LCP data are given by $q = F(z^k) - \nabla F(z^k) z^k$ and $M = \nabla F(z^k)$.

[6] In the following $x^+ = max(x, 0)$

[7] Parameter p may be selected with input parameter 'NORM'. The default value for p is $+\infty$.

5.25.4.1 The Working Tableau

In MILES, the pivoting scheme for solving the linear problem works with a re-labeled linear system of the form:

```
\f[  B x^B + N x^N = q , \f]
```

where $x^B \in R^n$, $x^N \in R^{2n}$, and the tableau $[B|N]$ is a conformal "complementary permutation" of $[-M|\ I\ |-I]$. That is, every column i in B must either be the ith column of M, I or $-I$, while the corresponding columns i and $i+n$ in N must be the two columns which were not selected for B.

To move from the problem defined in terms of z, w and v to the problem defined in terms of x^B and x^N, we assign upper and lower bounds for the x^B variables as follows:

$$\underline{x}_i^B = \begin{cases} \ell_i, & \texttt{if} \quad x_i^B = z_i \\ 0, & \texttt{if} \quad x_i^B = w_i \ \texttt{or} \ v_i, \end{cases}$$

$$\overline{x}_i^B = \begin{cases} u_i, & \texttt{if} \quad x_i^B = z_i \\ \infty, & \texttt{if} \quad x_i^B = w_i \ \texttt{or} \ v_i \end{cases}$$

The values of the non-basic variables x_i^N and x_{i+n}^N are determined by the assignment of x_i^B:

$$x_i^B = \begin{cases} z_i \ \Rightarrow \ \begin{cases} x_i^N = w_i = 0 \\ x_{i+n}^N = v_i = 0 \end{cases} \\ w_i \ \Rightarrow \ \begin{cases} x_i^N = z_i = \ell_i \\ x_{i+n}^N = v_i = 0 \end{cases} \\ v_i \ \Rightarrow \ \begin{cases} x_i^N = w_i = 0 \\ x_{i+n}^N = z_i = u_i. \end{cases} \end{cases}$$

In words: if z_i is basic then both w_i and v_i equal zero. If z_i is non-basic at its lower bound, then w_i is possibly non-zero and v_i is non-basic at zero. If z_i is non-basic at its upper bound, then v_i is possibly non-zero and w_i is non-basic at zero.

Conceptually, we could solve the LCP by evaluating 3^n linear systems of the form:

$$x^B = B^{-1}\Big(q - Nx^N\Big).$$

Lemke's pivoting algorithm provides a procedure for finding a solution by sequentially evaluating some (hopefully small) subset of these 3^n alternative linear systems.

5.25.4.2 Initialization

Let B^0 denote the initial basis matrix [8] The initial values for basic variables are then:

$$\hat{x}^B = (B^0)^{-1}(q - N\ \hat{x}^N).$$

If $\underline{x}^B \leq \hat{x}^B \leq \bar{x}^B$, then the initial basis is feasible and the complementarity problem is solved [9] Otherwise, MILES introduces an artificial variable z_0 and an artificial column h. Basic variables are then expressed as follows:

$$x^B = \hat{x}^B - \tilde{h}z_0,$$

where $\tilde{h}$ is the "transformed artificial column" (the untransformed column is $h = B^0\tilde{h}$). The coefficients of $\tilde{h}$ are selected so that:

1. The values of "feasible" basis variables are unaffected by z_0: ($\underline{x}_i^B \leq x_i^B \leq \bar{x}_i^B \Longrightarrow \tilde{h}_i = 0$).
2. The "most infeasible" basic variable ($i = p$) is driven to its upper or lower bound when $z_0 = 1$:
$$\tilde{h}_p = \begin{cases} \hat{x}_p^B - \bar{x}_p^B, & if \quad \tilde{x}_p^B > \bar{x}_p^B \\ \hat{x}_p^B - \underline{x}_p^B, & if \quad \tilde{x}_p^B < \underline{x}_p^B \ . \end{cases}$$
3. All other *infeasible* basic variables assume values between their upper and lower bounds when z_0 increases to 1:
$$x_i^B = \begin{cases} 1 + \underline{x}_i^B, & if \quad \underline{x}_i^B > -\infty, \quad \bar{x}_i^B = +\infty \\ \frac{\bar{x}_i^B + \underline{x}_i^B}{2}, & if \quad \underline{x}_i^B > -\infty, \quad \bar{x}_i^B < +\infty \\ \bar{x}_i^B - 1, & if \quad \underline{x}_i^B = -\infty, \quad \bar{x}_i^B < +\infty \ . \end{cases}$$

5.25.4.3 Pivoting Rules

When z_0 enters the basis, it assumes a value of unity, and at this point (barring degeneracy), the subsequent pivot sequence is entirely determined. The entering variable in one iteration is determined by the exiting basic variable in the previous iteration. For example, if z_i were in B^0 and introducing z_0 caused z_i to move onto its lower bound, then the subsequent iteration introduces w_i. Conversely, if w_i were in B^0 and z_0 caused w_i to fall to zero, the subsequent iteration increases z_i from ℓ_i. Finally, if v_i were in B^0 and z_0's introduction caused v_i to fall to zero, the subsequent iteration *decreases* z_i from u_i.

Table 1 Pivot Sequence Rules for Lemke's Algorithm with Implicit Bounds

N	Exiting Variable	Entering Variable	Change in Non-basic Values
I	z_i at lower bound	w_i increases from 0	$x_i^N = z_i = \ell_i$
II	z_i at upper bound	v_i increases from 0	$x_{i+n}^N = z_i = u_i$
III	w_i at 0	z_i increases from ℓ_i	$x_i^N = x_{i+n}^N = 0$
IV	v_i at 0	z_i decreases from u_i	$x_i^N = x_{i+n}^N = 0$

[8] In Miles, B^0 is chosen using the initially assigned values for z. When $z_i \leq \ell_i$, then $x_i^B = w_i$; when $z_i \geq u_i$, then $x_i^B = v_i$; otherwise $x_i^B = z_i$.

[9] The present version of the code simply sets $B^0 = -I$ and $x^B = w$ when the user-specified basis is singular. A subsequent version of the code will incorporate the algorithm described by Anstreicher, Lee, and Rutherford [1992] for coping with singularity.

The full set of pivoting rules is displayed in Table 1. One difference between this algorithm and the original Lemke (type III) pivoting scheme (see Lemke (1965), Garcia and Zangwill (1981), or Cottle and Pang (1992)) is that structural variables (z_i's) may enter or exit the basis at their upper bound values. The algorithm, therefore, must distinguish between pivots in which the entering variable increases from a lower bound and those in which the entering variable decreases from an upper bound.

Another difference with the "usual" Lemke pivot procedure is that an entering structural variable may move from one bound to another. When this occurs, the subsequent pivot introduces the corresponding slack variable. For example, if z_i is increased from ℓ_i to u_i without driving a basic variable infeasible, then z_i becomes non-basic at u_i, and the subsequent pivot introduces v_i. This type of pivot may be interpreted as z_i entering and exiting the basis in a single step [10]

In theory it is convenient to ignore degeneracy, while in practice degeneracy is unavoidable. The present algorithm does not incorporate a lexicographic scheme to avoid cycling, but it does implement a ratio test procedure which assures that when there is more than one candidate, priority is given to the most stable pivot. The admissible set of pivots is determined on both an absolute pivot tolerance (`ZTOLPV`) and a relative pivot tolerance (`ZTOLRP`). No pivot with absolute value smaller than min (`ZTOLPV`, `ZTOLRP` $\|V\|$) is considered, where $\|V\|$ is the norm of the incoming column.

5.25.4.4 Termination on a Secondary Ray

Lemke's algorithm terminates normally when the introduction of a new variable drives z_0 to zero. This is the desired outcome, but it does not always happen. The algorithm may be interrupted prematurely when no basic variable "blocks" an incoming variable, a condition known as "termination on a secondary ray". In anticipation of such outcomes, MILES maintains a record of the value of z_0 for successive iterations, and it saves basis information associated with the smallest observed value, z_0^*. (In Lemke's algorithm, the pivot sequence is determined without regard for the effect on z_0, and the value of the artificial variable may follow an erratic (non-monotone) path from its initial value of one to the final value of zero.)

When MILES encounters a secondary ray, a restart procedure is invoked in which the set of basic variables associated with z_0^* are reinstalled. This basis (augmented with one column from the non-basic triplet to replace z_0) serves as B^0, and the algorithm is restarted. In some cases this procedure permits the pivot sequence to continue smoothly to a solution, while in other cases may only lead to another secondary ray.

5.25.5 The Options File

The standard GAMS options (e.g. iterlim, reslim) can be used to control MILES. For more details, see section Controlling a Solver via GAMS Options.

In addition, MILES-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

The following is a typical MILES options file:

```
ITLIMT = 50
CONTOL = 1.0E-8
LUSIZE = 16
```

In the remainder of this section we describe the MILES options and give their default values.

[10] If all structural variables are subject to finite upper and lower bounds, then no z_i variables may be part of a homogeneous solution adjacent to a secondary ray. This does not imply, however, that secondary rays are impossible when all z_i variables are bounded, as a ray may then be comprised of w_i and v_i variables.

Option	Description	Default
contol	convergence tolerance Whenever an iterate is encountered for which `epsilon(z)` < `CONTOL`, the algorithm terminates. This corresponds to the GAMS/MINOS parameter "Row tolerance".	`1e-6`
dens1	LUSOL: density to start searching maxcol columns The density at which the Markowitz strategy should search `maxcol` columns and no rows. Range: [`0`, `1`]	`0.3`
dens2	LUSOL: density to start searching 1 column The density at which the Markowitz strategy should search only 1 column or (preferably) use a dense LU for all the remaining rows and columns. Range: [`0`, `1`]	`0.6`
dmpfac	damping factor in backtracking linesearch	`0.5`
elmax1	LUSOL: max multiplier allowed in L during factor	`10`
elmax2	LUSOL: max multiplier allowed in L during update	`10`
factim	basis reinversion time Indicates the maximum number of CPU seconds between recalculation of the basis factors.	`120`
invfrq	basis reinversion frequency Determines the maximum number of Lemke iterations between recalculation of the basis factors. This corresponds to the GAMS/MINOS parameter "Factorization frequency".	`200`
invlog	toggle Lemke inversion logging A switch which requests LUSOL to generate a report with basis statistics following each refactorization.	`1`
itch	iteration refinement frequency Indicates the frequency with which the factorization is checked. The number refers to the number of basis replacement operations between refinements. This corresponds to the GAMS/MINOS parameter "Check frequency".	`25`
iterlim	minor (Lemke) iterations limit Can also be set with the GAMS IterLim option, and takes its default from that. This corresponds to the GAMS/MINOS parameter "Iterations Limit".	`GAMS IterLim`
itlimt	major (Newton) iterations limit An upper bound on the number of Newton iterations. This corresponds to the GAMS/MINOS parameter "Major iterations". Range: {`0`, ..., `1000`}	`100`
lcpdmp	LCP dump (post-scaling) A switch to generate a printout of the LCP data after scaling.	`0`
lcpech	LCP echo print (pre-scaling) A switch to generate a printout of the LCP data before scaling, as evaluated.	`0`
levout	set output level Sets the level of debug output written to the log and status files. The lowest meaningful value is -1 and the highest is 3. This corresponds, roughly, to the GAMS/MINOS parameter "Print level".	`1`

Option	Description	Default
lprint	LUSOL: print level Controls the amount of logging from the LUSOL routines. <0: suppresses output 0: gives error messages 1: gives debug output from some of the routines in LUSOL >=2: gives the pivot row and column and the no. of rows and columns involved at each elimination step in lu1fac	0
lusize	LUSOL: multiplier for estimating memory requirements Used to estimate the number of LU nonzeros which will be stored, as a multiple of the number of nonzeros in the Jacobian matrix. Range: {3, ..., ∞}	5
maxcol	LUSOL: max cols to search for pivot element Sets the maximum number of columns to search in a Markowitz-type search for the next pivot element. For some of the factorization, the number of rows searched is maxrow = maxcol - 1.	5
minstp	minimum step length in backtracking linesearch	0.01
norm	norm to use in measuring deviation Defines the vector norm to be used for evaluating epsilon(z). Acceptable values are 1, 2 or 3 which correspond to p = 1, 2 and +INF.	1
nrsmax	restart limit Sets an upper bound on the number of restarts which the linear subproblem solver will undertake before giving up.	1
pivlog	toggle Lemke pivot logging A switch to generate a status file listing of the Lemke pivot sequence.	0
plinfy	infinity used by the solver The value assigned for "plus infinity" ("+INF" in GAMS notation).	1e20
scale	turn on scaling at every iteration Invokes row and column scaling of the LCP tableau in every iteration. This corresponds, roughly, to the GAMS/MINOS switch "scale all variables".	1
small	LUSOL: absolute zero tolerance The absolute tolerance for treating reals as zero.	3.0e-13
uspace	LUSOL: factor limiting waste space in U The row or column lists are compressed if their length exceeds this value times the length of either file after the last compression. Range: [1, ∞]	3
utol1	LUSOL: absolute tolerance U diagonal The absolute tol for flagging small diagonals of U.	3.64e-11
utol2	LUSOL: relative tolerance U diagonal The relative tol for flagging small diagonals of U.	3.64e-11
ztolda	zero tolerance on matrix coefficients, default: sqrt(machEps)	1.48e-08
ztolpv	absolute zero tolerance on pivots, default: machEps**(2/3) The absolute pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter "Pivot tolerance" as it applies for nonlinear problems.	3.64e-11
ztolrp	relative zero tolerance on pivots, default: machEps**(2/3) The relative pivot tolerance. This corresponds, roughly, to the GAMS/MINOS parameter "Pivot tolerance" as it applies for nonlinear problems.	3.64e-11
ztolz0	absolute tolerance for installing covering vector elements	1e-6

Option	Description	Default
ztolze	feasibility tolerance Used in the subproblem solution to determine when any variable has exceeded an upper or lower bound. This corresponds to GAMS/MINOS parameter "Feasibility tolerance".	1e-6

5.25.6 Log File Output

The log file is intended for display on the screen in order to permit monitoring progress. Relatively little output is generated.

A sample iteration log is displayed in Table 2. This output is from two cases solved in succession. This and subsequent output comes from program `TRNSP.FOR` which calls the MILES library directly. (When MILES is invoked from within GAMS, at most one case is processed at a time.)

The first line of the log output gives the MILES program date and version information. This information is important for bug reports.

The line beginning "Work space..." reports the amount of memory which has been allocated to solve the model - 10K for this example. Thereafter is reported the initial deviation together with the name of the variable associated with the largest imbalance ($\epsilon_i^B + \epsilon_i^C$). The next line reports the convergence tolerance.

The lines beginning 0 and 1 are the major iteration reports for those iterations. the number following the iteration number is the current deviation, and the third number is the Armijo step length. The name of the variable complementary to the equation with the largest associated deviation is reported in parenthesis at the end of the line.

Following the final iteration is a summary of iterations, refactorizations, amd final deviation. The final message reports the solution status. In this case, the model has been successfully processed ("Solved.").

Table 2 Sample Iteration Log

```
MILES  (July 1993)                                        Ver:225-386-02

Thomas F. Rutherford
Department of Economics
University of Colorado

Technical support available only by Email:  TOM@GAMS.COM

Work space allocated           --    0.01 Mb

Initial deviation ........ 3.250E+02    P_01
Convergence tolerance .... 1.000E-06

   0    3.25E+02    1.00E+00 (P_01    )
   1    1.14E-13    1.00E+00 (W_02    )

Major iterations ........     1
Lemke pivots ............    10
Refactorizations ........     2
Deviation ............... 1.137E-13

Solved.
```

```
Work space allocated           --    0.01 Mb

Initial deviation ........ 5.750E+02    W_02
Convergence tolerance .... 1.000E-06

   0    5.75E+02    1.00E+00 (W_02    )
   1    2.51E+01    1.00E+00 (P_01    )
   2    4.53E+00    1.00E+00 (P_01    )
   3    1.16E+00    1.00E+00 (P_01    )
   4    3.05E-01    1.00E+00 (P_01    )
   5    8.08E-02    1.00E+00 (P_01    )
   6    2.14E-02    1.00E+00 (P_01    )
   7    5.68E-03    1.00E+00 (P_01    )
   8    1.51E-03    1.00E+00 (P_01    )
   9    4.00E-04    1.00E+00 (P_01    )
  10    1.06E-04    1.00E+00 (P_01    )
  11    2.82E-05    1.00E+00 (P_01    )
  12    7.47E-06    1.00E+00 (P_01    )
  13    1.98E-06    1.00E+00 (P_01    )
  14    5.26E-07    1.00E+00 (P_01    )

Major iterations ........    14
Lemke pivots ............    23
Refactorizations ........    15
Deviation ............... 5.262E-07

Solved.
```

5.25.7 Status File Output

The status file reports more details regarding the solution process than are provided in the log file. Typically, this file is written to disk and examined only if a problem arises. Within GAMS, the status file appears in the listing only following the GAMS statement `OPTION SYSOUT=ON;`.

The level of output to the status file is determined by the options passed to the solver. In the default configuration, the status file receives all information written to the log file together with a detailed listing of all switches and tolerances and a report of basis factorization statistics.

When output levels are increased from their default values using the options file, the status file can receive considerably more output to assist in debugging. Table 3 - Table 6 present a status file generated with `LEVOUT=3` (maximum), `PIVLOG=T`, and `LCPECH=T`.

The status file begins with the same header as the log file. Thereafter is a complete echo-print of the user-supplied option file when one is provided. Following the core allocation report is a full echo-print of control parameters, switches and tolerance as specified for the current run.

Table 4 continues the status file. The iteration-by- iteration report of variable and function values is produced whenever `LEVOUT >= 2`. Table 4 also contains an LCP echo-print. This report has two sections: $ROWS and $COLUMNS. The four columns of numbers in the $ROWS section are the constant vector (q), the current estimate of level values for the associated variables (z), and the lower and upper bounds vectors (ℓ and u). The letters L and U which appear between the ROW and Z columns are used to identify variables which are non-basic at their lower and upper bounds, respectively. In this example, all upper bounds equal $+\infty$, so no variables are non-basic at their upper bound.

By convention, only variable (and not equation names) appear in the status file. An equation is identified by the corresponding variable. We therefore see in the $COLUMNS: section of the matrix echo-print, the row names correspond to the names of z variables. The names assigned to variables z_i, w_i and v_i are

$z-$ <name i >, $w-$ <name i >, and $v-$ <name i >, as shown in the $COLUMNS section. The nonzeros for $w-$ <> and $v-$ <> variables are not shown because they are assumed to be $-/+I$.

The status file output continues on Table 5 where the first half of the table reports output from the matrix scaling procedure, and the second half reports the messages associated with initiation of Lemke's procedure.

The "lu6chk warning" is a LUSOL report. Thereafter are two factorization reports. Two factorizations are undertaken here because the first basis was singular, so the program installs all the lower bound slacks in place of the matrix defined by the initial values, z.

Following the second factorization report, at the bottom of Table 5 is a summary of the initial pivot. "Infeasible in 3 rows." indicates that $\tilde{h}$ contains 3 nonzero elements. "Maximum infeasibility" reports the largest amount by which a structural variable violates an upper or lower bound. "Artificial column with 3 elements." indicates that the vector $h = B^0\tilde{h}$ contains 3 elements (note that in this case $B^0 = -I$ because the initial basis was singular, hence the equivalence between the number of nonzeros in $\tilde{h}$ and h.).

Table 6 displays the final section of the status file. At the top of the table is the Lemke iteration log. The columns are interpreted as follows:

- `ITER` is the iteration index beginning with 0.
- `STATUS` is a statistic representing the efficiency of the Lemke path. Formally, status is the ratio of the minimum number of pivots from B_0 to the current basis divided by the actual number of pivots. When the status is 1, Lemke's algorithm is performing virtually as efficiently as a direct factorization (apart from the overhead of basis factor updates.)
- `Z%` indicates the percentage of columns in the basis are "structural" (z's).
- `Z0` indicates the value of the artificial variable. Notice that in this example, the artificial variable declines monotonically from its initial value of unity.
- `ERROR` is a column in which the factorization error is reported, when it is computed. For this run, ITCH=30 and hence no factorization errors are computed.
- `INFEAS` is a column in which the magnitude of the infeasibility introduced by the artificial column (defined using the box-norm) is reported. (In MILES the cover vector h contains many different nonzero values, not just 1's; so there may be a large difference between the magnitude of the artificial variable and the magnitude of the induced infeasibility.
- `PIVOTS` reports the pivot magnitude in both absolute terms (the first number) and relative terms (the second number). The relative pivot size is the ratio of the pivot element to the norm of the incoming column.
- `IN/OUT` report the indices (not names) of the incoming and outgoing columns for every iteration. Notice that Lemke's iteration log concludes with variable z_0 exiting the basis.

The convergence report for iteration 1 is no different from the report written to the log file, and following this is a second report of variable and function values. We see here that a solution has been obtained following a single subproblem. This is because the underlying problem is, in fact, linear.

The status file (for this case) concludes with an iteration summary identical to the log file report and a summary of how much CPU time was employed overall and within various subtasks. (Don't be alarmed if the sum of the last five numbers does not add up to the first number - some cycles are not monitored precisely.)

Table 3 Status File with Debugging Output (page 1 of 4)

```
MILES  (July 1993)                                        Ver:225-386-02
Thomas F. Rutherford
Department of Economics
University of Colorado

Technical support available only by Email:  TOM@GAMS.COM

User supplied option file:

>BEGIN
> PIVLOG = .TRUE.
> LCPECH = .TRUE.
> LEVOUT = 3
>END

Work space allocated           --    0.01 Mb

NEWTON algorithm control parameters:
    Major iteration limit ..(ITLIMT).          25
    Damping factor .........(DMPFAC).   5.000E-01
    Minimum step length ....(MINSTP).   1.000E-02
    Norm for deviation .....(NORM)...           3
    Convergence tolerance ..(CONTOL).   1.000E-06
LEMKE algorithm control parameters:
    Iteration limit .......(ITERLIM).        1000
    Factorization frequency (INVFRQ).         200
    Feasibility tolerance ..(ZTOLZE).   1.000E-06
    Coefficient tolerance ..(ZTOLDA).   1.483E-08
    Abs. pivot tolerance ...(ZTOLPV).   3.644E-11
    Rel. pivot tolerance ...(ZTOLRP).   3.644E-11
    Cover vector tolerance .(ZTOLZ0).   1.000E-06
    Scale every iteration ...(SCALE).   T
    Restart limit ..........(NRSMAX).           1
Output control switches:
    LCP echo print ...........(LCPECH).   F
    LCP dump .................(LCPDMP).   T
    Lemke inversion log ......(INVLOG).   T
    Lemke pivot log ......... (PIVLOG).   T
Initial deviation ........ 3.250E+02    P_01
Convergence tolerance .... 1.000E-06

================================
Convergence Report, Iteration  0
================================================================
ITER   DEVIATION        STEP
   0    3.25E+02    1.00E+00 (P_01    )
================================================================
```

Table 4 Status File with Debugging Output (page 2 of 4)

```
 Iteration   0 values.
               ROW             Z                      F
              --------    ------------           ------------
              X_01_01  L   0.00000E+00           -7.75000E-01
              X_01_02  L   0.00000E+00           -8.47000E-01
              X_01_03  L   0.00000E+00           -8.38000E-01
              X_02_01  L   0.00000E+00           -7.75000E-01
              X_02_02  L   0.00000E+00           -8.38000E-01
              X_02_03  L   0.00000E+00           -8.74000E-01
              W_01     L   0.00000E+00            3.25000E+02
              W_02     L   0.00000E+00            5.75000E+02
              P_01         1.00000E+00           -3.25000E+02
              P_02         1.00000E+00           -3.00000E+02
              P_03         1.00000E+00           -2.75000E+02
 ==================================
 Function Evaluation, Iteration:  0
 ==================================
$ROWS:
X_01_01    -2.25000000E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_01_02    -1.53000004E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_01_03    -1.61999996E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_01    -2.25000000E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_02    -1.61999996E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
X_02_03    -1.25999998E-01  0.00000000E+00  0.00000000E+00  1.00000000E+20
W_01       -3.25000000E+02  0.00000000E+00  0.00000000E+00  1.00000000E+00
W_02       -5.75000000E+02  0.00000000E+00  0.00000000E+00  1.00000000E+00
P_01        3.25000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
P_02        3.00000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
P_03        2.75000000E+02  1.00000000E+00  0.00000000E+00  1.00000000E+20
     ...    0.00000000E+00  0.00000000E+00  0.00000000E+00  0.00000000E+00
$COLUMNS:
Z-X_01_01   W_01     -1.00000000E+00
            P_01      1.00000000E+00
Z-X_01_02   W_01     -1.00000000E+00
            P_02      1.00000000E+00
Z-X_01_03   W_01     -1.00000000E+00
            P_03      1.00000000E+00
Z-X_02_01   W_02     -1.00000000E+00
            P_01      1.00000000E+00
Z-X_02_02   W_02     -1.00000000E+00
            P_02      1.00000000E+00
Z-X_02_03   W_02     -1.00000000E+00
            P_03      1.00000000E+00
Z-W_01      X_01_01   1.00000000E+00
            X_01_02   1.00000000E+00
            X_01_03   1.00000000E+00
Z-W_02      X_02_01   1.00000000E+00
            X_02_02   1.00000000E+00
            X_02_03   1.00000000E+00
Z-P_01      X_01_01  -1.00000000E+00
            X_02_01  -1.00000000E+00
Z-P_02      X_01_02  -1.00000000E+00
            X_02_02  -1.00000000E+00
Z-P_03      X_01_03  -1.00000000E+00
            X_02_03  -1.00000000E+00
       ...       ...  0.00000000E+00
```

Table 5 Status File with Debugging Output (page 3 of 4)

```
SCALING LCP DATA
-------
            MIN ELEM    MAX ELEM       MAX COL RATIO
AFTER  0    1.00E+00    1.00E+00                1.00
AFTER  1    1.00E+00    1.00E+00                1.00
AFTER  2    1.00E+00    1.00E+00                1.00
AFTER  3    1.00E+00    1.00E+00                1.00
SCALING RESULTS:

      A(I,J) <= A(I,J) * R(I) / C(J)

  ROW          ROW    Z COLUMN    W COLUMN    V COLUMN
    1       1.0000      1.0000      1.0000      1.0000
    2       1.0000      1.0000      1.0000      1.0000
    3       1.0000      1.0000      1.0000      1.0000
    4       1.0000      1.0000      1.0000      1.0000
    5       1.0000      1.0000      1.0000      1.0000
    6       1.0000      1.0000      1.0000      1.0000
    7       1.0000      1.0000      1.0000      1.0000
    8       1.0000      1.0000      1.0000      1.0000
    9       1.0000      1.0000      1.0000      1.0000
   10       1.0000      1.0000      1.0000      1.0000
   11       1.0000      1.0000      1.0000      1.0000

lu6chk  warning.  The matrix appears to be singular.
    nrank =      8        rank of U
n - nrank =      3        rank deficiency
    nsing =      3        singularities
    jsing =     10        last singular column
    dumax =   1.00E+00    largest  triangular diag
    dumin =   1.00E+00    smallest triangular diag

LUSOL 5.4  FACTORIZATION STATISTICS
Compressns    0    Merit      0.00    LenL          0    LenU         14
Increase   0.00    M            11    UT           11    D1            0
Lmax    0.0E+00    Bmax    1.0E+00    Umax    1.0E+00    Umin    1.0E+00
Growth  1.0E+00    LT            0    BP            0    D2            0

LUSOL 5.4  FACTORIZATION STATISTICS
Compressns    0    Merit      0.00    LenL          0    LenU         11
Increase   0.00    M            11    UT           11    D1            0
Lmax    0.0E+00    Bmax    1.0E+00    Umax    1.0E+00    Umin    1.0E+00
Growth  1.0E+00    LT            0    BP            0    D2            0

CONSTRUCTING ARTIFICIAL COLUMN

--- Infeasible in   3 rows.
--- Maximum infeasibility:    3.250E+02
--- Artificial column with   3 elements.
--- Pivoting in row:   9 to replace column  20
--- Pivot element:   -3.250E+02
```

Table 6 Status File with Debugging Output (page 4 of 4)

```
                        LEMKE PIVOT STEPS
```

```
                          =================
ITER  STATUS   Z%   Z0     ERROR   INFEAS.  ---- PIVOTS ----  IN      OUT
   1    1.00   0  1.000                    3.E+02 1.E+00  1  Z0      W    9
   2    1.00   9  1.000                    1.E+00 1.E+00  2  Z    9  W    1
   3    1.00  18  0.997                    9.E-01 9.E-01  1  Z    1  W   10
   4    1.00  27  0.997                    1.E+00 1.E+00  1  Z   10  W    2
   5    1.00  36  0.996                    9.E-01 4.E-01  1  Z    2  W   11
   6    1.00  45  0.996                    1.E+00 1.E+00  1  Z   11  W    6
   7    1.00  55  0.479                    2.E+00 1.E+00  1  Z    6  W    7
   8    1.00  64  0.479                    1.E+00 1.E+00  1  Z    7  W    4
   9    1.00  73  0.000                    1.E+00 1.E+00  2  Z    4  W    8
  10    1.00  73  0.000                    1.E-03 2.E-03  1  V    8  Z0

================================
Convergence Report, Iteration  1
================================================================
ITER   DEVIATION        STEP
   0    3.25E+02    1.00E+00
   1    1.14E-13    1.00E+00 (W_02    )
================================================================

Iteration   1 values.

               ROW             Z                    F
              --------    ------------         ------------
              X_01_01      2.50000E+01         -8.32667E-17
              X_01_02      3.00000E+02         -5.55112E-17
              X_01_03  L   0.00000E+00          3.60000E-02
              X_02_01      3.00000E+02         -8.32667E-17
              X_02_02  L   0.00000E+00          8.99999E-03
              X_02_03      2.75000E+02          2.77556E-17
              W_01         1.00000E+00         -1.13687E-13
              W_02         1.00000E+00          1.13687E-13
              P_01         1.22500E+00          0.00000E+00
              P_02         1.15300E+00          0.00000E+00
              P_03         1.12600E+00          0.00000E+00

Major iterations ........     1
Lemke pivots ............    10
Refactorizations ........     2
Deviation ............... 1.137E-13
Solved.

Total solution time .:        0.6 sec.
Function & Jacobian..:        0.2 sec.
LCP solution ........:        0.2 sec.
Refactorizations ....:        0.1 sec.
FTRAN ...............:        0.0 sec.
Update ..............:        0.1 sec.
```

5.25.8 Termination Messages

Basis factorization error in INVERT. An unexpected error code returned by LUSOL. This should normally not occur. Examine the status file for a message from LUSOL [11] .

[11] Within GAMS, insert the line "OPTION SYSOUT=ON;" prior to the solve statement and resubmit the program in order to pass the MILES solver status file through to the listing.

Failure to converge. Two successive iterates are identical - the Newton search direction is not defined. This should normally not occur.

Inconsistent parameters `ZTOLZ0`**,** `ZTOLZE`**.** `ZTOLZ0` determines the smallest value loaded into the cover vector h, whereas `ZTOLZE` is the feasibility tolerance employed in the Harris pivot selection procedure. If `ZTOLZ0` $<$ `-ZTOLZE`, Lemke's algorithm cannot be executed because the initial basis is infeasible.

Insufficient space for linearization. Available memory is inadequate for holding the nonzeros in the Jacobian. More memory needs to be allocated. If there is insufficient space for the {Jacobi} matrix, there is far too little memory for holding the LU factors of the same matrix.

Insufficient space to invert. More memory needs to be allocated for basis factors. Increase the value of `LUSIZE` in the options file, or assign a larger value to `<model>.workspace`.

Iteration limit exceeded. This can result from either exceeding the major (Newton) or minor (Lemke) iterations limit. When MILES is invoked from GAMS, the cumulative Lemke iteration limit can be set with the statement `<model>.iterlim = xx};`. The Newton iteration limit is 100 by default, and it can be modified only through the `ITLIMT` option.

Resource interrupt. Elapsed CPU time exceeds options parameter `RESLIM`. To increase this limit, either add `RESLIM` = xxx in the options file or add a GAMS statement `<model>.RESLIM = xxx;`.

Singular matrix encountered. Lemke's algorithm has been interrupted due to a singularity arising in the basis factorization, either during a column replacement or during a refactorization. For some reason, a restart is not possible.

Termination on a secondary ray. Lemke's algorithm terminated on a secondary ray. For some reason, a restart is not possible.

Unknown termination status. The termination status flag has not been set, but the code has interrupted. Look in the status file for a previous message. This termination code should not happen often.

5.25.9 References

K.J. Anstreicher, J. Lee and T.F. Rutherford "Crashing a Maximum Weight Complementary Basis", Mathematical Programming. (1992)

A. Brooke, D. Kendrick, and A. Meeraus "GAMS: A User's Guide", Scientific Press, (1987).

R.W. Cottle and J.S. Pang "The Linear Complementarity Problem", Academic Press, (1992).

J.E. Dennis and R.B. Schnabel "Numerical Methods for Unconstrained Optimization and Nonlinear Equations", Prentice-Hall (1983).

S. Dirkse "Robust solution of mixed complementarity problems", Computer Sciences Department, University of Wisconsin (1992).

B.C. Eaves, "A locally quadratically convergent algorithm for computing stationary points," Tech. Rep., Department of Operations Research, Stanford University, Stanford, CA (1978).

P.E. Gill, W. Murray, M.A. Saunders and M.H. Wright "Maintaining LU factors of a general sparse matrix", Linear Algebra and its Applications 88/89, 239-270 (1991).

C.B. Garcia and W.I. Zangwill "Pathways to Solutions, Fixed Points, and Equilibria", Prentice-Hall (1981)

P. Harker and J.S. Pang "Finite-dimensional variational inequality and nonlinear complementarity problems: a survey of theory, algorithms and applications", Mathematical Programming 48, pp. 161-220 (1990).

W.W. Hogan, "Energy policy models for project independence," Computation and Operations Research 2 (1975) 251-271.

N.H. Josephy, "Newton's method for generalized equations", Technical Summary Report #1965, Mathematical Research Center, University of Wisconsin - Madison (1979).

I. Kaneko, "A linear complementarity problem with an n by 2n 'P'- matrix", Mathematical Programming Study 7, pp. 120-141, (1978).

C.E. Lemke "Bimatrix equilibrium points and mathematical programming", Management Science 11, pp. 681-689, (1965).

L. Mathiesen, "Computation of economic equilibria by a sequence of linear complementarity problems", Mathematical Programming Study 23 (1985).

P.V. Preckel, "NCPLU Version 2.0 User's Guide", Working Paper, Department of Agricultural Economics, Purdue University, (1987).

W.H. Press, B.P.Flannery, S.A. Teukolsky, W.T. Vetterling "Numerical Recipes: The Art of Scientific Computing", Cambridge University Press (1986).

J.M. Ortega and W.C. Rheinboldt, "Iterative Solution of Nonlinear Equations in Several Variables", Academic Press (1970).

S.M. Robinson, "A quadratically-convergent algorithm for general nonlinear programming problems", Mathematical Programming Study 3 (1975).

T.F. Rutherford "Extensions of GAMS for variational and complementarity problems with applications in economic equilibrium analysis", Working Paper 92-7, Department of Economics, University of Colorado (1992a).

T.F. Rutherford "Applied general equilibrium modeling using MPS/GE as a GAMS subsystem", Working Paper 92-15, Department of Economics, University of Colorado (1992b).

Table 7 Transport Model in GAMS/MCP (page 1 of 2)

```
*==>TRNSP.GMS
option mcp=miles;


$TITLE  LP TRANSPORTATION PROBLEM FORMULATED AS A ECONOMIC EQUILIBRIUM
*       ===============================================================
*       In this file, Dantzig's original transportation model is
*       reformulated as a linear complementarity problem.  We first
*       solve the model with fixed demand and supply quantities, and
*       then we incorporate price-responsiveness on both sides of the
*       market.
*
*       T.Rutherford  3/91  (revised 5/91)
*       ===============================================================
*       This problem finds a least cost shipping schedule that meets
*       requirements at markets and supplies at factories
*
*       References:
*              Dantzig, G B., Linear Programming and Extensions
*              Princeton University Press, Princeton, New Jersey, 1963,
*              Chapter 3-3.
*
*              This formulation is described in detail in Chapter 2
*              (by Richard E. Rosenthal) of GAMS: A Users' Guide.
*              (A Brooke, D Kendrick and A Meeraus, The Scientific Press,
*              Redwood City, California, 1988.
*
   SETS
        I   canning plants   / SEATTLE, SAN-DIEGO /
        J   markets          / NEW-YORK, CHICAGO, TOPEKA / ;

   PARAMETERS
       A(I)  capacity of plant i in cases (when prices are unity)
         /    SEATTLE     325
              SAN-DIEGO   575  /,

       B(J)  demand at market j in cases (when prices equal unity)
         /    NEW-YORK    325
              CHICAGO     300
              TOPEKA      275  /,

       ESUB(J)  Price elasticity of demand (at prices equal to unity)
         /    NEW-YORK    1.5
              CHICAGO     1.2
              TOPEKA      2.0  /;

   TABLE D(I,J)  distance in thousands of miles
                     NEW-YORK       CHICAGO      TOPEKA
       SEATTLE          2.5           1.7          1.8
       SAN-DIEGO        2.5           1.8          1.4  ;

  SCALAR F  freight in dollars per case per thousand miles  /90/ ;

  PARAMETER C(I,J)  transport cost in thousands of dollars per case ;

            C(I,J) = F * D(I,J) / 1000 ;

  PARAMETER PBAR(J) Reference price at demand node J;
```

Table 8 Transport Model in GAMS/MCP (page 2 of 2)

```
  POSITIVE  VARIABLES
       W(I)             shadow price at supply node i,
       P(J)             shadow price at demand node j,
       X(I,J)           shipment quantities in cases;

  EQUATIONS
       SUPPLY(I)        supply limit at plant i,
       FXDEMAND(J)      fixed demand at market j,
       PRDEMAND(J)      price-responsive demand at market j,
       PROFIT(I,J)      zero profit conditions;

PROFIT(I,J)..   W(I) + C(I,J)   =G= P(J);

SUPPLY(I)..     A(I) =G= SUM(J, X(I,J));

FXDEMAND(J)..   SUM(I, X(I,J))  =G=  B(J);

PRDEMAND(J)..   SUM(I, X(I,J))  =G=  B(J) * (PBAR(J)/P(J))**ESUB(J);

*       Declare models including specification of equation-variable association:
  MODEL FIXEDQTY / PROFIT.X, SUPPLY.W, FXDEMAND.P/ ;
  MODEL EQUILQTY / PROFIT.X, SUPPLY.W, PRDEMAND.P/ ;

*       Initial estimate:

  P.L(J) = 1;    W.L(I) = 1;

  PARAMETER REPORT(*,*,*)  Summary report;

  SOLVE FIXEDQTY USING MCP;
  REPORT("FIXED",I,J) = X.L(I,J);   REPORT("FIXED","Price",J) = P.L(J);
  REPORT("FIXED",I,"Price") = W.L(I);

*       Calibrate the demand functions:

  PBAR(J) = P.L(J);

*       Replicate the fixed demand equilibrium:

  SOLVE EQUILQTY USING MCP;

  REPORT("EQUIL",I,J) = X.L(I,J);  REPORT("EQUIL","Price",J) = P.L(J);
  REPORT("EQUIL",I,"Price") = W.L(I);

  DISPLAY "BENCHMARK CALIBRATION", REPORT;

*       Compute a counter-factual equilibrium:

  C("SEATTLE","CHICAGO") = 0.5 * C("SEATTLE","CHICAGO");

  SOLVE FIXEDQTY USING MCP;
  REPORT("FIXED",I,J) = X.L(I,J);  REPORT("FIXED","Price",J) = P.L(J);
  REPORT("FIXED",I,"Price") = W.L(I);

*       Replicate the fixed demand equilibrium:
```

```
SOLVE EQUILQTY USING MCP;
REPORT("EQUIL",I,J) = X.L(I,J);  REPORT("EQUIL","Price",J) = P.L(J);
REPORT("EQUIL",I,"Price") = W.L(I);

DISPLAY "Reduced Seattle-Chicago transport cost:", REPORT;
```

5.26 MINOS and QUADMINOS

Bruce A. Murtagh; Graduate School of Management, Macquarie University, Sydney, Australia

Michael A. Saunders, Walter Murray; Department of EESOR, Stanford University, CA

Philip E. Gill; Department of Mathematics, University of California, San Diego, La Jolla, CA

5.26.1 Introduction

This document describes the GAMS interface to MINOS which is a general purpose nonlinear programming solver. For a quad-precision MINOS, see quadMINOS.

GAMS/MINOS is a specially adapted version of the solver that is used for solving linear and nonlinear programming problems in a GAMS environment.

GAMS/MINOS is designed to find solutions that are *locally optimal*. The nonlinear functions in a problem must be *smooth* (i.e., their first derivatives must exist).The functions need not be separable. Integer restrictions cannot be imposed directly.

A certain region is defined by the linear constraints in a problem and by the bounds on the variables. If the nonlinear objective and constraint functions are convex within this region, any optimal solution obtained will be a *global optimum*. Otherwise there may be several local optima, and some of these may not be global. In such cases the chances of finding a global optimum are usually increased by choosing a staring point that is "sufficiently close", but there is no general procedure for determining what "close" means, or for verifying that a given local optimum is indeed global.

Linearly constrained models are solved with a very efficient and reliable reduced gradient technique that takes advantage of the sparsity of the model. Models with nonlinear constraints are solved with a method that iteratively solves subproblems with linearized constraints and an augmented Lagrangian objective function. This iterative scheme implies that only the final, optimal solution is sure to be feasible w.r.t the nonlinear constraints. This is in contrast to the feasible path method used by some other NLP solvers, e.g., CONOPT. MINOS and CONOPT are very complementary to each other as they employ very different algorithms. See MINOS vs CONOPT for a comparison of the two solvers.

GAMS allows you to specify values for many parameters that control GAMS/MINOS, and with careful experimentation you may be able to influence the solution process in a helpful way. All MINOS options available through GAMS/MINOS are summarized at the end of this document.

5.26.2 How to Run a Model with GAMS/MINOS

MINOS is capable of solving many types of models, including LP, NLP, DNLP and QCP. If MINOS is not specified as the default solver for the desired model type (e.g. NLP), then the following statement can be used in your GAMS model to select MINOS:

```
option nlp=minos;
```

The option statement should appear before the `solve` statement.

To be complete, we mention that the solver can be also specified on the command line, as in:

```
> gams camcge nlp=minos
```

This will override the global default, but if an algorithm option has been specified inside the model, then that specification takes precedence.

5.26.3 Overview of GAMS/MINOS

GAMS/MINOS is a system designed to solve large-scale optimization problems expressed in the following form:

$$\begin{array}{lllll} \text{NLP}: & & \text{minimize} & F(x) + c^T x + d^T y & (1) \\ & & \text{subject to} & f(x) + A_1 y \sim b_1 & (2) \\ & & & A_2 x + A_3 y \sim b_2 & (3) \\ & & & \ell \leq \begin{pmatrix} x \\ y \end{pmatrix} \leq u & (4) \end{array}$$

where the vectors c, d, b_1, b_2, ℓ, u and the matrices A_1, A_2, A_3 are constant, $F(x)$ is a smooth scalar function, and $f(x)$ is a vector of smooth functions. The $\sim$ signs mean that individual constraints may be defined using $\leq$, $=$ or $\geq$ corresponding to the GAMS constructs `=L=` , `=E=` and `=G=`.

The components of x are called the nonlinear variables, and the components of y are the linear variables. Similarly, the equations in (2) are called the nonlinear constraints, and the equations in (3) are the linear constraints. Equations (2) and (3) together are called the general constraints.

Let m_1 and n_1 denote the number of nonlinear constraints and variables, and let m and n denote the total number of (general) constraints and variables. Thus, A_3 has $m - m_1$ rows and $n - n_1$ columns. The constraints (4) specify upper and lower bounds on all variables. These are fundamental to many problem formulations and are treated specially by the solution algorithms in GAMS/MINOS. Some of the components of ℓ and u may be $-\infty$ or $+\infty$ respectively, in accordance with the GAMS use of `-INF` and `+INF`.

The vectors b_1 and b_2 are called the right-hand side, and together are denoted by b.

5.26.3.1 Linear Programming

If the functions $F(x)$ and $f(x)$ are absent, the problem becomes a *linear program*. Since there is no need to distinguish between linear and nonlinear variables, we use x rather than y. GAMS/MINOS converts all general constraints into equalities, and the only remaining inequalities are simple bounds on the variables. Thus, we write linear programs in the form

$$\begin{array}{lll} \text{LP}: & \text{minimize} & c^T x \\ & \text{subject to} & Ax + Is = 0 \\ & & \ell \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u \end{array}$$

where the elements of x are your own GAMS variables, and s is a set of *slack variables*: one for each general constraint. For computational reasons, the right-hand side b is incorporated into the bounds on s.

In the expression $Ax + Is = 0$ we write the identity matrix explicitly if we are concerned with columns of the associated matrix $\begin{pmatrix} A & I \end{pmatrix}$. Otherwise we will use the equivalent notation $Ax + s = 0$.

GAMS/MINOS solves linear programs using a reliable implementation of the *primal simplex method* [41] , in which the constraints $Ax + Is = 0$ are partitioned into the form

$$Bx_B + Nx_N = 0,$$

where the *basis matrix* is square and nonsingular. The elements of x_B and x_N are called the basic or nonbasic variables respectively. Together they are a permutation of the vector

$$\begin{pmatrix} x \\ s \end{pmatrix}.$$

Normally, each nonbasic variable is equal to one of its bounds, and the basic variables take on whatever values are needed to satisfy the general constraints. (The basic variables may be computed by solving the linear equations $Bx_B = Nx_N$.) It can be shown that if an optimal solution to a linear program exists, then it has this form.

The simplex method reaches such a solution by performing a sequence of *iterations*, in which one column of B is replaced by one column of N (and vice versa), until no such interchange can be found that will reduce the value of $c^T x$.

As indicated nonbasic variables usually satisfy their upper and lower bounds. If any components of x_B lie significantly outside their bounds, we say that the current point is *infeasible*. In this case, the simplex method uses a Phase 1 procedure to reduce the sum of infeasibilities to zero. This is similar to the subsequent Phase 2 procedure that optimizes the true objective function $c^T x$.

If the solution procedures are interrupted, some of the nonbasic variables may lie strictly *between* their bounds $\ell_j < x_j < u_j$. In addition, at a "feasible" or "optimal" solution, some of the basic variables may lie slightly outside their bounds: $\ell_j - \delta < x_j < \ell_j$ or $u_j < x_j < u_j + \delta$ where δ is a *feasibility tolerance* (typically 10^{-6}). In rare cases, even nonbasic variables might lie outside their bounds by as much as δ.

GAMS/MINOS maintains a sparse LU factorization of the basis matrix B, using a Markowitz ordering scheme and Bartels-Golub updates, as implemented in the Fortran package LUSOL [82] (see [16] [17] [147] [148]). The basis factorization is central to the efficient handling of sparse linear and nonlinear constraints.

5.26.3.2 Problems with a Nonlinear Objective

When nonlinearities are confined to the term $F(x)$ in the objective function, the problem is a linearly constrained nonlinear program. GAMS/MINOS solves such problems using a *reduced-gradient* algorithm [203] combined with a *quasi-Newton* algorithm that is described in [139] . In the reduced-gradient method, the constraints $Ax + Is = 0$ are partitioned into the form

$$Bx_B + Sx_S + Nx_N = 0$$

where x_s is a set of *superbasic variables*. At a solution, the basic and superbasic variables will lie somewhere between their bounds (to within the feasibility tolerance δ, while nonbasic variables will normally be equal to one of their bounds, as before. Let the number of superbasic variables be s, the number of columns in S. (The context will always distinguish s from the vector of slack variables.) At a solution, s will be no more than n_1, the number of nonlinear variables. In many practical cases we have found that s remains reasonably small, say 200 or less, even if n_1 is large.

In the reduced-gradient algorithm, x_s is regarded as a set of "independent variables" or "free variables" that are allowed to move in any desirable direction, namely one that will improve the value of the objective function (or reduce the sum of infeasibilities). The basic variables can then be adjusted in order to continue satisfying the linear constraints.

If it appears that no improvement can be made with the current definition of B, S and N, some of the nonbasic variables are selected to be added to S, and the process is repeated with an increased value of s. At all stages, if a basic or superbasic variable encounters one of its bounds, the variable is made nonbasic and the value of s is reduced by one.

A step of the reduced-gradient method is called a *minor iteration*. For linear problems, we may interpret the simplex method as being the same as the reduced-gradient method, with the number of superbasic variable oscillating between 0 and 1.

A certain matrix Z is needed now for descriptive purposes. It takes the form

$$\begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix}$$

though it is never computed explicitly. Given an LU factorization of the basis matrix B, it is possible to compute products of the form Zq and Z^Tg by solving linear equations involving B or B^T. This in turn allows optimization to be performed on the superbasic variables, while the basic variables are adjusted to satisfy the general linear constraints.

An important feature of GAMS/MINOS is a stable implementation of a quasi-Newton algorithm for optimizing the superbasic variables. This can achieve superlinear convergence during any sequence of iterations for which the B, S, N partition remains constant. A *search direction* q for the superbasic variables is obtained by solving a system of the form

$$R^TRq = -Z^Tg$$

where g is a gradient of $F(x)$, Z^Tg is the *reduced gradient*, and R is a dense upper triangular matrix. GAMS computes the gradient vector g analytically, using automatic differentiation. The matrix R is updated in various ways in order to approximate the *reduced Hessian* according to $R^TR \approx Z^THZ$ where H is the matrix of second derivatives of $F(x)$ (the *Hessian*).

Once q is available, the search direction for all variables is defined by $p = Zq$. A *ine search* is then performed to find an approximate solution to the one-dimensional (w.r.t. α) problem

$$\begin{aligned} &\text{minimize } F(x + \alpha p) \\ &\text{subject to } 0 < \alpha < \beta \end{aligned}$$

where β is determined by the bounds on the variables. Another important piece in GAMS/MINOS is a step-length procedure used in the linesearch to determine the step-length α (see [80]). The number of nonlinear function evaluations required may be influenced by setting the **Linesearch tolerance**, as discussed in Section Detailed Description of MINOS Options .

As in a linear programming solver, an equation $B^T\pi = gB$ is solved to obtain the *dual variables* or *shadow prices* π where gB is the gradient of the objective function associated with basic variables. It follows that $gB - B^T\pi = 0$. The analogous quantity for superbasic variables is the reduced-gradient vector $Z^T g = gs - s^T\pi$; this should also be zero at an optimal solution. (In practice its components will be of order $r||\pi||$ where r is the optimality tolerance, typically 10^{-6}, and $||\pi||$ is a measure of the size of the elements of π.)

5.26.3.3 Problems with Nonlinear Constraints

If any of the constraints are nonlinear, GAMS/MINOS employs a *project Lagrangian* algorithm, based on a method due to [150] , see [140] . This involves a sequence of *major iterations*, each of which requires the solution of a *linearly constrained subproblem*. Each subproblem contains linearized versions of the nonlinear constraints, as well as the original linear constraints and bounds.

At the start of the k^{th} major iteration, let x_k be an estimate of the nonlinear variables, and let λ_k be an estimate of the Lagrange multipliers (or dual variables) associated with the nonlinear constraints. The constraints are linearized by changing $f(x)$ in equation (2) to its linear approximation:

$$f'(x, x_k) = f(x_k) + J(x_k)(x - x_k)$$

or more briefly

$$f' = f_k + J_k(x - x_k)$$

where $J(x_k)$ is the *Jacobian matrix* evaluated at x_k. (The i-th row of the Jacobian is the gradient vector of the i-th nonlinear constraint function. As with the objective gradient, GAMS calculates the Jacobian using automatic differentiation).

The subproblem to be solved during the k-th major iteration is then

$$\begin{aligned}
&\text{minimize} && F(x) + c^T x + d^T y - \lambda_k^T(f - f') + 0.5\rho(f - f')^T(f - f') && (5)\\
&\text{subject to} && f' + A_1 y \sim b_1 && (6)\\
& && A_2 x + A_3 y \sim b_2 && (7)\\
& && \ell \le \begin{pmatrix} x \\ y \end{pmatrix} \le u && (8)
\end{aligned}$$

The objective function (5) is called an *augmented Lagrangian*. The scalar ρ is a *penalty parameter*, and the term involving ρ is a modified *quadratic penalty function*.

GAMS/MINOS uses the reduced-gradient algorithm to minimize (5) subject to (6) – (8). As before, slack variables are introduced and b_1 and b_2 are incorporated into the bounds on the slacks. The linearized constraints take the form

$$\begin{pmatrix} J_k & A_1 \\ A_2 & A_3 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} I & 0 \\ 0 & I \end{pmatrix}\begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} J_k x_k - f_k \\ 0 \end{pmatrix}$$

This system will be referred to as $Ax + Is = 0$ as in the linear case. The Jacobian J_k is treated as a sparse matrix, the same as the matrices A_1, A_2, and A_3.

In the output from GAMS/MINOS, the term *Feasible subproblem* indicates that the *linearized constraints* have been satisfied. In general, the nonlinear constraints are satisfied only in the limit, so that *feasibility* and *optimality* occur at essentially the same time. The nonlinear constraint violation is printed every major iteration. Even if it is zero early on (say at the initial point), it may increase and perhaps fluctuate before tending to zero. On "well behaved problems", the constraint violation will decrease quadratically (i.e., very quickly) during the final few major iterations.

5.26.4 Modeling Issues

Formulating nonlinear models requires that the modeler pays attention to some details that play no role when dealing with linear models.

5.26.4.1 Starting Points

The first issue is specifying a *starting point*. It is advised to specify a good starting point for as many nonlinear variables as possible. The GAMS default of zero is often a very poor choice, making this even more important.

As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points (x_i, y_i):

$$\begin{array}{lll} \text{Example :} & \text{minimize} & r \\ & \text{subject to} & (x_i - a)^2 + (y_i - b)^2 \leq r^2,\ r \geq 0. \end{array}$$

This problem can be modeled in GAMS as follows.

```
set i 'points' /p1*p10/;

parameters
    x(i)    'x coordinates',
    y(i)    'y coordinates';

* fill with random data
x(i) = uniform(1,10);
y(i) = uniform(1,10);

variables
    a       'x coordinate of center of circle'
    b       'y coordinate of center of circle'
    r       'radius';

equations
    e(i)    'points must be inside circle';


e(i).. sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);

r.lo = 0;

model m /all/;
option nlp=minos;
solve m using nlp minimizing r;
```

Without help, MINOS will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

$$\begin{aligned} x_{\min} &= \min_i x_i, \\ y_{\min} &= \min_i y_i, \\ x_{\max} &= \max_i x_i, \\ y_{\max} &= \max_i y_i, \end{aligned}$$

then good estimates are

$$\begin{aligned} a &= (x_{\min} + x_{\max})/2, \\ b &= (y_{\min} + y_{\max})/2, \\ r &= \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \end{aligned}$$

Thus we include in our model:

```
parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, y(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqr(a.l-xmin) + sqr(b.l-ymin) );
```

and now the model solves very easily.

Level values can also be set away from zero implicitly as a result of assigning bounds. When a variable is bounded away from zero, for instance by the statement `Y.LO = 1;`, the implicit projection of variable levels onto their bounds that occurs when a model is solved will initialize `Y` away from zero.

5.26.4.2 Bounds

Setting appropriate bounds can be very important to steer the algorithm away from uninteresting areas, and to prevent *function evaluation errors* from happening.

If your model contains a real power of the form `x**y` it is important to add a bound $x > 0.001$, as real exponentation is evaluated in GAMS as $\exp(y \log(x))$. In some cases one cannot write a bound directly, e.g. if the equation is $z = x^{f(y)}$. In that case it is advised to introduce an extra variable and equation:

$$\begin{aligned} z &= x^{\vartheta} \\ \vartheta &= f(y) \\ \vartheta &\geq \varepsilon \end{aligned}$$

(Note that the functions `SQR(x)` and `POWER(x,k)` are integer powers and do not require x to be positive).

If the model produces *function evaluation errors* adding bounds is prefered to raising the `DOMLIM` limit.

Bounds in GAMS are specified using `X.LO(i)=0.001` and `X.UP(i) = 1000`.

5.26.4.3 Scaling

Although MINOS has some facilities to scale the problem before starting to optimize it, it remains an important task for the modeler to provide a well-scaled model. This is especially the case for nonlinear models. GAMS has special syntax features to specify row and column scales that allow the modeler to keep the equations in a most natural form. For more information consult the GAMS User's Guide.

5.26.4.4 The Objective Function

The first step GAMS/MINOS performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. MINOS however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```
obj.. z =e= sum(i, sqr(resid(i)));

model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form NLP (equations $(1)-(4)$) by saying minimize z subject to $z = \sum_i resid_i^2$ and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize $\sum_i resid_i^2$ directly. This can be achieved by a simple reformulation: z can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable z is a free continuous variable (no bounds are defined on z),
- z appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- z is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by MINOS. For instance the model **[CHEM]** from the model library solves in 21 iterations. When we add the bound

```
energy.lo = 0;
```

to the objective variable `energy` and thus prevent it from being substituted out, MINOS will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```
Variables x,y,z;
Equations e1,e2;

e1..z =e= y;
e2..y =e= sqr(1+x);

model m /all/;

option nlp=minos;
solve m using nlp minimizing z;
```

will be reformulated as an *unconstrained* optimization problem

$$\text{minimize} \quad \text{f(x)} \quad = (1+x)^2.$$

These additional reformulations can be turned off by using the statement `option reform = 0;` (see Section GAMS Options).

5.26.5 GAMS Options

The standard GAMS options (e.g. iterlim, domlim) can be used to control GAMS/MINOS. For more details, see section Controlling a Solver via GAMS Options. We highlight some of the details of this usage below for cases of special interest.

iterlim

Sets the *minor* iteration limit. MINOS will stop as soon as the number of *minor iterations* exceeds the iteration limit and report the current solution.

domlim

Sets the domain error limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate $\sqrt{x}$ for $x < 0$. Other examples include taking logs of negative numbers, and evaluating the real power x^y for $x < \varepsilon$ (x^y is evaluated as $\exp(y \log x)$). When such an error occurs the count of domain errors is incremented: MINOS will stop if this count exceeds the limit. If the limit has not been reached, reasonable estimates for the function (and derivatives, if requested) are returned and MINOS continues. For example, in the case of $\sqrt{x}, x < 0$ a zero is passed back for the function value and a large value for the derivative. In many cases MINOS will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don't occur. For example, when an expression $\log(x)$ is present in the model, add a statement like `x.lo = 0.001;`.

bratio

Ratio used in basis acceptance test. When a previous solution or solution estimate exists, GAMS automatically passes this solution to MINOS so that it can reconstruct an advanced basis. When too many new (i.e. uninitialized with level and/or marginal values) variables or constraints enter the model, it may be better not to use existing basis information, but to instead *crash* a new basis. The `bratio` determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis.

workfactor

By default, GAMS/MINOS computes an estimate of the amount of workspace needed by MINOS, and passes this workspace on to MINOS for use in solving the model. This estimate is based on the model statistics: number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes, etc. In most cases this is sufficient to solve the model. In some rare cases MINOS may need more memory, and the user can provide this by specifying a value of `workfactor` greater than 1. The computed memory estimate is multiplied by the workfactor to determine the amount of workspace made available to MINOS for the solve.

workspace

The `workspace` option is *deprecated*: use the `workfactor` option instead. The `workspace` option specifies the amount of memory, in MB, that MINOS will use.

reform

This option controls the objective reformulation mechanism described in Section The Objective Function The default value of 100 will cause GAMS/MINOS to try further substitutions along the diagonal after the objective variable has been removed. Any other value will disable this diagonal procedure.

5.26.6 Summary of MINOS Options

The performance of GAMS/MINOS is controlled by a number of parameters or "options." Each option has a default value that should be appropriate for most problems. For special situations it is possible to specify non-standard values for some or all of the options through the MINOS option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

Note that the option file is not case sensitive. Examples for using the option file can be found at the end of this section. The tables below contain summary information about the MINOS options, default values, and links to more detailed explanations.

5.26.6.1 Output related options

Option	Description	Default
debug level	Controls amount of debug information written	0
log frequency	Controls iteration logging to listing file	100
print level	Controls amount of information printed during optimization	0
scale print	Print scaling factors	
solution	Prints MINOS solution	NO
summary frequency	Controls iteration logging to summary (log file)	100

5.26.6.2 Tolerances

Option	Description	Default
crash tolerance	Allow crash procedure to ignore small elements in eligible columns	0.1
feasibility tolerance	Feasibility tolerance for linear constraints	1.0e-6
linesearch tolerance	Controls accuracy of steplength selected	0.1
LU density tolerance	When to use dense factorization	0.5
LU factor tolerance	Trade-off between stability and sparsity in basis factorization	100.0
LU singularity tolerance	Protection against ill-conditioned basis matrices	1.0e-11
LU update tolerance	Trade-off between stability and sparsity in basis updating	10.0
optimality tolerance	Reduced gradient optimality check	1.0e-6
row tolerance	Accuracy requirement for nonlinear rows	1.0e-6
scale print tolerance	Scale print flag and set tolerance	0.9
scale tolerance	Scale tolerance	0.9
subspace tolerance	Determines when nonbasics becomes superbasic	0.5

5.26.6.3 Limits

Option	Description	Default
hessian dimension	Size of Hessian matrix	1
iterations limit	Minor iteration limit	GAMS iterlim
major iterations	Max number of major iterations	50
minor iterations	Max number of minor iterations between linearizations of nonlinear constraints	40
superbasics limit	Maximum number of superbasics	1
unbounded objective value	Determines when a problem is called unbounded	1.0e20
unbounded step size	Determines when a problem is called unbounded	1.0e10

5.26.6.4 Other algorithmic options

Option	Description	Default
check frequency	Controls frequency of linear constraint satisfaction test	60
completion	Completion level for subproblems (full/partial)	FULL
crash option	Controls the basis crash algorithm	3
expand frequency	Setting for anti-cycling mechanism	10000
factorization frequency	Number of iterations between basis factorizations	100
lagrangian	Determines form of objection function in the linearized subproblems	YES
LU complete pivoting	LUSOL pivoting strategy	
LU partial pivoting	LUSOL pivoting strategy	
LU rook pivoting	LUSOL pivoting strategy	
major damping parameter	Prevents large relative changes between subproblem solutions	2.0
minor damping parameter	Limit change in x during linesearch	2.0
multiple price	Multiple pricing	1
partial price	Number of segments in partial pricing strategy	10
penalty parameter	Used in modified augmented Lagrangian	automatic
radius of convergence	controls final reduction of penalty parameter	0.01
scale all variables	Synonym to scale option 2	
scale linear variables	Synonym to scale option 1	
scale no	Synonym to scale option 0	
scale nonlinear variables	Synonym to scale option 2	
scale option	Scaling	1
scale yes	Synonym to scale option 1	
start assigned nonlinears	Starting strategy when there is no basis	SUPERBASIC

Option	Description	Default
verify constraint gradients	Synonym to verify level 2	
verify gradients	Synonym to verify level 3	
verify level	Controls verification of gradients	0
verify no	Synonym to verify level 0	
verify objective gradients	Synonym to verify level 1	
verify yes	Synonym to verify level 3	
weight on linear objective	Composite objective weight	0.0

5.26.6.5 Examples of GAMS/MINOS Option File

The following example illustrates the use of certain options that might be helpful for "difficult" models involving nonlinear constraints. Experimentation may be necessary with the values specified, particularly if the sequence of major iterations does not converge using default values.

```
* These options might be relevant for very nonlinear models.
Major damping parameter  0.2   * may prevent divergence.
Minor damping parameter  0.2   * if there are singularities
                               * in the nonlinear functions.
Penalty parameter       10.0   * or 100.0 perhaps-a value
                               * higher than the default.
Scale linear variables         * (This is the default.)
```

Conversely, nonlinearly constrained models that are very nearly linear may optimize more efficiently if some of the cautious defaults are relaxed:

```
* Suggestions for models with MILDLY nonlinear constraints
Completion   Full
Penalty parameter        0.0  * or 0.1 perhaps-a value
                              * smaller than the default.
                              * Scale one of the following
Scale all variables           * if starting point is VERY GOOD.
Scale linear variables        * if they need it.
Scale No                      * otherwise.
```

Most of the options should be left at their default values for any given model. If experimentation is necessary, we recommend changing just one option at a time.

5.26.7 Special Notes

5.26.7.1 Modeling Hints

Unfortunately, there is no guarantee that the algorithm just described will converge from an arbitrary starting point. The concerned modeler can influence the likelihood of convergence as follows:

- Specify initial activity levels for the nonlinear variables as carefully as possible (using the GAMS suffix `.L`).
- Include sensible upper and lower bounds on all variables.
- Specify a *Major damping parameter* that is lower than the default value, if the problem is suspected of being highly nonlinear
- Specify a *Penalty parameter* ρ that is higher than the default value, again if the problem is highly nonlinear.

In rare cases it may be safe to request the values $\lambda_k = 0$ and $\rho = 0$ for all subproblems, by specifying *Lagrangian=No*. However, convergence is much more likely with the default setting, *Lagrangian=Yes*. The initial estimate of the Lagrange multipliers is then $\lambda_0 = 0$, but for later subproblems λ_k is taken to be the Lagrange multipliers associated with the (linearized) nonlinear constraints at the end of the previous major iteration.

For the first subproblem, the default value for the penalty parameter is $\rho = 100.0/m_1$ where m_1 is the number of nonlinear constraints. For later subproblems, ρ is reduced in stages when it appears that the sequence $\{x_k, \lambda_k\}$ is converging. In many cases it is safe to specify $\lambda = 0$, particularly if the problem is only mildly nonlinear. This may improve the overall efficiency.

5.26.7.2 Storage

GAMS/MINOS uses one large array of memory for most of its workspace. The implementation places no fixed limit on the size of a problem or on its shape (many constraints and relatively few variables, or *vice versa*). In general, the limiting factor will be the amount of physical memory available on a particular machine, and the amount of computation time one is willing to spend.

Some detailed knowledge of a particular model will usually indicate whether the solution procedure is likely to be efficient. An important quantity is m, the total number of general constraints in (2) and (3). The amount of workspace required by GAMS/MINOS is roughly $100m$ doubles, or $800m$ bytes for workspace. A further 300K bytes, approximately, are needed for the program itself, along with buffer space for several files. Very roughly, then, a model with m general constraints requires about $(m + 300)$ K bytes of memory.

Another important quantity is n, the total number of variables in x and y. The above comments assume that n is not much larger than m, the number of constraints. A typical ratio for n/m is 2 or 3.

If there are many nonlinear variables (i.e., if n_1 is large), much depends on whether the objective function or the constraints are highly nonlinear or not. The degree of nonlinearity affects s, the number of superbasic variables. Recall that s is zero for purely linear problems. We know that s need never be larger than $n_1 + 1$. In practice, s is often very much less than this upper limit.

In the quasi-Newton algorithm, the dense triangular matrix R has dimension s and requires about $s^2/2$ words of storage. If it seems likely that s will be very large, some aggregation or reformulation of the problem should be considered.

5.26.8 The GAMS/MINOS Log File

MINOS writes different logs for LPs, NLPs with linear constraints, and NLPs with non-linear constraints. In this section, a sample log file is shown for each case, and the messages that appear are explained.

5.26.8.1 Linear Programs

MINOS uses a standard two-phase simplex method for LPs. In the first phase, the sum of the infeasibilities at each iteration is minimized. Once feasibility is attained, MINOS switches to phase 2 where it minimizes (or maximizes) the original objective function. The different objective functions are called the phase 1 and phase 2 objectives. Notice that the marginals in phase 1 are with respect to the phase 1 objective. This means that if MINOS interrupts in phase 1, the marginals are "wrong" in the sense that they do not reflect the original objective.

The log for the problem **TURKPOW** is as follows:

```
GAMS Rev 235  Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- turkpow.gms(230) 3 Mb
--- Starting execution: elapsed 0:00:00.009
--- turkpow.gms(202) 4 Mb
--- Generating LP model turkey
--- turkpow.gms(205) 4 Mb
---   350 rows  949 columns  5,872 non-zeroes
--- Executing MINOS: elapsed 0:00:00.025

GAMS/MINOS       Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
M I N O S  5.51     (Jun 2004)

    GAMS/MINOS 5.51, Large Scale Nonlinear Solver
    B. A. Murtagh, University of New South Wales
    P. E. Gill, University of California at San Diego,
    W. Murray, M. A. Saunders, and M. H. Wright,
    Systems Optimization Laboratory, Stanford University

 Work space allocated           --    1.60 Mb

  Reading Rows...
  Reading Columns...

    Itn  ninf      sinf       objective
    100     3 2.283E-01 -2.51821463E+04
    200     0 0.000E+00  2.02819284E+04
    300     0 0.000E+00  1.54107277E+04
    400     0 0.000E+00  1.40211808E+04
    500     0 0.000E+00  1.33804183E+04
    600     0 0.000E+00  1.27082709E+04

 EXIT - Optimal Solution found, objective:        12657.77

--- Restarting execution
--- turkpow.gms(205) 0 Mb
--- Reading solution for model turkey
--- turkpow.gms(230) 3 Mb
*** Status: Normal completion
```

The first line that is written by MINOS is the version string: `GAMS/MINOS Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows` This line identifies which version of the MINOS libraries and links you are using, and is only to be deciphered by GAMS support personnel.

After some advertisement text we see the amount of work space (i.e. memory) that is allocated: *1.60 Mb*. When MINOS is loaded, the amount of memory needed is first estimated. This estimate is based on

statistics like the number of rows, columns and non-zeros. This amount of memory is then allocated and the problem loaded into MINOS.

The columns have the following meaning:

Itn

Iteration number.

ninf

Number of infeasibilities. If nonzero the current iterate is still infeasible.

sinf

The sum of the infeasibilities. This number is minimized during Phase I. Once the model is feasible this number is zero.

objective

The value of the objective function: $z = \sum c_i x_i$. In phase II this number is maximized or minimized. In phase I it may move in the wrong direction.

The final line indicates the exit status of MINOS.

5.26.8.2 Linearly Constrained NLP's

The log is basically the same as for linear models. The only difference is that not only matrix rows and columns need to be loaded, but also instructions for evaluating functions and gradients.

The log for the problem **WEAPONS** is as follows:

```
GAMS Rev 235  Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- weapons.gms(77) 3 Mb
--- Starting execution: elapsed 0:00:00.005
--- weapons.gms(66) 4 Mb
--- Generating NLP model war
--- weapons.gms(68) 6 Mb
---   13 rows  66 columns  156 non-zeroes
---   706 nl-code  65 nl-non-zeroes
--- weapons.gms(68) 4 Mb
--- Executing MINOS: elapsed 0:00:00.013

GAMS/MINOS       Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
M I N O S  5.51     (Jun 2004)

    GAMS/MINOS 5.51, Large Scale Nonlinear Solver
    B. A. Murtagh, University of New South Wales
    P. E. Gill, University of California at San Diego,
    W. Murray, M. A. Saunders, and M. H. Wright,
    Systems Optimization Laboratory, Stanford University

 Work space allocated           --    0.82 Mb

  Reading Rows...
```

```
  Reading Columns...
  Reading Instructions...

    Itn  ninf      sinf       objective
    100     0 0.000E+00  1.71416714E+03
    200     0 0.000E+00  1.73483184E+03

 EXIT - Optimal Solution found, objective:        1735.570

--- Restarting execution
--- weapons.gms(68) 0 Mb
--- Reading solution for model war
--- weapons.gms(77) 3 Mb
*** Status: Normal completion
```

5.26.8.3 NLP's with Nonlinear Constraints

For models with nonlinear constraints the log is more complicated. The library model **[CAMCGE]** from the model library is such an example: the log output resulting from running it is shown below.

```
GAMS Rev 235  Copyright (C) 1987-2010 GAMS Development. All rights reserved
--- Starting compilation
--- camcge.gms(450) 3 Mb
--- Starting execution: elapsed 0:00:00.010
--- camcge.gms(441) 4 Mb
--- Generating NLP model camcge
--- camcge.gms(450) 6 Mb
---   243 rows  280 columns  1,356 non-zeroes
---   5,524 nl-code  850 nl-non-zeroes
--- camcge.gms(450) 4 Mb
--- Executing MINOS: elapsed 0:00:00.023

GAMS/MINOS       Aug 18, 2010 23.5.2 WIN 19143.19383 VS8 x86/MS Windows
M I N O S  5.51     (Jun 2004)

    GAMS/MINOS 5.51, Large Scale Nonlinear Solver
    B. A. Murtagh, University of New South Wales
    P. E. Gill, University of California at San Diego,
    W. Murray, M. A. Saunders, and M. H. Wright,
    Systems Optimization Laboratory, Stanford University

 Work space allocated           --    1.48 Mb

  Reading Rows...
  Reading Columns...
  Reading Instructions...

 Major minor  step     objective  Feasible Optimal  nsb   ncon penalty BSswp
     1     2T 0.0E+00  1.91724E+02 1.8E+02 2.0E-01    0      1 1.0E+00     0
     2    90  1.0E+00  1.91735E+02 1.5E-03 7.6E+00    0      3 1.0E+00     0
     3     0  1.0E+00  1.91735E+02 1.3E-09 5.5E-06    0      4 1.0E+00     0
     4     0  1.0E+00  1.91735E+02 1.1E-12 2.8E-13    0      5 1.0E-01     0

 EXIT - Optimal Solution found, objective:        191.7346

--- Restarting execution
```

```
--- camcge.gms(450) 0 Mb
--- Reading solution for model camcge
*** Status: Normal completion
```

Two sets of iterations, *major* and *minor*, are now reported. A description of the various columns present in this log file follows:

Major

A major iteration involves linearizing the nonlinear constraints and performing a number of minor iterations on the resulting subproblem. The objective for the subproblem is an augmented Lagrangian, not the true objective function.

minor

The number of minor iterations performed on the linearized subproblem. If it is a simple number like 90, then the subproblem was solved to optimality. Here, $2T$ means that the subproblem was terminated. In general the T is not something to worry about. Other possible flags are I and U, which mean that the subproblem was infeasible or unbounded. MINOS may have difficulty if these keep occurring.

step

The step size taken towards the solution suggested by the last major iteration. Ideally this should be 1.0, especially near an optimum. If the subproblem solutions are widely different, MINOS may reduce the step size under control of the *Major Damping parameter*.

objective

The objective function for the original nonlinear program.

Feasible

Primal infeasibility, indicating the maximum non-linear constraint violation.

Optimal

The maximum dual infeasibility, measured as the maximum departure from complementarity. If we call d_j the reduced cost of variable x_j, then the dual infeasibility of x_j is $d_j \times \min\{x_j - \ell_j, 1\}$ or $-d_j \times \min\{u_j - x_j, 1\}$ depending on the sign of d_j.

nsb

Number of superbasics. If the model is feasible this number cannot exceed the superbasic limit, which may need to be reset to a larger number if the numbers in this column become larger.

ncon

The number of times MINOS has evaluated the nonlinear constraints and their derivatives.

penalty

The current value of the penalty parameter in the augmented Lagrangian (the objective for the subproblems). If the major iterations appear to be converging, MINOS will decrease the penalty parameter. If there appears to be difficulty, such as unbounded subproblems, the penalty parameter will be increased.

BSswp

Number of basis swaps: the number of $\begin{pmatrix} B & S \end{pmatrix}$ (i.e. basic vs. superbasic) changes.

Note: The **CAMCGE** model (like many CGE models or other almost square systems) can better be solved with the MINOS option *Start Assigned Nonlinears Basic*.

5.26.9 Detailed Description of MINOS Options

The following is an alphabetical list of the keywords that may appear in the GAMS/MINOS options file, and a description of their effect. Options not specified will take the default values shown.

check frequency *(integer)*: Controls frequency of linear constraint satisfaction test ↩

Every i^{th} iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax+s = 0$ where s is the set of slack variables. To perform the numerical test, the residual vector $r = Ax + s$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

Range: {1, ..., ∞}

Default: 60

completion *(string)*: Completion level for subproblems (full/partial) ↩

When there are nonlinear constraints, this determines whether subproblems should be solved to moderate accuracy (partial completion) or to full accuracy (full completion). GAMS/MINOS implements the option by using two sets of convergence tolerances for the subproblems.
Use of partial completion may reduce the work during early major iterations, unless the Minor iterations limit is active. The optimal set of basic and superbasic variables will probably be determined for any given subproblem, but the reduced gradient may be larger than it would have been with full completion. An automatic switch to full completion occurs when it appears that the sequence of major iterations is converging. The switch is made when the nonlinear constraint error is reduced below 100 $*$ (Row tolerance), the relative change in $Lambda_k$ is 0.1 or less, and the previous subproblem was solved to optimality. Full completion tends to give better Langrange-multiplier estimates. It may lead to fewer major iterations, but may result in more minor iterations.

Default: FULL

value	meaning
FULL	Solve subproblems to full accuracy
PARTIAL	Solve subproblems to moderate accuracy

crash option *(integer)*: Controls the basis crash algorithm ↩

If a restart is not being performed, an initial basis will be selected from certain columns of the constraint matrix *(A I)*. The value of the parameter i determines which columns of A are eligible. Columns of I are used to fill gaps where necessary. If $i > 0$, three passes are made through the relevant columns of A, searching for a basis matrix that is essentially triangular. A column is assigned to pivot on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis). Pass 1 selects pivots from free columns (corresponding to variables with no upper and lower bounds). Pass 2 requires pivots to be in rows associated with equality (=E=) constraints. Pass 3 allows the pivots to be in inequality rows. For remaining (unassigned) rows, the associated slack variables are inserted to complete the basis.

Default: 3

value	meaning
0	Initial basis will be a slack basis

value	meaning
1	All columns are eligible
2	Only linear columns are eligible
3	Columns appearing nonlinearly in the objective are not eligible
4	Columns appearing nonlinearly in the constraints are not eligible

crash tolerance *(real)*: Allow crash procedure to ignore small elements in eligible columns ↩

The *Crash tolerance r* allows the starting procedure *CRASH* to ignore certain small nonzeros in each column of A. If a_{max} is the largest element in column j, other nonzeros a_{ij} in the column are ignored if $|a_{ij}| < a_{max} * r$. To be meaningful, the parameter r should be in the range $0 <= r < 1$. When $r > 0.0$ the basis obtained by *CRASH* may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems. For example, suppose the first m columns of A are the matrix shown under LU factor tolerance; i.e., a tridiagonal matrix with entries -1, 4, -1. To help *CRASH* choose all m columns for the initial basis, we could specify *Crash tolerance r* for some value of $r > 0.25$.

Range: [0, 1.0]

Default: 0.1

debug level *(integer)*: Controls amount of debug information written ↩

This causes various amounts of information to be output. Most debug levels will not be helpful to GAMS users, but they are listed here for completeness. Note that you will need to use the GAMS statement `OPTION SYSOUT=on;` to echo the MINOS listing to the GAMS listing file.

- **debug level 0**
  No debug output.
- **debug level 2**(or more)
  Output from *M5SETX* showing the maximum residual after a row check.
- **debug level 40**
  Output from *LU8RPC* (which updates the LU factors of the basis matrix), showing the position of the last nonzero in the transformed incoming column.
- **debug level 50**
  Output from *LU1MAR* (which updates the LU factors each refactorization), showing each pivot row and column and the dimensions of the dense matrix involved in the associated elimination.
- **debug level 100**
  Output from *M2BFAC* and *M5LOG* listing the basic and superbasic variables and their values at every iteration.

Default: 0

expand frequency *(integer)*: Setting for anti-cycling mechanism ↩

This option is part of an anti-cycling procedure designed to guarantee progress even on highly degenerate problems. For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose the specified feasibility tolerance is *delta* and the expand frequency is k. Over a period of k iterations, the tolerance actually used by GAMS/MINOS increases from *0.5*delta* to *delta* (in steps *0.5*delta/k*). For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is

at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced. At least every k iterations, a resetting procedure eliminates any infeasible nonbasic variables. Increasing k helps to reduce the number of these slightly infeasible nonbasic variables. However, it also diminishes the freedom to choose a large pivot element (see Pivot tolerance).

Range: {1, ..., ∞}

Default: `10000`

factorization frequency *(integer)*: Number of iterations between basis factorizations ↩

At most i basis updates will occur between factorizations of the basis matrix. With linear programs, basis updates usually occur at every iteration. The default i is reasonable for typical problems. Higher values up to $i = 200$ (say) may be more efficient on problems that are extremely sparse and well scaled. When the objective function is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the Check frequency) to ensure that the general constraints are satisfied. If necessary the basis will be re-factorized before the limit of i updates is reached. When the constraints are nonlinear, the Minor iterations limit will probably preempt i.

Range: {1, ..., ∞}

Default: `100`

feasibility tolerance *(real)*: Feasibility tolerance for linear constraints ↩

When the constraints are linear, a feasible solution is one in which all variables, including slacks, satisfy their upper and lower bounds to within the absolute tolerance r. (Since slacks are included, this means that the general linear constraints are also satisfied within r.) GAMS/MINOS attempts to find a feasible solution before optimizing the objective function. If the sum of infeasibilities cannot be reduced to zero, the problem is declared infeasible. Let *SINF* be the corresponding sum of infeasibilities. If *SINF* is quite small, it may be appropriate to raise r by a factor of 10 or 100. Otherwise, some error in the data should be suspected. If *SINF* is not small, there may be other points that have a significantly smaller sum of infeasibilities. GAMS/MINOS does not attempt to find a solution that minimizes the sum. If *Scale option* = 1 or 2, feasibility is defined in terms of the scaled problem (since it is then more likely to be meaningful). A nonlinear objective function *F(x)* will be evaluated only at feasible points. If there are regions where *F(x)* is undefined, every attempt should be made to eliminate these regions from the problem. For example, for a function *F(x) = sqrt(x1)+log(x2)*, it should be essential to place lower bounds on both variables. If *Feasibility tolerance* = 10^{-6}, the bounds $x_1 > 10^{-5}$ and $x_2 > 10^{-4}$ might be appropriate. (The log singularity is more serious; in general, keep variables as far away from singularities as possible.) If the constraints are nonlinear, the above comments apply to each major iteration. A feasible solution satisfies the current linearization of the constraints to within the tolerance r. The associated subproblem is said to be feasible. As for the objective function, bounds should be used to keep x more than r away from singularities in the constraint functions *f(x)*. At the start of major iteration k, the constraint functions $f(x_k)$ are evaluated at a certain point x_k. This point always satisfies the relevant bounds ($l < x_k < u$), but may not satisfy the general linear constraints. During the associated minor iterations, *F(x)* and *f(x)* will be evaluated only at points x that satisfy the bound and the general linear constraints (as well as the linearized nonlinear constraints). If a subproblem is infeasible, the bounds on the linearized constraints are relaxed temporarily, in several stages. Feasibility with respect to the nonlinear constraints themselves is measured against the Row tolerance (not against r). The relevant test is made at the start of a major iteration.

Default: `1.0e-6`

hessian dimension *(integer)*: Size of Hessian matrix ↩

This specifies that an $r*r$ triangular matrix R is to be available for use by the quasi-Newton algorithm. The matrix R approximates the reduced Hessian in that R^TR approximates Z^THZ. Suppose there are s superbasic variables at a particular iteration. Whenever possible, r should be greater than s. If $r > s$, the first s columns of R will be used to approximate the reduced Hessian in the normal manner. If there are no further changes to the set of superbasic variables, the rate of convergence will ultimately be superlinear. If $r < s$, a matrix of the form

$R = diag(R_r, D)$

will be used to approximate the reduced Hessian, where R_r is an $r * r$ upper triangular matrix and D is a diagonal matrix of order s - r. The rate of convergence will no longer be superlinear (and may be arbitrarily slow). The storage required is of the order *sqr(r)/2*, i.e. quadratic in r. In general, r should be a slight over-estimate of the final number of superbasic variables, whenever storage permits. It need never be larger than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems it can be much smaller than n_1. If *Superbasics limit s* is specified, the default value of r is the same number, s (and conversely). This is a safeguard to ensure super-linear convergence wherever possible. If neither r nor s is specified, GAMS chooses values for both, using certain characteristics of the problem.

Range: {1, ..., ∞}

Default: `1`

iterations limit *(integer)*: Minor iteration limit ↩

The maximum number of minor iterations allowed (i.e., iterations of the simplex method or the reduced-gradient method). This option, if set, overrides the GAMS *ITERLIM* specification. If $i = 0$, no minor iterations are performed, but the starting point is tested for both feasibility and optimality.

Default: `GAMS iterlim`

lagrangian *(string)*: Determines form of objection function in the linearized subproblems ↩

This determines the form of the objective function used for the linearized subproblems. The default value *yes* is highly recommended. The *Penalty parameter* value is then also relevant. If *No* is specified, the nonlinear constraint functions will be evaluated only twice per major iteration. Hence this option may be useful if the nonlinear constraints are very expensive to evaluate. However, in general there is a great risk that convergence may not occur.

Default: `YES`

value	meaning
`NO`	Nondefault value (not recommended)
`YES`	Default value (recommended)

linesearch tolerance *(real)*: Controls accuracy of steplength selected ↩

For nonlinear problems, this controls the accuracy with which a steplength *alpha* is located in the one-dimensional problem

minimize *F(x+alpha∗p)*
subject to *0 < alpha <= beta*

A linesearch occurs on most minor iterations for which x is feasible. (If the constraints are nonlinear, the function being minimized is the augmented Lagrangian.) r must be a real value in the range $0.0 < r < 1.0$. The default value $r = 0.1$ requests a moderately accurate search. It should be satisfactory in most cases. If the nonlinear functions are cheap to evaluate, a more accurate search may be appropriate: try $r = 0.01$ or $r = 0.001$. The number of iterations should decrease, and this will reduce total run time if there are many linear or nonlinear constraints. If the nonlinear function are expensive to evaluate, a less accurate search may be appropriate; try $r = 0.5$ or perhaps $r = 0.9$. (The number of iterations will probably increase but the total number of function evaluations may decrease enough to compensate.)

Range: [`0`, `1.0`]

Default: `0.1`

log frequency *(integer)*: Controls iteration logging to listing file ↩

In general, one line of the iteration log is printed every i^{th} minor iteration. A heading labels the printed items, which include the current iteration number, the number and sum of feasibilities (if any), the subproblem objective value (if feasible), and the number of evaluations of the nonlinear functions. A value such as $i = 10, 100$ or larger is suggested for those interested only in the final solution. *Log frequency 0* may be used as shorthand for *Log frequency 99999*. If *Print level* > 0, the default value of i is 1. If *Print level* $= 0$, the default value of i is 100. If *Print level* $= 0$ and the constraints are nonlinear, the minor iteration log is not printed (and the *Log frequency* is ignored). Instead, one line is printed at the beginning of each major iteration.

Range: {`1`, ..., ∞}

Default: `100`

LU complete pivoting *(no value)*: LUSOL pivoting strategy ↩

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say < 2.0).

LU density tolerance *(real)*: When to use dense factorization ↩

The density tolerance is used during LUSOL's basis factorization B=LU. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds this tolerance, the Markowitz strategy for choosing pivots is terminated and the remaining matrix is factored by a dense LU procedure. Raising the tolerance towards 1.0 may give slightly sparser factors, with a slight increase in factorization time.

Range: [`0`, `1.0`]

Default: `0.5`

LU factor tolerance *(real)*: Trade-off between stability and sparsity in basis factorization ↩

This tolerance affects the stability and sparsity of the basis factorization $B = LU$ during factorization. The value r specified must satisfy $r >= 1.0$.

- The default value $r = 100.0$ usually strikes a good compromise between stability and sparsity.

- For large and relatively dense problems, a larger value may give a useful improvement in sparsity without impairing stability to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to set r to a value smaller than the default in order to achieve stability.

Range: [`1.0`, ∞]

Default: `100.0`

LU partial pivoting *(no value)*: LUSOL pivoting strategy ↩

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say $<$ *2.0*).

LU rook pivoting *(no value)*: LUSOL pivoting strategy ↩

The LUSOL factorization implements a Markowitz-style search for pivots that locally minimize fill-in subject to a threshold pivoting stability criterion. The *rook* and *complete pivoting* options are more expensive than *partial pivoting* but are more stable and better at revealing rank, as long as the *LU factor tolerance* is not too large (say $<$ *2.0*).

LU singularity tolerance *(real)*: Protection against ill-conditioned basis matrices ↩

When the basis is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| <= r$ or $|U_{jj}| < r * max_i |U_{ii}|$, the j^{th} column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart, or at the start of a major iteration.) In some cases, the Jacobian matrix may converge to values that make the basis very ill-conditioned, causing the optimization to progress very slowly (if at all). Setting r = *1.0*$^{-5}$, say, may help cause a judicious change of basis.

Default: `1.0e-11`

LU update tolerance *(real)*: Trade-off between stability and sparsity in basis updating ↩

This tolerance affects the stability and sparsity of the basis factorization $B = LU$ during updates. The value r specified must satisfy $r >= 1.0$.

- The default value r = *10.0* usually strikes a good compromise between stability and sparsity.
- For large and relatively dense problems, r = *25.0* (say) may give a useful improvement in sparsity without impairing stability to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to set r to a value smaller than the default in order to achieve stability.

Range: [`1.0`, ∞]

Default: `10.0`

major damping parameter *(real)*: Prevents large relative changes between subproblem solutions ↩

The parameter may assist convergence on problems that have highly nonlinear constraints. It is intended to prevent large relative changes between subproblem solutions *(x_k, $lambda_k$)* and *(x_{k+1}, $lambda_{k+1}$)*. For example, the default value 2.0 prevents the relative change in either x_k or $lambda_k$ from exceeding 200 percent. It will not be active on well behaved problems. The parameter is used to interpolate between the solutions at the beginning and end of each major iteration. Thus x_{k+1} and $lambda_{k+1}$ are changed to x_k + *sigma*(x_{k+1} - x_k)* and $lambda_k$ + *sigma*($lambda_{k+1}$ - $lambda_k$)* for some step-length *sigma* < *1*. In the case of nonlinear equations (where the number of constraints is the same as the number of variables) this gives a damped Newton method. This is a very crude control. If the sequence of major iterations does not appear to be converging, one should first re-run the problem with a higher Penalty parameter (say 10 or 100 times the default *rho*). (Skip this re-run in the case of nonlinear equations: there are no degrees of freedom and the value of *rho* is irrelevant.) If the subproblem solutions continue to change violently, try reducing *r* to 0.2 or 0.1 (say). For implementation reasons, the shortened step to *sigma* applies to the nonlinear variables *x*, but not to the linear variables *y* or the slack variables *s*. This may reduce the efficiency of the control.

Default: `2.0`

major iterations *(integer)*: Max number of major iterations ↩

The maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the nonlinear constraints, since in some cases the sequence of major iterations may not converge. The progress of the major iterations can be best monitored using *Print level 0* (the default).

Default: `50`

minor damping parameter *(real)*: Limit change in x during linesearch ↩

This parameter limits the change in *x* during a linesearch. It applies to all nonlinear problems, once a feasible solution or feasible subproblem has been found. A linesearch of the form

$$minimize_{alpha}\ F(x + alpha*p)$$

is performed over the range $0 < alpha <= beta$, where *beta* is the step to the nearest upper or lower bound on *x*. Normally, the first step length tried is $a_1 = min(1, beta)$. In some cases, such as $F(x) = ae^{bx}$ or $F(x) = ax^b$, even a moderate change in the components of *x* can lead to floating-point overflow. The parameter *r* is therefore used to define a limit

$$beta2 = r(1 + ||x||)/||p||$$

and the first evaluation of *F(x)* is at the potentially smaller steplength $alpha_1 = min(1, beta, beta2)$. Wherever possible, upper and lower bounds on *x* should be used to prevent evaluation of nonlinear functions at meaningless points. The *Minor damping parameter* provides an additional safeguard. The default value *r = 2.0* should not affect progress on well behaved problems, but setting *r = 0.1* or *0.01* may be helpful when rapidly varying functions are present. A good starting point may be required. An important application is to the class of nonlinear least squares problems. In cases where several local optima exist, specifying a small value for *r* may help locate an optimum near the starting point.

Default: `2.0`

minor iterations *(integer)*: Max number of minor iterations between linearizations of nonlinear constraints ↩

The the maximum number of feasible minor iterations allowed between successive linearizations of the nonlinear constraints. A moderate value (e.g., *20 <= i <= 50*) prevents excessive efforts being expended on early major iterations, but allows later subproblems to be solved to completion. The limit applies to both infeasible and feasible iterations. In some cases, a large number of iterations (say *K*) might be required to obtain a feasible subproblem. If good starting values are supplied for variables appearing nonlinearly in the constraints, it may be sensible to specify a limit $> K$, to allow the first major iteration to terminate at a feasible (and perhaps optimal) subproblem solution. If a good initial subproblem is arbitrarily interrupted by a small limit, the subsequent linearization may be less favorable than the first. In general it is unsafe to specify a value as small as *i = 1* or *2*. Even when an optimal solution has been reached, a few minor iterations may be needed for the corresponding subproblem to be recognized as optimal. The *Iteration limit* provides an independent limit on the total minor iterations (across all subproblems). If the constraints are linear, only the *Iteration limit* applies: the minor iterations value is ignored.

Default: `40`

multiple price *(integer)*: Multiple pricing ↩

Pricing refers to a scan of the current non-basic variables to determine which, if any, are eligible to become (super)basic. The multiple pricing parameter *k* controls the number of entering variables to choose: the *k* best non-basic variables are selected for admission to the set of (super)basic variables. The default *k = 1* is best for linear programs, since an optimal solution will have zero superbasic variables. *Warning* : If *k > 1*, GAMS/MINOS will use the reduced-gradient method rather than the simplex method, even on purely linear problems. The subsequent iterations do not correspond to the efficient minor iterations carried out by commercial linear programming systems using multiple pricing. (In the latter systems, the classical simplex method is applied to a tableau involving *k* dense columns of dimension *m*, and *k* is therefore limited for storage reasons typically to the range *2 <= k <= 7*.) GAMS/MINOS varies all superbasic variables simultaneously. For linear problems its storage requirements are essentially independent of *k*. Larger values of *k* are therefore practical, but in general the iterations and time required when *k > 1* are greater than when the simplex method is used (*k = 1*). On large nonlinear problems it may be important to set *k > 1* if the starting point does not contain many superbasic variables. For example, if a problem has 3000 variables and 500 of them are nonlinear, the optimal solution may well have 200 variables superbasic. If the problem is solved in several runs, it may be beneficial to use *k = 10* (say) for early runs, until it seems that the number of superbasics has leveled off. If *Multiple price k* is specified, it is also necessary to specify *Superbasic limit s* for some *s > k*.

Range: {`1`, ..., ∞}

Default: `1`

optimality tolerance *(real)*: Reduced gradient optimality check ↩

This is used to judge the size of the reduced gradients $d_j = g_j$ - pi^T a_j, where g_j is the gradient of the objective function corresponding to the j^{th} variable, a_j is the associated column of the constraint matrix (or Jacobian), and *pi* is the set of dual variables. By construction, the reduced gradients for basic variables are always zero. Optimality will be declared if the reduced gradients for nonbasic variables at their lower or upper bounds satisfy $d_j/||pi|| >=$ *-r* or $d_j/||pi|| <= r$ respectively, and if $d_j/||pi|| <= r$ for superbasic variables. The $||pi||$ is a measure of the size of the dual variables. It is included to make the tests independent of a scale factor on the objective function. The quantity actually used is defined by

sigma = sum(i, abs(pi(i))), ||pi|| = max{sigma/sqrt(m),1}

so that only large scale factors are compensated for. As the objective scale decreases, the optimality test tends to become an absolute (instead of a relative) test.

Default: `1.0e-6`

partial price *(integer)*: Number of segments in partial pricing strategy ↩

This parameter is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each pricing operation (when a nonbasic variable is selected to become basic or superbasic). When $i = 1$, all columns of the constraints matrix *(A I)* are searched. Otherwise, A_j and I are partitioned to give i roughly equal segments A_j, I_j ($j = 1$ to i). If the previous search was successful on A_{j-1}, I_{j-1}, the next search begins on the segments A_j, I_j. (All subscripts here are modulo i.) If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. (Several may be selected if multiple pricing has been specified.) If nothing is found, the search continues on the next segments A_{j+1}, I_{j+1} and so on. *Partial price t* (or *t/2* or *t/3*) may be appropriate for time-stage models having t time periods

Range: {1, ..., ∞}

Default: `10`

penalty parameter *(real)*: Used in modified augmented Lagrangian ↩

This specifies the value of *rho* in the modified augmented Lagrangian. It is used only when *Lagrangian = yes* (the default setting). For early runs on a problem with unknown characteristics, the default value should be acceptable. If the problem is known to be highly nonlinear, specify a large value, such as 10 times the default. In general, a positive value of *rho* may be necessary to ensure convergence, even for convex programs. On the other hand, if *rho* is too large, the rate of convergence may be unnecessarily slow. If the functions are not highly nonlinear or a good starting point is known, it will often be safe to specify *penalty parameter 0.0*. When solving a sequence of related problems, initially, use a moderate value for *rho* (such as the default) and a reasonably low *Iterations* and/or *major iterations limit*. If successive major iterations appear to be terminating with radically different solutions, the *penalty parameter* should be increased. (See also the *Major damping parameter*.) If there appears to be little progress between major iterations, it may help to reduce the *penalty parameter*.

Default: `automatic`

print level *(integer)*: Controls amount of information printed during optimization ↩

This varies the amount of information that will be output during optimization. *Print level 0* sets the default log and summary frequencies to 100. It is then easy to monitor the progress of the run. *Print level 1* (or more) sets the default log and summary frequencies to 1, giving a line of output for every minor iteration. *Print level 1* also produces basis statistics, i.e., information relating to LU factors of the basis matrix whenever the basis is re-factorized. For problems with nonlinear constraints, certain quantities are printed at the start of each major iteration. The option value is best thought of as a binary number of the form

Print level `JFLXB`

where each letter stands for a digit that is either 0 or 1. The quantities referred to are:

- **B** Basis statistics, as mentioned above
- **X** x_k, the nonlinear variables involved in the objective function or the constraints.
- **L** $lambda_k$, the Lagrange-multiplier estimates for the nonlinear constraints. (Suppressed if *Lagrangian=No*, since then $lambda_k = 0$.)
- **F** $f(x_k)$, the values of the nonlinear constraint functions.
- **J** $J(x_k)$, the Jacobian matrix.

To obtain output of any item, set the corresponding digit to 1, otherwise to 0. For example, *Print level 10* sets `X = 1` and the other digits equal to zero; the nonlinear variables will be printed each major iteration. If `J = 1`, the Jacobian matrix will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if `J = 1`, there is no reason to specify `X = 1` unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3 1.250000D+01 BS 1 1.00000D+00 4 2.00000D+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4. (Note: the GAMS/MINOS row numbers are usually different from the GAMS row numbers; see the Solution option.)

Default: `0`

radius of convergence *(real)*: controls final reduction of penalty parameter ↩

This determines when the penalty parameter *rho* will be reduced, assuming *rho* was initially positive. Both the nonlinear constraint violation (see *ROWERR* below) and the relative change in consecutive Lagrange multiplier estimates must be less than r at the start of a major iteration before *rho* is reduced or set to zero. A few major iterations later, full completion will be requested if not already set, and the remaining sequence of major iterations should converge quadratically to an optimum.

Default: `0.01`

row tolerance *(real)*: Accuracy requirement for nonlinear rows ↩

This specifies how accurately the nonlinear constraints should be satisfied at a solution. The default value is usually small enough, since model data is often specified to about this accuracy. Let *ROWERR* be the maximum component of the residual vector $f(x) + A_1 y - b_1$, normalized by the size of the solution. Thus

$$ROWERR = ||f(x) + A_1 y - b_1||_{inf}/(1 + XNORM)$$

where *XNORM* is a measure of the size of the current solution (x, y). The solution is considered to be feasible if $ROWERR <= r$. If the problem functions involve data that is known to be of low accuracy, a larger *Row tolerance* may be appropriate.

Default: `1.0e-6`

scale all variables *(no value)*: Synonym to scale option 2 ↩

scale linear variables *(no value)*: Synonym to scale option 1 ↩

scale no *(no value)*: Synonym to scale option 0 ↩

scale nonlinear variables *(no value)*: Synonym to scale option 2 ↩

scale option *(integer)*: Scaling ↩

Scale Yes sets the default. (*Caution*: If all variables are nonlinear, *Scale Yes* unexpectedly does nothing, because there are no linear variables to scale). *Scale No* suppresses scaling (equivalent to *Scale Option 0*). If nonlinear constraints are present, *Scale option 1* or *0* should generally be tried at first. *Scale option 2* gives scales that depend on the initial Jacobian, and should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

Default: `1`

value	meaning
0	No scaling If storage is at a premium, this option should be used.
1	Scale linear variables Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see [5]). This will sometimes improve the performance of the solution procedures. *Scale linear variables* is an equivalent option.
2	Scale linear + nonlinear variables All constraints and variables are scaled by the iterative procedure. Also, a certain additional scaling is performed that may be helpful if the right-hand side b or the solution x is large. This takes into account columns of *(A I)* that are fixed or have positive lower bounds or negative upper bounds. *Scale nonlinear variables* or *Scale all variables* are equivalent options.

scale print *(no value)*: Print scaling factors ↩

This causes the row-scales *r(i)* and column-scales *c(j)* to be printed. The scaled matrix coefficients are $\hat{a}_{ij} = a_{ij}c(j)/r(i)$. The scaled bounds on the variables and slacks are $\hat{l}_j = l_j/c(j)$ and $\hat{u}_j = u_j/c(j)$, where $c(j) = r(j - n)$ if $j > n$. If a Scale option has not already been specified, *Scale print* sets the default scaling.

scale print tolerance *(real)*: Scale print flag and set tolerance ↩

See *Scale Tolerance*. This option also turns on printing of the scale factors.

Range: [0, 1.0]

Default: 0.9

scale tolerance *(real)*: Scale tolerance ↩

All forms except *Scale option* may specify a tolerance r where $0 < r < 1$ (for example: *Scale Print Tolerance = 0.99*). This affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$rho_j = max_i\ |a_{ij}|/min_i\ |a_{ij}|\ \ (a_{ij} \neq 0)$$

If $max_j\ rho_j$ is less than r times its previous value, another scaling pass is performed to adjust the row and column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A. At most 10 passes are made. If a *Scale option* has not already been specified, *Scale tolerance* sets the default scaling.

Range: [0, 1.0]

Default: 0.9

scale yes *(no value)*: Synonym to scale option 1 ↩

solution *(string)*: Prints MINOS solution ↩

This controls whether or not GAMS/MINOS prints the final solution obtained. There is one line of output for each constraint and variable. The lines are in the same order as in the GAMS solution, but the constraints and variables labeled with internal GAMS/MINOS numbers rather than GAMS names. (The numbers at the left of each line are GAMS/MINOS column numbers, and those at the right of each line in the rows section are GAMS/MINOS slacks.) The GAMS/MINOS solution may be useful occasionally to interpret certain messages that occur during the optimization, and to determine the final status of certain variables (basic, superbasic or nonbasic).

Default: NO

value	meaning
NO	Turn off printing of solution
YES	Turn on printing of solution

start assigned nonlinears *(string)*: Starting strategy when there is no basis ↩

This option affects the starting strategy when there is no basis (i.e., for the first solve or when the GAMS statement *option bratio = 1* is used to reject an existing basis.) This option applies to all nonlinear variables that have been assigned nondefault initial values and are strictly between their bounds. Free variables at their default value of zero are excluded. Let K denote the number of such assigned nonlinear variables.

Default: SUPERBASIC

value	meaning
SUPERBASIC	Default Specify *superbasic* for highly nonlinear models, as long as K is not too large (say $K < 100$) and the initial values are good.
BASIC	Good for square systems Specify *basic* for models that are essentially square (i.e., if there are about as many general constraints as variables).
NONBASIC	Specify *nonbasic* if K is large.
ELIGIBLE FOR CRASH	Specify *Eligible for Crash* for linear or nearly linear models. The nonlinear variables will be treated in the manner described under *Crash* option.

subspace tolerance *(real)*: Determines when nonbasics becomes superbasic ↩

This controls the extent to which optimization is confined to the current set of basic and superbasic variables (Phase 4 iterations), before one or more nonbasic variables are added to the superbasic set (Phase 3). The parameter r must be a real number in the range $0 < r <= 1$. When a nonbasic variable x_j is made superbasic, the resulting norm of the reduced-gradient vector (for all superbasics) is recorded. Let this be $||Z^T\ g_0||$. (In fact, the norm will be $|d_j|$, the size of the reduced gradient for the new superbasic variable x_j. Subsequent Phase 4 iterations will continue at least until the norm of the reduced-gradient vector satisfies $||Z^T\ g_0|| <= r||Z^T\ g_0||$ is the size of the largest reduced-gradient component among the superbasic variables.) A smaller value of r is likely to increase the total number of iterations, but may reduce the number of basic changes. A larger value such as $r = 0.9$ may sometimes lead to improved overall efficiency, if the number of superbasic variables has to increase substantially between the starting point and an optimal solution. Other convergence tests on the change in the function being minimized and the change in the variables may prolong Phase 4 iterations. This helps to make the overall performance insensitive to larger values of r.

Range: [0, 1.0]

Default: 0.5

summary frequency *(integer)*: Controls iteration logging to summary (log file) ↩

A brief form of the iteration log is output to the MINOS summary file (i.e. the GAMS log file). In general, one line is output every i^{th} minor iteration. In an interactive environment, the output normally appears at the terminal and allows a run to be monitored. If something looks wrong, the run can be manually terminated. The summary frequency controls summary output in the same way as the log frequency controls output to the print file. A value such as

Summary Frequency = 10 or *100* is often adequate to determine if the solve is making progress. If *Print level = 0*, the default value of *Summary Frequency* is 100. If *Print level > 0*, the default value of *Summary Frequency* is 1. If *Print level = 0* and the constraints are nonlinear, the *Summary Frequency* is ignored. Instead, one line is printed at the beginning of each major iteration.

Range: {1, ..., ∞}

Default: 100

superbasics limit *(integer)*: Maximum number of superbasics ↩

This places a limit on the storage allocated for superbasic variables. Ideally, the parameter i should be set slightly larger than the number of degrees of freedom expected at an optimal solution. For linear problems, an optimum is normally a basic solution with no degrees of freedom. (The number of variables lying strictly between their bounds is not more than m, the number of general constraints.) The default value of i is therefore 1. For nonlinear problems, the number of degrees of freedom is often called the number of independent variables. Normally, i need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems, i may be considerably smaller than n_1. This will save storage if n_1 is very large. This parameter also sets the Hessian dimension, unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, using certain characteristics of the problem.

Range: {1, ..., ∞}

Default: 1

unbounded objective value *(real)*: Determines when a problem is called unbounded ↩

This parameter is intended to detect unboundedness in nonlinear problems. During a line search of the form

$$minimize_{alpha}\ F(x + alpha*p)$$

If $|F|$ exceeds the parameter r or if *alpha* exceeds the *unbounded stepsize*, iterations are terminated with the exit message PROBLEM IS UNBOUNDED (OR BADLY SCALED). If singularities are present, unboundedness in *F(x)* may be manifested by a floating-point overflow (during the evaluation of *F(x + alpha∗p)*, before the test against r can be made. Unboundedness is best avoided by placing finite upper and lower bounds on the variables. See also the *Minor damping parameter*.

Default: 1.0e20

unbounded step size *(real)*: Determines when a problem is called unbounded ↩

This parameter is intended to detect unboundedness in nonlinear problems. During a line search of the form

$$minimize_{alpha}\ F(x + alpha*p)$$

If *alpha* exceeds the parameter r or if $|F|$ exceeds the *unbounded objective value*, iterations are terminated with the exit message PROBLEM IS UNBOUNDED (OR BADLY SCALED). If singularities are present, unboundedness in *F(x)* may be manifested by a floating-point overflow (during the evaluation of *F(x + alpha∗p)*, before the test against r can be made. Unboundedness is best avoided by placing finite upper and lower bounds on the variables. See also the *Minor damping parameter*.

Default: 1.0e10

verify constraint gradients *(no value)*: Synonym to verify level 2 ↩

verify gradients *(no value)*: Synonym to verify level 3 ↩

verify level *(integer)*: Controls verification of gradients ↩

This option controls the finite-difference check performed by MINOS on the gradients (first derivatives) computed by GAMS for each nonlinear function. GAMS computes gradients analytically, and the values obtained should normally be taken as correct.

Default: `0`

value	meaning
`0`	Cheap test Only a cheap test is performed, requiring three evaluations of the nonlinear objective and two evaluations of the nonlinear constraints. *Verify No* is an equivalent option.
`1`	Check objective A more reliable check is made on each component of the objective gradient. *Verify objective gradients* is an equivalent option.
`2`	Check Jacobian A check is made on each column of the Jacobian matrix associated with the nonlinear constraints. *Verify constraint gradients* is an equivalent option.
`3`	Check objective and Jacobian A detailed check is made on both the objective and the Jacobian. *Verify*, *Verify gradients*, and *Verify Yes* are equivalent options.
`-1`	No check

verify no *(no value)*: Synonym to verify level 0 ↩

verify objective gradients *(no value)*: Synonym to verify level 1 ↩

verify yes *(no value)*: Synonym to verify level 3 ↩

weight on linear objective *(real)*: Composite objective weight ↩

This option controls the so-called composite objective technique. If the first solution obtained is infeasible, and if the objective function contains linear terms, and the objective weight w is positive, this technique is used. While trying to reduce the sum of infeasibilities, the method also attempts to optimize the linear portion of the objective. At each infeasible iteration, the objective function is defined to be

$$minimize_x\ sigma{*}w(c^T x)\ +\ (sum\ of\ infeasibilities)$$

where *sigma = 1* for minimization and *sigma = -1* for maximization and c is the linear portion of the objective. If an optimal solution is reached while still infeasible, w is reduced by a factor of 10. This helps to allow for the possibility that the initial w is too large. It also provides dynamic allowance for the fact the sum of infeasibilities is tending towards zero. The effect of w is disabled after five such reductions, or if a feasible solution is obtained. This option is intended mainly for linear programs. It is unlikely to be helpful if the objective function is nonlinear.

Default: `0.0`

5.26.10 Exit Conditions

This section discusses the exit codes printed by MINOS at the end of the optimization run.

EXIT – Optimal solution found

This is the message we all hope to see! It is certainly preferable to every other message. Of course it is quite possible that there are model formulation errors, which will (hopefully) lead to unexpected objective values and solutions. The reported optimum may be a local, and other much better optima may exist.

EXIT – The problem is infeasible

When the constraints are linear, this message can probably be trusted. Feasibility is measured with respect to the upper and lower bounds on the variables (the bounds on the slack variables correspond to the GAMS constraints). The message tells us that among all the points satisfying the general constraints $Ax + s = 0$, there is apparently no point that satisfies the bounds on x and s. Violations as small as the `FEASIBILITY TOLERANCE` are ignored, but at least one component of x or s violates a bound by more than the tolerance.

Note: Although the objective function is the sum of the infeasibilities, this sum will usually not have been *minimized* when MINOS recognizes the situation and exits. There may exist other points that have significantly lower sum of infeasibilities.

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation MINOS may relax the bounds on the slacks associated with nonlinear rows. This perturbation is allowed a fixed number of times. Normally a feasible point will be obtained relative to the perturbed constraints, and optimization can continue on the subproblem. However, if several consecutive subproblems require such perturbation, the problem is terminated and declared `INFEASIBLE`. Clearly this is an ad-hoc procedure. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

EXIT – The problem is unbounded (or badly scaled)

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased by an arbitrary amount without causing a basic variable to violate a bound. A simple way to diagnose such a model is to add an appropriate bound on the objective variable.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the `SCALE` option.

For nonlinear problems, MINOS monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (as judged by the `UNBOUNDED` parameter), the problem is terminated and declared `UNBOUNDED`. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

EXIT – User Interrupt

This exit code is a result of interrupting the optimization process by hitting `Ctrl-C`. Inside the IDE this is accomplished by hitting the `Interrupt` button. The solver will finish its current iteration, and return the current solution. This solution can be still intermediate infeasible or intermediate non-optimal.

EXIT – Too many iterations

The iteration limit was hit. Either the ITERLIM, or in some cases the ITERATIONS LIMIT or MAJOR ITERATION LIMIT was too small to solve the problem. In most cases increasing the GAMS ITERLIM option will resolve the problem. In other cases you will need to create a MINOS option file and set a MAJOR ITERATION LIMIT. The listing file will give more information regarding what limit was hit.

The GAMS iteration limit is displayed in the listing file under the section SOLVE SUMMARY. If the ITERLIM was hit, the message will look like:

```
ITERATION COUNT, LIMIT        10001      10000
```

EXIT – Resource Interrupt

The solver hit the RESLIM resource limit, which is a time limit. It returned the solution at that time, which may be still intermediate infeasible or intermediate non-optimal.

The GAMS resource limit is displayed in the listing file under the section SOLVE SUMMARY. If the GAMS RESLIM was hit, the message will look like:

```
RESOURCE USAGE, LIMIT       1001.570      1000.000
```

EXIT – The objective has not changed for many iterations

This is an emergency measure for the rare occasions when the solution procedure appears to be *cycling*. Suppose that a zero step is taken for several consecutive iterations, with a basis change occurring each time. It is theoretically possible for the set of basic variables to become the same as they were one or more iterations earlier. The same sequence of iterations would then occur *ad infinitum*.

EXIT – The Superbasics Limit is too small

The problem appears to be more non-linear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and an increase in the number of superbasics is needed. You can use the option SUPERBASICS LIMIT to increase the limit. See also option HESSIAN DIMENSION.

EXIT – Constraint and objective function could not be calculated

The function or gradient could not be evaluated. For example, this can occur when MINOS attempts to take a log or a square root of a negative number, when evaluating the expression x^y with $x \leq 0$, or when evaluating *exp(x)* for large x and the result is too large to store. The listing file will contain details about where and why evaluation errors occur. To fix this problem, add bounds so that all functions can be properly evaluated. E.g. if you have an expression x^y, add a lower bound X.LO=0.001 to your model.

In many cases the algorithm can recover from function evaluation errors, for instance if they happen in the line search while evaluating trial points. The message above appears in cases where the algorithm can not recover, and requires a reliable function or gradient evaluation.

EXIT – Function evaluation error limit

The limit of allowed function evaluation errors DOMLIM has been exceeded.

Function evaluation errors occur when MINOS attempts to evaluate the objective and/or constraints at points where these functions or their derivatives are not defined or where overflows occur. Some examples are given above. The listing file contains details about these errors.

The quick and dirty way to solve this is to increase the GAMS DOMLIM setting, but in general it is better to add bounds. E.g. if you have an expression x^y, then add a bound X.LO=0.001 to your model.

EXIT – The current point can not be improved

The line search failed. This can happen if the model is very nonlinear or if the functions are nonsmooth (using a DNLP model type).

If the model is non-smooth, consider a smooth approximation. It may be useful to check the scaling of the model and think more carefully about choosing a good starting point. Sometimes it can help to restart the model with full scaling turned on:

```
option nlp=minos;
solve m minimizing z using nlp; // this one gives "current point cannot be improved"
file fopt /minos.opt/;          // write option file
putclose fopt "scale all variables"/;
m.optfile=1;
solve m minimizing z using nlp; // solve with "scale all variables"
```

EXIT – Numerical error in trying to satisfy the linear constraints (or the linearized constraints)
The basis is very ill-conditioned.

This is often a scaling problem. Try the full scaling option *scale all variables* or, better yet, rescale the model in GAMS via the .scale suffix or by choosing more appropriate units for variables and RHS values.

EXIT – Not enough storage to solve the model

The amount of workspace allocated for MINOS to solve the model is insufficient. Consider increasing the GAMS option workfactor to increase the workspace allocated for MINOS to use. The listing file and log file (screen) will contain information about the current workspace allocation. Increasing the workfactor by 50% is a reasonable strategy.

EXIT– Systems error

This is a catch all return for other serious problems. Check the listing file for more messages. If needed rerun the model with OPTION SYSOUT=ON;.

5.27 MOSEK

MOSEK ApS, C/O Symbion Science Park, Fruebjergvej 3, Box 16, 2100 Copenhagen Ø, Denmark

5.27.1 Introduction

MOSEK is a software package for the solution of linear, mixed-integer linear, quadratic, mixed-integer quadratic, quadratically constraint, conic, and semidefinite mathematical optimization problems. MOSEK is particularly well suited for solving large-scale linear, convex quadratically constraint, and conic programs using an extremely efficient interior point algorithm.

These problem classes can be solved using an appropriate optimizer built into MOSEK. All the optimizers available in MOSEK are built for the solution of large-scale sparse problems. Current optimizers include:

- Interior-point optimizers for continuous and conic problems
- Simplex optimizer for linear problems
- Branch-and-cut optimizer for mixed-integer linear, quadratic, and conic problems

5.27.1.1 Licensing

Licensing of GAMS/MOSEK is similar to other GAMS solvers. MOSEK is licensed in two different ways:

- **GAMS/MOSEK:**
  All model types.
- **GAMS/MOSEK Solver Link:**
  Users must have a separate, licensed MOSEK system. For users who wish to use MOSEK within GAMS and also in other environments.

Attention

The free bare-bone link mode (previously GAMS/OSIMOSEK) that allowed to solve LP and MIP when the user had a separate MOSEK license installed has been removed. If you relied on using this bare-bone link option, then do not hesitate to contact sales@mosek.com to arrange for a GAMS/MOSEK Solver Link license.

For information regarding MOSEK standalone or interfacing MOSEK with other applications contact sales@mosek.com.

5.27.1.2 Solving Problems in Parallel

MOSEK can exploit multiple CPUs (or a CPU with multiple cores) to solve an optimization problem when using the interior-point or the mixed-integer optimizers.

This implies that whenever the MOSEK interior-point optimizer should solve an optimization problem, then it will try to divide the work so each CPU gets a share of the work. The user decides how many CPUs MOSEK should exploit. Unfortunately, it is not always easy to divide the work. Also some of the coordination work must occur in sequential. Therefore, the speed-up obtained when using multiple CPUs is highly problem dependent. However, as a rule of thumb, if the problem solves very quickly, i.e., in less than 60 seconds, then there is no advantage in using the parallel option.

The parameter MSK_IPAR_NUM_THREADS sets the number of threads (and therefore the number of CPU's) that the optimizer will use.

5.27.1.3 Infeasible/Unbounded Models

Farkas Certificates

MOSEK determines if either the primal or the dual problem is infeasible by means of a Farkas certificate. In such a case MOSEK returns a certificate indicating primal or dual infeasibility.

The primal infeasibility certificate indicates a primal infeasible model. For a minimization problem

$$\begin{array}{ll} \text{minimize} & \langle c, x \rangle \\ \text{subject to} & Ax = b, \\ & x \geq 0, \end{array}$$

the primal infeasibility certificate is the solution y satisfying $A'y \leq 0$ and $\langle b, y \rangle > 0$.

A primal infeasibility certificate is reported in the marginal records of the variables and equations. As no primal solution is available in this case, the level values for variables and equations and the objective function value are set to 0 (setting them to NA would be more appropriate, but GAMS does not support this well). At the moment, primal infeasibility certificate are not available for conic programs.

Since GAMS reports all model statuses in the primal space, the notion of dual infeasibility does not exist and GAMS reports a status of unboundedness, which assumes that the primal problem is feasible. Although GAMS reports the primal as unbounded, there is the possibility that both the primal *and* dual problem are infeasible. To check if this is the case, the user can set appropriate lower and upper bounds on the objective variable, using the `(variable).LO` and `(variable).UP` suffixes and resolve.

The dual infeasibility certificate is reported in the level values for the variables. As no dual solution exists, the marginal values for both variables and equations are set to NA.

For more details on primal and dual infeasibility certificates see the MOSEK Modeling Cookbook.

Infeasibility Report

MOSEK has some facilities for diagnosing the cause of a primal or dual infeasibility. They can be turned on using the parameter setting MSK_IPAR_INFEAS_REPORT_AUTO. This causes MOSEK to print a report about an infeasible subset of the constraints, when an infeasibility is encountered. Moreover, the parameter MSK_IPAR_INFEAS_REPORT_LEVEL controls the amount of information presented in the infeasibility report. We will use the **TRNSPORT example from the GAMS Model Library** with increased demand ($b(j) \leftarrow 1.6\, b(j)$) to make the model infeasible. MOSEK produces the following infeasibility report:

```
MOSEK PRIMAL INFEASIBILITY REPORT.

Problem status: The problem is primal infeasible

The following constraints are involved in the primal infeasibility.

Index    Name                  Lower bound      Upper bound      Dual lower       Dual upper
1        supply(seattle)       none             3.500000e+002    0.000000e+000    1.000000e+000
2        supply(san-diego)     none             6.000000e+002    0.000000e+000    1.000000e+000
3        demand(new-york)      5.200000e+002    none             1.000000e+000    0.000000e+000
4        demand(chicago)       4.800000e+002    none             1.000000e+000    0.000000e+000
5        demand(topeka)        4.400000e+002    none             1.000000e+000    0.000000e+000

The following bound constraints are involved in the infeasibility.

Index    Name                  Lower bound      Upper bound      Dual lower       Dual upper
```

The report indicates which constraints and bounds are causing the infeasibility. In this case, the constraints causing infeasibility are `supply` and `demand`. The values in the columns `Dual lower` and `Dual upper` are also useful, because if the dual lower value is different from zero for a constraint, then it implies that the lower bound on the constraint is important for the infeasibility. Similarly, if the dual upper value is different from zero on a constraint, then this implies the upper bound on the constraint is important for infeasibility.

5.27.1.4 Conic Programming

MOSEK is well suited for solving generalized linear programs involving certain conic constraints.

For an overview of quadratic conic programming and how these conic constraints are implemented in GAMS, see the Section Conic Programming in the GAMS User's Guide. Note, that for Mosek no variable can appear in more than one conic constraint.

Additionally, the primal power cone, defined as

$$x_0^{\alpha} x_1^{(1-\alpha)} \geq \sqrt{\sum_{i=2}^{n} x_i^2}, \quad x_0, x_1 \geq 0,$$

with $\alpha \in (0,1)$, and the primal exponential cone, defined as

$$x_0 \geq x_1 \exp(x_2/x_1), \quad x_0, x_1 \geq 0,$$

are available.

Since GAMS does not offer capabilities to directly restrict a variable to one of these cones, the GAMS/MOSEK link tries to detect the above algebra from a general nonlinear equation. For example, the following GAMS code should work with MOSEK:

```
Set i / 0*10 /;
Variable x(i);
Equations e1, e2;
Scalar alpha;
e1.. x('0')**alpha * x('1')**(1-alpha) =G= sqrt(sum(i$(ord(i)>2), sqr(x(i))));
e2.. x('0') =G= x('1') * exp(x('2') / x('1'));
x.lo('0') = 0;
x.lo('1') = 0;
```

See also testlib models **powercone1**, **powercone2**, and **expcone1**.

5.27.2 Solver Options

MOSEK works like other GAMS solvers, and many options can be set in the GAMS model (see GAMS Options). The most relevant GAMS options are reslim, nodlim, iterlim (iteration limit for simplex and interior point algorithms), optca, optcr, and optfile. A description of all available GAMS options can be found in GAMS Options and Solver related options.

We remark that MOSEK contains many complex solver options, many of which require a deep understanding of the algorithms used. For information on how to use a GAMS/Mosek options file, see The Solver Option File. For example, an option file

```
MSK_IPAR_INTPNT_MAX_ITERATIONS                20
MSK_IPAR_INTPNT_SCALING         MSK_SCALING_NONE
```

limits the number of interior-point iterations to 20 and disables scaling.

In the following, we summarize the MOSEK options that are available through the GAMS/MOSEK interface. For details, see Detailed Descriptions of MOSEK Options.

5.27.2.1 General

Option	Description	Default
MSK_DPAR_OPTIMIZER_MAX_TIME	Maximum amount of time the optimizer is allowed to spent on the optimization.	GAMS ResLim
MSK_DPAR_SEMIDEFINITE_TOL_APPROX	Tolerance to define a matrix to be positive semidefinite.	1e-10
MSK_IPAR_AUTO_SORT_A_BEFORE_OPT	Controls whether the elements in each column of the coefficient matrix are sorted before an optimization is performed.	MSK_OFF
MSK_IPAR_NUM_THREADS	Controls the number of threads employed by the optimizer.	GAMS Threads
MSK_IPAR_OPTIMIZER	The parameter controls which optimizer is used to optimize the task.	MSK_OPTIMIZER_FREE
MSK_IPAR_TIMING_LEVEL	Controls the amount of timing performed inside MOSEK.	1
MSK_SPAR_PARAM_READ_FILE_NAME	Modifications to the parameter database is read from this file.	

5.27.2.2 Problem Data

Option	Description	Default
MSK_DPAR_CHECK_CONVEXITY_REL_TOL	Not in use.	1e-10
MSK_DPAR_DATA_SYM_MAT_TOL	Absolute zero tolerance for elements in in symmetric matrices.	1e-12
MSK_DPAR_DATA_SYM_MAT_TOL_HUGE	An element in a symmetric matrix which is larger than this value in absolute size causes an error.	1e+20
MSK_DPAR_DATA_SYM_MAT_TOL_LARGE	An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed.	1e+10
MSK_DPAR_DATA_TOL_AIJ_HUGE	An element in the constraint matrix which is larger than this value in absolute size causes an error.	1e+20
MSK_DPAR_DATA_TOL_AIJ_LARGE	An element in the constraint matrix which is larger than this value in absolute size causes a warning message.	1e+10
MSK_DPAR_DATA_TOL_BOUND_INF	Any bound which in absolute value is greater than this parameter is considered infinite.	1e+16
MSK_DPAR_DATA_TOL_BOUND_WRN	If a bound value is larger than this value in absolute size, then a warning message is issued.	1e+08
MSK_DPAR_DATA_TOL_CJ_LARGE	A coefficient in the objective function which is larger than this value in absolute terms causes a warning message.	1e+08
MSK_DPAR_DATA_TOL_C_HUGE	A coefficient in the objective function which is larger than the value in absolute terms is considered to be huge and generates an error.	1e+16
MSK_DPAR_DATA_TOL_QIJ	Absolute zero tolerance for coefficients of quadratic terms.	1e-16

Option	Description	Default
MSK_DPAR_DATA_TOL_X	Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical.	1e-08
MSK_DPAR_LOWER_OBJ_CUT	Lower objective limit.	-1e+30
MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH	Lower objective limit threshold.	-5e+29
MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL	This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems.	1e-15
MSK_DPAR_UPPER_OBJ_CUT	Upper objective limit.	1e+30
MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH	Upper objective limit threshold.	5e+29
QEXTRACTALG	Switch to choose extraction algorithm for quadratic equations in GAMS interface.	0
SDPCHECKVARS	Switch to disable checking that for every entry of a PSD matrix variable also a corresponding GAMS variable is present.	1

5.27.2.3 Presolving

Option	Description	Default
MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP	Absolute tolerance employed by the linear dependency checker.	1e-06
MSK_DPAR_PRESOLVE_TOL_AIJ	Absolute zero tolerance employed for constraint coefficients in presolve.	1e-12
MSK_DPAR_PRESOLVE_TOL_PRIMAL_INFEAS_PERTURBATION	The presolve is allowed to perturb a bound on a constraint or variable by this amount if it removes an infeasibility.	1e-06
MSK_DPAR_PRESOLVE_TOL_REL_LINDEP	Relative tolerance employed by the linear dependency checker.	1e-10
MSK_DPAR_PRESOLVE_TOL_S	Absolute zero tolerance employed for dual variables in presolve.	1e-08
MSK_DPAR_PRESOLVE_TOL_X	Absolute zero tolerance employed for primal variables in presolve.	1e-08
MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL	Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of presolve.	-1
MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES	Control the maximum number of times the eliminator is tried.	-1
MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH	Controls the linear dependency check, which is potentially computationally expensive.	100
MSK_IPAR_PRESOLVE_LINDEP_NEW	Controls whether a new experimental linear dependency checker is employed.	MSK_OFF

Option	Description	Default
MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH	Controls the linear dependency check, which is potentially computationally expensive.	100
MSK_IPAR_PRESOLVE_LINDEP_USE	Controls whether the linear constraints are checked for linear dependencies.	MSK_ON
MSK_IPAR_PRESOLVE_MAX_NUM_PASS	Control the maximum number of times presolve passes over the problem.	-1
MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS	Controls the maximum number of reductions performed by the presolve.	-1
MSK_IPAR_PRESOLVE_USE	Controls whether the presolve is applied to a problem before it is optimized.	MSK_PRESOLVE_MODE_FREE

5.27.2.4 Simplex Optimizer

Option	Description	Default
MSK_DPAR_BASIS_REL_TOL_S	Maximum relative dual bound violation allowed in an optimal basic solution.	1e-12
MSK_DPAR_BASIS_TOL_S	Maximum absolute dual bound violation in an optimal basic solution.	1e-06
MSK_DPAR_BASIS_TOL_X	Maximum absolute primal bound violation allowed in an optimal basic solution.	1e-06
MSK_DPAR_SIMPLEX_ABS_TOL_PIV	Absolute pivot tolerance employed by the simplex optimizers.	1e-07
MSK_DPAR_SIM_LU_TOL_REL_PIV	Relative pivot tolerance for LU factorization in simplex optimizers and basis identification.	0.01
MSK_IPAR_SIM_BASIS_FACTOR_USE	Controls whether an LU factorization of the basis is used in a hot-start.	MSK_ON
MSK_IPAR_SIM_DEGEN	Controls how aggressively degeneration is handled.	MSK_SIM_DEGEN_FREE
MSK_IPAR_SIM_DUAL_CRASH	Controls whether crashing is performed in the dual simplex optimizer.	90
MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION	Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the dual simplex.	50
MSK_IPAR_SIM_DUAL_SELECTION	Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer.	MSK_SIM_SELECTION_FREE
MSK_IPAR_SIM_EXPLOIT_DUPVEC	Controls if the simplex optimizers are allowed to exploit duplicated columns.	MSK_SIM_EXPLOIT_DUPVEC_OFF

Option	Description	Default
MSK_IPAR_SIM_HOTSTART	Controls the type of hot-start that the simplex optimizer perform.	MSK_SIM_HOTSTART_FREE
MSK_IPAR_SIM_HOTSTART_LU	Determines if the simplex optimizer should exploit the initial factorization.	MSK_ON
MSK_IPAR_SIM_MAX_ITERATIONS	Maximum number of iterations that can be used by a simplex optimizer.	GAMS IterLim
MSK_IPAR_SIM_MAX_NUM_SETBACKS	Controls how many set-backs are allowed within a simplex optimizer.	250
MSK_IPAR_SIM_NON_SINGULAR	Controls if the simplex optimizer ensures a non-singular basis, if possible.	MSK_ON
MSK_IPAR_SIM_PRIMAL_CRASH	Controls whether crashing is performed in the primal simplex optimizer.	90
MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION	Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the primal simplex.	50
MSK_IPAR_SIM_PRIMAL_SELECTION	Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer.	MSK_SIM_SELECTION_FREE
MSK_IPAR_SIM_REFORMULATION	Controls if the simplex optimizers are allowed to reformulate the problem.	MSK_SIM_REFORMULATION_OFF
MSK_IPAR_SIM_SAVE_LU	Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis.	MSK_OFF
MSK_IPAR_SIM_SCALING	Controls how much effort is used in scaling the problem before a simplex optimizer is used.	MSK_SCALING_FREE
MSK_IPAR_SIM_SCALING_METHOD	Controls how the problem is scaled before a simplex optimizer is used.	MSK_SCALING_METHOD_POW2
MSK_IPAR_SIM_SEED	Sets the random seed used for randomization in the simplex optimizers.	23456
MSK_IPAR_SIM_SOLVE_FORM	Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer.	MSK_SOLVE_FREE
MSK_IPAR_SIM_STABILITY_PRIORITY	Controls how high priority the numerical stability should be given.	50
MSK_IPAR_SIM_SWITCH_OPTIMIZER	Controls the simplex behavior.	MSK_OFF

5.27.2.5 Interior Point Optimizer and Basis Identification

Option	Description	Default
MSK_DPAR_INTPNT_CO_TOL_DFEAS	Dual feasibility tolerance used by the interior-point optimizer for conic problems.	1e-08
MSK_DPAR_INTPNT_CO_TOL_INFEAS	Infeasibility tolerance used by the interior-point optimizer for conic problems.	1e-12
MSK_DPAR_INTPNT_CO_TOL_MU_RED	Relative complementarity gap tolerance used by the interior-point optimizer for conic problems.	1e-08
MSK_DPAR_INTPNT_CO_TOL_NEAR_REL	Termination tolerance multiplier that is used if no accurate solution can be found.	1
MSK_DPAR_INTPNT_CO_TOL_PFEAS	Primal feasibility tolerance used by the interior-point optimizer for conic problems.	1e-08
MSK_DPAR_INTPNT_CO_TOL_REL_GAP	Relative gap termination tolerance used by the interior-point optimizer for conic problems.	1e-08
MSK_DPAR_INTPNT_QO_TOL_DFEAS	Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem.	1e-08
MSK_DPAR_INTPNT_QO_TOL_INFEAS	Infeasibility tolerance used by the interior-point optimizer for quadratic problems.	1e-12
MSK_DPAR_INTPNT_QO_TOL_MU_RED	Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems.	1e-08
MSK_DPAR_INTPNT_QO_TOL_NEAR_REL	Termination tolerance multiplier that is used if no accurate solution can be found.	1
MSK_DPAR_INTPNT_QO_TOL_PFEAS	Primal feasibility tolerance used by the interior-point optimizer for quadratic problems.	1e-08
MSK_DPAR_INTPNT_QO_TOL_REL_GAP	Relative gap termination tolerance used by the interior-point optimizer for quadratic problems.	1e-08
MSK_DPAR_INTPNT_TOL_DFEAS	Dual feasibility tolerance used by the interior-point optimizer for linear problems.	1e-08
MSK_DPAR_INTPNT_TOL_DSAFE	Controls the initial dual starting point used by the interior-point optimizer.	1
MSK_DPAR_INTPNT_TOL_INFEAS	Infeasibility tolerance used by the interior-point optimizer for linear problems.	1e-10
MSK_DPAR_INTPNT_TOL_MU_RED	Relative complementarity gap tolerance used by the interior-point optimizer for linear problems.	1e-16
MSK_DPAR_INTPNT_TOL_PATH	Controls how close the interior-point optimizer follows the central path.	1e-08

Option	Description	Default
MSK_DPAR_INTPNT_TOL_PFEAS	Primal feasibility tolerance used by the interior-point optimizer for linear problems.	1e-08
MSK_DPAR_INTPNT_TOL_PSAFE	Controls the initial primal starting point used by the interior-point optimizer.	1
MSK_DPAR_INTPNT_TOL_REL_GAP	Relative gap termination tolerance used by the interior-point optimizer for linear problems.	1e-08
MSK_DPAR_INTPNT_TOL_REL_STEP	Relative step size to the boundary for linear and quadratic optimization problems.	0.9999
MSK_DPAR_INTPNT_TOL_STEP_SIZE	Step size tolerance.	1e-06
MSK_IPAR_BI_CLEAN_OPTIMIZER	Controls which simplex optimizer is used in the clean-up phase.	MSK_OPTIMIZER_FREE
MSK_IPAR_BI_IGNORE_MAX_ITER	Controls if basis identification is performed under certain conditions.	MSK_OFF
MSK_IPAR_BI_IGNORE_NUM_ERROR	Turns on basis identification if interior-point optimizer is terminated due to a numerical problem.	MSK_ON
MSK_IPAR_BI_MAX_ITERATIONS	Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification.	1000000
MSK_IPAR_INTPNT_BASIS	Controls whether the interior-point optimizer also computes an optimal basis.	MSK_BI_ALWAYS
MSK_IPAR_INTPNT_DIFF_STEP	Controls whether different step sizes are allowed in the primal and dual space.	MSK_ON
MSK_IPAR_INTPNT_MAX_ITERATIONS	Controls the maximum number of iterations allowed in the interior-point optimizer.	GAMS IterLim
MSK_IPAR_INTPNT_MAX_NUM_COR	Controls the maximum number of correctors allowed by the multiple corrector procedure.	-1
MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS	Maximum number of steps to be used by the iterative refinement of the search direction.	-1
MSK_IPAR_INTPNT_OFF_COL_TRH	Controls how aggressively offending columns are detected in the Jacobian of the constraint matrix.	40
MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS	The GP ordering is dependent on a random seed.	0
MSK_IPAR_INTPNT_ORDER_METHOD	Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system.	MSK_ORDER_METHOD_FREE
MSK_IPAR_INTPNT_REGULARIZATION_USE	Controls whether regularization is allowed.	MSK_ON
MSK_IPAR_INTPNT_SCALING	Controls how the problem is scaled before the interior-point optimizer is used.	MSK_SCALING_FREE

Option	Description	Default
MSK_IPAR_INTPNT_SOLVE_FORM	Controls whether the primal or the dual problem is solved.	MSK_SOLVE_FREE
MSK_IPAR_INTPNT_STARTING_POINT	Starting point used by the interior-point optimizer.	MSK_STARTING_POINT_FREE

5.27.2.6 Mixed Integer Optimizer

Option	Description	Default
FIXOPTFILE	Name of option file which is read just before solving the fixed problem.	
MSK_DPAR_MIO_MAX_TIME	This parameter limits the maximum time spent by the mixed-integer optimizer.	-1
MSK_DPAR_MIO_REL_GAP_CONST	This value is used to compute the relative gap for the solution to an integer optimization problem.	1e-10
MSK_DPAR_MIO_TOL_ABS_GAP	Absolute optimality tolerance employed by the mixed-integer optimizer.	GAMS OptCA
MSK_DPAR_MIO_TOL_ABS_RELAX_INT	Absolute relaxation tolerance of the integer constraints.	1e-05
MSK_DPAR_MIO_TOL_FEAS	Feasibility tolerance for mixed integer solver.	1e-06
MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT	If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation.	0
MSK_DPAR_MIO_TOL_REL_GAP	Relative optimality tolerance employed by the mixed-integer optimizer.	GAMS OptCR
MSK_IPAR_MIO_BRANCH_DIR	Controls whether the mixed-integer optimizer is branching up or down by default.	MSK_BRANCH_DIR_FREE
MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION	If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used.	MSK_OFF
MSK_IPAR_MIO_CONSTRUCT_SOL	Whether to construct an initial solution from starting point	MSK_OFF
MSK_IPAR_MIO_CUT_CLIQUE	Controls whether clique cuts should be generated.	MSK_ON
MSK_IPAR_MIO_CUT_CMIR	Controls whether mixed integer rounding cuts should be generated.	MSK_ON
MSK_IPAR_MIO_CUT_GMI	Controls whether GMI cuts should be generated.	MSK_ON
MSK_IPAR_MIO_CUT_IMPLIED_BOUND	Controls whether implied bound cuts should be generated.	MSK_ON
MSK_IPAR_MIO_CUT_KNAPSACK_COVER	Controls whether knapsack cover cuts should be generated.	MSK_ON

Option	Description	Default
MSK_IPAR_MIO_CUT_LIPRO	Controls whether lift-and-project cuts should be generated.	MSK_OFF
MSK_IPAR_MIO_CUT_SELECTION_LEVEL	Controls how aggressively generated cuts are selected to be included in the relaxation.	-1
MSK_IPAR_MIO_DATA_PERMUTATION_METHOD	Controls what problem data permutation method is appplied to mixed-integer problems.	MSK_MIO_DATA_PERMUTATION_METHOD_NONE
MSK_IPAR_MIO_DUAL_RAY_ANALYSIS_LEVEL	Controls the amount of symmetry detection and handling employed by the mixed-integer optimizer in presolve.	-1
MSK_IPAR_MIO_FEASPUMP_LEVEL	Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer.	-1
MSK_IPAR_MIO_HEURISTIC_LEVEL	Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution.	-1
MSK_IPAR_MIO_MAX_NUM_BRANCHES	Maximum number of branches allowed during the branch and bound search.	-1
MSK_IPAR_MIO_MAX_NUM_RELAXS	Maximum number of relaxations allowed during the branch and bound search.	GAMS NodLim
MSK_IPAR_MIO_MAX_NUM_RESTARTS	Maximum number of restarts allowed during the branch and bound search.	0
MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS	Maximum number of cut separation rounds at the root node.	100
MSK_IPAR_MIO_MAX_NUM_SOLUTIONS	The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located.	-1
MSK_IPAR_MIO_MEMORY_EMPHASIS_LEVEL	Controls how much emphasis is put on reducing memory usage.	0
MSK_IPAR_MIO_MIN_REL	Number of times a variable must have been branched on for its pseudocost to be considered reliable.	5
MSK_IPAR_MIO_NODE_OPTIMIZER	Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer.	MSK_OPTIMIZER_FREE
MSK_IPAR_MIO_NODE_SELECTION	Controls the node selection strategy employed by the mixed-integer optimizer.	MSK_MIO_NODE_SELECTION_FREE
MSK_IPAR_MIO_NUMERICAL_EMPHASIS_LEVEL	Controls how much emphasis is put on reducing numerical problems possibly at the expense of solution speed.	0
MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE	Enables or disables perspective reformulation in presolve.	MSK_ON
MSK_IPAR_MIO_PRESOLVE_AGGREGATOR_USE	Controls if the aggregator should be used.	MSK_ON

Option	Description	Default
MSK_IPAR_MIO_PROBING_LEVEL	Controls the amount of probing employed by the mixed-integer optimizer in presolve.	-1
MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT	Use objective domain propagation.	MSK_OFF
MSK_IPAR_MIO_QCQO_REFORMULATION_METHOD	Controls what reformulation method is applied to mixed-integer quadratic problems.	MSK_MIO_QCQO_REFORMULATION_METHOD_FREE
MSK_IPAR_MIO_RINS_MAX_NODES	Controls the maximum number of nodes allowed in each call to the RINS heuristic.	-1
MSK_IPAR_MIO_ROOT_OPTIMIZER	Controls which optimizer is employed at the root node in the mixed-integer optimizer.	MSK_OPTIMIZER_FREE
MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL	Controls whether presolve can be repeated at root node.	-1
MSK_IPAR_MIO_SEED	Sets the random seed used for randomization in the mixed integer optimizer.	42
MSK_IPAR_MIO_SYMMETRY_LEVEL	Controls the amount of symmetry detection and handling employed by the mixed-integer optimizer in presolve.	-1
MSK_IPAR_MIO_VAR_SELECTION	Controls the variable selection strategy employed by the mixed-integer optimizer.	MSK_MIO_VAR_SELECTION_FREE
MSK_IPAR_MIO_VB_DETECTION_LEVEL	Controls how much effort is put into detecting variable bounds.	-1
SOLVEFINAL	Switch to resolve the problem with fixed discrete variables after the MOSEK optimizer finished.	1

5.27.2.7 Infeasibility Analyser for Continuous Problems

Option	Description	Default
MSK_DPAR_ANA_SOL_INFEAS_TOL	If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer.	1e-06
MSK_IPAR_INFEAS_PREFER_PRIMAL	If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on.	MSK_ON
MSK_IPAR_INFEAS_REPORT_AUTO	Controls whether an infeasibility report is automatically produced after the optimization if the problem is primal or dual infeasible.	MSK_OFF
MSK_IPAR_INFEAS_REPORT_LEVEL	Controls the amount of information presented in an infeasibility report.	1

5.27.2.8 Output

Option	Description	Default
MSK_IPAR_LOG	Controls the amount of log information.	10
MSK_IPAR_LOG_BI	Controls the amount of output printed by the basis identification procedure.	1
MSK_IPAR_LOG_BI_FREQ	Controls logging frequency of the basis identification	2500
MSK_IPAR_LOG_FEAS_REPAIR	Controls the amount of output printed when performing feasibility repair.	1
MSK_IPAR_LOG_INFEAS_ANA	Controls amount of output printed by the infeasibility analyzer procedures.	1
MSK_IPAR_LOG_INTPNT	Controls amount of output printed by the interior-point optimizer.	1
MSK_IPAR_LOG_MIO	Controls the log level for the mixed-integer optimizer.	4
MSK_IPAR_LOG_MIO_FREQ	Controls how frequent the mixed-integer optimizer prints the log line.	10
MSK_IPAR_LOG_ORDER	If turned on, then factor lines are added to the log.	1
MSK_IPAR_LOG_PRESOLVE	Controls amount of output printed by the presolve procedure.	1
MSK_IPAR_LOG_RESPONSE	Controls amount of output printed when response codes are reported.	0
MSK_IPAR_LOG_SIM	Controls amount of output printed by the simplex optimizer.	4
MSK_IPAR_LOG_SIM_FREQ	Controls simplex optimizer logging frequency.	1000
MSK_IPAR_LOG_STORAGE	When turned on, MOSEK prints messages regarding the storage usage and allocation.	0
MSK_IPAR_OPF_WRITE_HEADER	Write a text header with date and MOSEK version in an OPF file.	MSK_ON
MSK_IPAR_OPF_WRITE_HINTS	Write a hint section with problem dimensions in the beginning of an OPF file.	MSK_ON
MSK_IPAR_OPF_WRITE_LINE_LENGTH	Aim to keep lines in OPF files not much longer than this.	80
MSK_IPAR_OPF_WRITE_PARAMETERS	Write a parameter section in an OPF file.	MSK_OFF
MSK_IPAR_OPF_WRITE_PROBLEM	Write objective, constraints, bounds etc.	MSK_ON
MSK_IPAR_OPF_WRITE_SOLUTIONS	Enable inclusion of solutions in the OPF files.	MSK_OFF
MSK_IPAR_OPF_WRITE_SOL_BAS	Whether to include basic solution in OPF files.	MSK_ON
MSK_IPAR_OPF_WRITE_SOL_ITG	Whether to include integer solution in OPF files.	MSK_ON
MSK_IPAR_OPF_WRITE_SOL_ITR	Whether to include interior solution in OPF files.	MSK_ON
MSK_IPAR_PTF_WRITE_PARAMETERS	If enabled, then the parameters section is written.	MSK_OFF

Option	Description	Default
MSK_IPAR_PTF_WRITE_SOLUTIONS	If enabled, then the solution section is written if any solutions are available, otherwise solution section is not written even if solutions are available.	MSK_OFF
MSK_IPAR_PTF_WRITE_TRANSFORM	If enabled, then constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated.	MSK_ON
MSK_IPAR_WRITE_COMPRESSION	Controls whether the data file is compressed while it is written.	9
MSK_IPAR_WRITE_DATA_PARAM	If this option is turned on the parameter settings are written to the data file as parameters.	MSK_OFF
MSK_IPAR_WRITE_GENERIC_NAMES	Controls whether generic names should be used instead of user-defined names when writing to the data file.	MSK_ON
MSK_IPAR_WRITE_GENERIC_NAMES_IO	Index origin used in generic names.	1
MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS	Controls if the writer ignores incompatible problem items when writing files.	MSK_OFF
MSK_IPAR_WRITE_JSON_INDENTATION	When set, the JSON task and solution files are written with indentation for better readability.	MSK_OFF
MSK_IPAR_WRITE_LP_FULL_OBJ	Write all variables, including the ones with 0-coefficients, in the objective.	MSK_ON
MSK_IPAR_WRITE_LP_LINE_WIDTH	Maximum width of line in an LP file written by MOSEK.	80
MSK_IPAR_WRITE_MPS_FORMAT	Controls in which format the MPS is written.	MSK_MPS_FORMAT_FREE
MSK_IPAR_WRITE_MPS_INT	Controls if marker records are written to the MPS file to indicate whether variables are integer restricted.	MSK_ON
MSK_IPAR_WRITE_TASK_INC_SOL	Controls whether the solutions are stored in the task file too.	MSK_ON
MSK_IPAR_WRITE_XML_MODE	Controls if linear coefficients should be written by row or column when writing in the XML file format.	MSK_WRITE_XML_MODE_ROW
MSK_SPAR_DATA_FILE_NAME	If set, problem data is written to this file. File extension determines format.	
MSK_SPAR_PARAM_WRITE_FILE_NAME	The parameter database is written to this file.	
MSK_SPAR_WRITE_LP_GEN_VAR_NAME	Sometimes when an LP file is written additional variables must be inserted.	"xmskgen"
SDPSOLUFILE	Name of GDX file to write primal solution of all PSD matrix variables and dual solution for PSD constraints.	

5.27.3 The MOSEK Log File

The MOSEK log output gives much useful information about the current solver progress and individual phases.

5.27.3.1 Log Using the Interior Point Optimizer

The following is a MOSEK log output from running the **TRNSPORT** model from the GAMS Model Library:

```
Optimizer started.
Interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries                  : 0                 time                   : 0.00
Lin. dep.  - tries                  : 1                 time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated. Time: 0.00
Optimizer  - threads                : 1
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 5
Optimizer  - Cones                  : 0
Optimizer  - Scalar variables       : 11                conic                  : 0
Optimizer  - Semi-definite variables: 0                 scalarized             : 0
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 11                after factor           : 12
Factor     - dense dim.             : 0                 flops                  : 1.80e+02
```

The first part gives information about the presolve (if used). The main log follows:

```
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   6.0e+02  1.0e+02  1.0e+02  1.00e+02   1.053000000e+00   0.000000000e+00   1.0e+02  0.00
1   5.9e+02  1.3e+02  2.6e+02  0.00e+00   3.063797526e+00   2.650041612e+02   2.4e+02  0.00
2   4.6e+01  1.0e+01  2.0e+01  -9.59e-01  3.650704301e+01   2.594816940e+02   1.9e+01  0.00
3   3.9e-01  8.7e-02  1.7e-01  -4.39e-01  1.604589379e+02   2.276036163e+02   1.6e-01  0.00
4   2.7e-02  6.0e-03  1.2e-02  9.62e-01   1.627664502e+02   1.676438787e+02   1.1e-02  0.00
5   2.2e-03  4.9e-04  9.7e-04  1.04e+00   1.585004810e+02   1.591499235e+02   8.9e-04  0.00
6   3.1e-04  6.9e-05  1.4e-04  1.01e+00   1.546312243e+02   1.547272945e+02   1.2e-04  0.00
7   2.9e-05  6.5e-06  1.3e-05  1.01e+00   1.536906429e+02   1.536999628e+02   1.2e-05  0.00
8   7.6e-08  1.7e-08  3.4e-08  1.00e+00   1.536751995e+02   1.536752387e+02   3.1e-08  0.00
9   7.5e-12  1.7e-12  3.4e-12  1.00e+00   1.536750000e+02   1.536750000e+02   3.1e-12  0.00
Basis identification started.
Primal basis identification phase started.
ITER      TIME
1         0.00
Primal basis identification phase terminated. Time: 0.00
Dual basis identification phase started.
ITER      TIME
0         0.00
Dual basis identification phase terminated. Time: 0.00
Basis identification terminated. Time: 0.00
Interior-point optimizer terminated. Time: 0.00.
```

```
Optimizer terminated. Time: 0.00

Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 1.5367500002e+02    nrm: 6e+02    Viol.  con: 3e-10    var: 0e+00
  Dual.    obj: 1.5367500002e+02    nrm: 2e-01    Viol.  con: 0e+00    var: 6e-11

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 1.5367500000e+02    nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: 1.5367500002e+02    nrm: 2e-01    Viol.  con: 0e+00    var: 5e-11

Return code - 0 [MSK_RES_OK]: No error occurred.
```

The last section gives details about the model and solver status, primal and dual feasibilities, as well as solver resource times. Furthermore, the log gives information about the basis identification phase. Some of this information is listed in the GAMS solve summary in the model listing (.LST) file as well.

The fields in the main MOSEK log output are:

Field	Description
ITE	The number of the current iteration.
PFEAS	Primal feasibility.
DFEAS	Dual feasibility.
GFEAS	The numbers in this column should converge monotonically toward to zero but may stall at low level due to rounding errors.
PRSTATUS	This number converges to 1 if the problem has an optimal solution whereas it converges to -1 if that is not the case.
POBJ	Current objective function value of primal problem.
DOBJ	Current objective function value of dual problem.
MU	Relative complementary gap.
TIME	Current elapsed solving time in seconds.

5.27.3.2 Log Using the Simplex Optimizer

Below is a log output running the **TRNSPORT** model from the GAMS Model Library using the MOSEK simplex optimizer.

```
Reading parameter(s) from "mosek.opt"
>>  MSK_IPAR_OPTIMIZER MSK_OPTIMIZER_DUAL_SIMPLEX
Finished reading from "mosek.opt"

Optimizer started.
Simplex optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries                  : 0                 time                   : 0.00
Lin. dep.  - tries                  : 1                 time                   : 0.00
Lin. dep.  - number                 : 0
```

```
Presolve terminated. Time: 0.00
Dual simplex optimizer started.
Dual simplex optimizer setup started.
Dual simplex optimizer setup terminated.
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 5
Optimizer  - Scalar variables       : 6                 conic                  : 0
Optimizer  - hotstart               : no
ITER      DEGITER(%)  PFEAS       DFEAS       POBJ                  DOBJ                  TIME      T
0         0.00        NA          0.00e+00    NA                    0.0000000000e+00      0.00      0
4         20.00       NA          0.00e+00    NA                    1.5367501014e+02      0.00      0
Dual simplex optimizer terminated.
Simplex optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 1.5367500000e+02    nrm: 6e+02    Viol.  con: 0e+00    var: 0e+00
  Dual.    obj: 1.5367500000e+02    nrm: 2e-01    Viol.  con: 0e+00    var: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.
```

The fields in the main MOSEK log output are:

Field	Description
ITER	Current number of iterations.
DEGITER(%)	Current percentage of degenerate iterations.
P/DFEAS	Current primal and dual infeasibility.
P/DOBJ	Current primal and dual objective value.
TIME	Current elapsed solving time in seconds.
TOTTIME	Total elapsed solving time in seconds.

5.27.3.3 Log Using the Mixed Integer Optimizer

Below is a log output running the model **CUBE** from the GAMS model library using the MOSEK mixed-integer optimizer.

```
Optimizer started.
Mixed integer optimizer started.
Threads used: 1
Presolve started.
Presolve terminated. Time = 0.00
Presolved problem: 76 variables, 99 constraints, 419 non-zeros
Presolved problem: 0 general integer, 27 binary, 49 continuous
Clique table size: 0
BRANCHES RELAXS   ACT_NDS  DEPTH    BEST_INT_OBJ         BEST_RELAX_OBJ       REL_GAP(%)  TIME
0        1        0        0        NA                   0.0000000000e+00     NA          0.0
0        1        0        0        6.0000000000e+00     0.0000000000e+00     100.00      0.0
Cut generation started.
0        2        0        0        6.0000000000e+00     0.0000000000e+00     100.00      0.0
Cut generation terminated. Time = 0.00
```

```
15       18       1        0        6.0000000000e+00     0.0000000000e+00     100.00       0.0
31       34       1        0        4.0000000000e+00     0.0000000000e+00     100.00       0.0
53       56       1        0        4.0000000000e+00     0.0000000000e+00     100.00       0.0
83       86       1        0        4.0000000000e+00     0.0000000000e+00     100.00       0.0
98       101      16       8        4.0000000000e+00     0.0000000000e+00     100.00       0.0
114      117      28       9        4.0000000000e+00     0.0000000000e+00     100.00       0.0
142      145      44       6        4.0000000000e+00     0.0000000000e+00     100.00       0.1
175      177      63       7        4.0000000000e+00     0.0000000000e+00     100.00       0.1
208      210      84       12       4.0000000000e+00     0.0000000000e+00     100.00       0.1
245      247      103      4        4.0000000000e+00     0.0000000000e+00     100.00       0.1
278      279      124      5        4.0000000000e+00     0.0000000000e+00     100.00       0.1
309      310      147      14       4.0000000000e+00     0.0000000000e+00     100.00       0.1
347      345      165      10       4.0000000000e+00     3.3333333333e-01     91.67        0.1
A near optimal solution satisfying the absolute gap tolerance of 0.00e+00 has been located.

Objective of best integer solution : 4.000000000000e+00
Best objective bound               : 3.333333333333e-01
Construct solution objective       : Not employed
Construct solution # roundings     : 0
User objective cut value           : 0
Number of cuts generated           : 3
  Number of Gomory cuts            : 3
Number of branches                 : 347
Number of relaxations solved       : 345
Number of interior point iterations: 6
Number of simplex iterations       : 3460
Time spend presolving the root     : 0.00
Time spend in the heuristic        : 0.00
Time spend in the sub optimizers   : 0.00
  Time spend optimizing the root   : 0.00
Mixed integer optimizer terminated. Time: 0.12

Optimizer terminated. Time: 0.12

Integer solution solution summary
  Problem status  : PRIMAL_FEASIBLE
  Solution status : NEAR_INTEGER_OPTIMAL
  Primal.  obj: 4.0000000000e+00    nrm: 1e+01    Viol.  con: 1e+01    var: 1e+00    itg: 3e-16

Return code - 10004 [MSK_RES_TRM_MIO_NEAR_ABS_GAP]: The mixed-integer optimizer terminated because tl
```

The fields in the main MOSEK log output are:

Field	Description
BRANCHES	Current number of branches in tree.
RELAXS	Current number of nodes in branch and bound tree.
ACT_NDS	Current number of active nodes.
BEST_INT_OBJ	Current best integer solution (primal bound).
BEST_RELAX_OBJ	Current best relaxed solution (dual bound).
REL_GAP(%)	Relative gap between current BEST_INT_OBJ and BEST_RELAX_OBJ.
TIME	Current elapsed solving time in seconds.

The log then gives information about solving the model with discrete variables fixed in order to determine marginals. Option SOLVEFINAL can be used to disable this step. The fixed problem is solved as a

regular LP with warm start information. So the log looks identical to the MOSEK simplex optimizer for linear programs:

```
Solving fixed problem...
[...]
Optimizer started.
Simplex optimizer started.
Presolve started.
Eliminator - tries                  : 0                 time                   : 0.00
Lin. dep.  - tries                  : 0                 time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated. Time: 0.00
Simplex optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Basic solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 4.0000000000e+00    nrm: 1e+01    Viol.  con: 4e-16    var: 0e+00
  Dual.    obj: 4.0000000000e+00    nrm: 5e+00    Viol.  con: 0e+00    var: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.

MIP Solution:            4.000000   (3466 iterations, 347 nodes)
Final Solve:             4.000000   (0 iterations)

Best possible:           0.333333
Absolute gap:            3.666667
Relative gap:            0.916667
```

5.27.4 Semidefinite Programming with GAMS/MOSEK (experimental)

With Version 7, MOSEK introduced a semidefinite programming (SDP) solver into their portfolio. The following introduction is taken from the Mosek Documentation.

Semidefinite programming is a generalization of quadratic conic programming, allowing the use of matrix variables belonging to the convex cone of positive semidefinite matrices

$$\mathcal{S}_r^+ = \left\{ X \in \mathcal{S}_r : z^T X z \geq 0,\ \forall z \in \mathbb{R}^r \right\},$$

where $\mathcal{S}_r$ is the set of $r \times r$ real-valued symmetric matrices. MOSEK can solve semidefinite optimization problems of the form

$$\begin{array}{lccccll} \text{minimize} & & \displaystyle\sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \langle \overline{C}_j, \overline{X}_j \rangle + c^f & & & & \\ \text{subject to} & l_i^c & \leq & \displaystyle\sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \langle \overline{A}_{ij}, \overline{X}_j \rangle & \leq & u_i^c, & i = 0, \ldots, m-1, \\ & l_j^x & \leq & x_j & \leq & u_j^x, & j = 0, \ldots, n-1, \\ & & & x \in \mathcal{C}, \overline{X}_j \in \mathcal{S}_{r_j}^+, & & & j = 0, \ldots, p-1, \end{array} \tag{SDP}$$

where the problem has p symmetric positive semidefinite (PSD) variables $\overline{X}_j \in \mathcal{S}_{r_j}^+$ of dimension r_j with symmetric coefficient matrices $\overline{C}_j \in \mathcal{S}_{r_j}$ and $\overline{A}_{i,j} \in \mathcal{S}_{r_j}$. We use the standard notation for the matrix inner product, i.e., for $A, B \in \mathbb{R}^{m \times n}$ we have

$$\langle A, B \rangle := \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} A_{ij} B_{ij}.$$

5.27.4.1 Example

An example for a mixed semidefinite and conic quadratic programming problem with a 3-dimensional PSD matrix variable is the following:

$$
\begin{aligned}
\text{minimize} \quad & \left\langle \begin{bmatrix} 2 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \bar{X} \right\rangle + x_0 \\
\text{subject to} \quad & \left\langle \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \bar{X} \right\rangle + x_0 = 1 \\
& \left\langle \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \bar{X} \right\rangle + x_1 + x_2 = \frac{1}{2} \\
& x_0 \geq \sqrt{x_1^2 + x_2^2} \\
& \bar{X} \succeq 0
\end{aligned}
$$

The GAMS/MOSEK interface offers an experimental interface to MOSEK's SDP solver. It allows to state SDP's of the form (SDP) in GAMS language. For instance, the example problem from above can be formulated as follows (**sdp01** in the GAMS Test Library):

```
Set i / 0 * 2 /;
alias(i, ip);
Variables barX(i,i) PSDMATRIX
          x(i)      simple vars
          z         objective var
;
x.lo('0') = 0;
Parameters barAobj(i,i)  coefficients of barX in objective
           barAe1(i,i)   coefficients of barX in e1
           barAe2(i,i)   coefficients of barX in e2
;
Table barAobj(i,i)
   0    1    2
0  2.0  1.0  0.0
1  1.0  2.0  1.0
2  0.0  1.0  2.0
;
  identity matrix
barAe1(i,i) = 1.0;
  all-one matrix
barAe2(i,ip) = 1.0;
Equations obj, e1, e2, e3;
obj.. z   =e= sum((i,ip), barAobj(i,ip) * barX(i,ip)) + x('0');
e1..  1   =e= sum((i,ip), barAe1(i,ip)  * barX(i,ip)) + x('0');
e2..  0.5 =e= sum((i,ip), barAe2(i,ip)  * barX(i,ip)) + x('1') + x('2');
e3..  - sqr(x('0')) + sqr(x('1')) + sqr(x('2')) =l= 0;
Model m / all /;
```

We see that the matrix $\bar{X}$ is defined via the 2-dimensional GAMS variable `barX`. Additionally, the keyword `PSDMATRIX` at the beginning of the descriptive text (!) of the variable is used to indicate that the variables belonging to symbol `barX` are *to be interpreted* as a matrix variable with PSD constraint.

The Model shown above can be solved with MOSEK via the statements

```
option qcp = mosek;
Solve m minimizing z using QCP;
```

As GAMS has no native support for conic programming, the modeltype is specified as QCP.

The solve statement produces the following log output (see also Log Using the Interior Point Optimizer):

```
    M O S E K       version 7.0.0.75 (Build date: 2013-7-1 19:28:43)
    Copyright (C)   MOSEK ApS, Fruebjergvej 3, Box 16
                    DK-2100 Copenhagen, Denmark
                    http://www.mosek.com

Recognizing SDP variables for symbols barX (1)
Recognized 1 quadratic equation as conic constraint.
Optimizer started.
Conic interior-point optimizer started.
Presolve started.
Linear dependency checker started.
Linear dependency checker terminated.
Eliminator - tries                  : 0                 time                   : 0.00
Eliminator - elim's                 : 0
Lin. dep.  - tries                  : 1                 time                   : 0.00
Lin. dep.  - number                 : 0
Presolve terminated. Time: 0.00
Optimizer  - threads                : 1
Optimizer  - solved problem         : the primal
Optimizer  - Constraints            : 2
Optimizer  - Cones                  : 1
Optimizer  - Scalar variables       : 3                 conic                  : 3
Optimizer  - Semi-definite variables: 1                 scalarized             : 6
Factor     - setup time             : 0.00              dense det. time        : 0.00
Factor     - ML order time          : 0.00              GP order time          : 0.00
Factor     - nonzeros before factor : 3                 after factor           : 3
Factor     - dense dim.             : 0                 flops                  : 2.88e+02
ITE PFEAS    DFEAS    GFEAS    PRSTATUS   POBJ              DOBJ              MU       TIME
0   3.0e+00  1.0e+00  8.0e+00  0.00e+00   7.000000000e+00   0.000000000e+00   1.0e+00  0.00
1   4.3e-01  1.4e-01  1.2e+00  1.67e-01   1.601234178e+00   3.103213067e-01   1.4e-01  0.00
2   6.6e-02  2.2e-02  1.8e-01  1.21e+00   8.534948745e-01   6.725260078e-01   2.2e-02  0.00
3   4.9e-03  1.6e-03  1.3e-02  1.02e+00   7.158760069e-01   7.026959645e-01   1.6e-03  0.00
4   2.8e-04  9.4e-05  7.5e-04  1.00e+00   7.063055286e-01   7.055481573e-01   9.4e-05  0.00
5   1.6e-05  5.2e-06  4.1e-05  1.00e+00   7.057440915e-01   7.057021878e-01   5.2e-06  0.00
6   8.9e-07  3.0e-07  2.4e-06  1.00e+00   7.057124546e-01   7.057100494e-01   3.0e-07  0.00
7   1.6e-08  5.4e-09  4.3e-08  1.00e+00   7.057105296e-01   7.057104862e-01   5.4e-09  0.00
Interior-point optimizer terminated. Time: 0.00.

Optimizer terminated. Time: 0.00

Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 7.0571052965e-01  Viol.  con: 2e-08  var: 0e+00  barvar: 0e+00  cones: 0e+00
  Dual.    obj: 7.0571048621e-01  Viol.  con: 0e+00  var: 1e-16  barvar: 0e+00  cones: 0e+00

Return code - 0 [MSK_RES_OK]: No error occurred.
```

Finally, the optimal value for the matrix $\bar{X}$ and the dual values associated with bound constraints on entries of the matrix $\bar{X}$ can be displayed in GAMS as usual: `display X.l, X.m;`

```
----     53 VARIABLE barX.L  PSDMATRIX
            0           1           2
0       0.217      -0.260       0.217
1      -0.260       0.311      -0.260
2       0.217      -0.260       0.217
----     53 VARIABLE barX.M  PSDMATRIX
            0           1           2
0         EPS         EPS         EPS
1         EPS         EPS         EPS
2         EPS         EPS         EPS
```

5.27.4.2 Usage

The general syntax for defining a symmetric matrix or an (indexed) set of symmetric matrices with PSD constraints in GAMS is

```
Variable X(a,b,c,...,i1,..,ik,i1,...,ik)  "PSDMATRIX_k <explanatory text>";
```

The number k>0 specifies the number of indices that define the row/column dimension of the matrix. Specifying only the term `PSDMATRIX` is equivalent to `PSDMATRIX_1`. For a given k, the 2k-last indices are used to index the rows and columns of the matrix. The dimensions `a`, `b`, `c`, ... are optional and can be used to index a set of matrices. In the GAMS/MOSEK output, a number in parenthesis is used to indicate the number of PSD matrices that have been found for one symbol. For example, the code

```
Set i / a,b /;
Set j / s01 * s42 /;
Variables z, X(j,i,i) PSDMATRIX;
```

generates the output

```
Recognizing SDP variables for symbols X (42)
```

if all matrices `X(j,.,.)` also occur in a model instantiation.

Note

> The current syntax for declaring PSD matrix variables via the explanatory text a variable is a temporary solution that will hopefully be replaced by a GAMS language feature in the future.

Variables that were tagged as belonging to a PSD matrix can only occur in linear constraints. Within these constraints, the coefficient matrix for a PSD matrix needs to be specified in symmetric form. That is, if the parameter matrix `barAobj` in the example above is equivalently specified as

```
Table barAobj(i,i)
   0    1    2
0  2.0  0.0  0.0
1  2.0  2.0  0.0
2  0.0  2.0  2.0
;
```

the GAMS/MOSEK interface will quit with the error message

```
SDP coefficient matrix not symmetric: defObj_z: 2*barX(1,0) != 0*barX(0,1)
```

Bounds on entries in a PSD matrix variable can be specified as usual with `.lo` and `.up` attributes. These bound constraints are translated into linear constraints by the interface. If different bounds are given to symmetric entries of a PSD matrix variable ($\bar{X}_{i,j}$ vs. $\bar{X}_{j,i}$), the stronger bounds are used, which is equivalent to adding constraints for each of the matrix entries. For nonpositive lower bounds on diagonal entries, no extra constraints are added, as they are implied by the PSD constraint.

5.27.4.3 GAMS variables vs. PSD matrix entries

As GAMS is not aware that a indexed variable will be interpreted as PSD matrix variable, it may not generate variables for all matrix entries when instantiating a model. This is especially critical if the modeler specified bounds on matrix entries that do not appear in any of the model equations, as these bounds would not be visible to the GAMS/MOSEK interface.

As an example, consider the SDP relaxation $\max\{-\langle W, \bar{X}\rangle : X_{i,i} = 1 \; \forall i, X \succeq 0\}$ corresponding to the Goemans-Williamson Randomized Approximation Algorithm for MaxCut:

```
Parameter W(i,i)    edge weights;
Variable X(i,j)     PSDMATRIX
         sdpobjvar  objective var;
Equation sdpobj     objective function;
sdpobj.. sum((i,j), -W(i,j)*(X(i,j) + X(j,i))) =e= sdpobjvar;
X.fx(i,i) = 1.0;
model maxcutsdp / all /;
```

As `W(i,i)=0`, GAMS will not create any variables for `X(i,i)` when instantiating the model in a solve statement. As a consequence, the constraints `X(i,i)=1` will not be visible to GAMS/MOSEK. However, as MOSEK will compute values for the full matrix X, it will also compute values for the diagonal entries of `X`.

Further, the GAMS/MOSEK interface can pass primal solution values only for those entries of a PSD matrix variable that have a corresponding GAMS variable. This may make a solution matrix appear not to be PSD in GAMS, because not all entries have been passed back.

To be aware of such problems, the GAMS/MOSEK interface checks that it has a GAMS variable available for every entry of a PSD matrix variable. If not, it will report an error like

```
ERROR: Have 1600 GAMS variables for entries of 400 x 400 PSD Variable X(,), expected 160000 many.
```

This check can be disabled by setting the option SDPCHECKVARS to 0. Note, that the check is not able to alarm the user in situations where no GAMS variables were created for all entries of a row and corresponding column.

A simple workaround for this issue is to force all variables to be generated when a model is instantiated. This can be done by adding something like `eps*sum((i,j),X(i,j))` to one of the equations. Note, that `eps` is numerically equal to a 0.0 in GAMS, but has the effect that the term `sum((i,j),X(i,j))` is passed to the solver with 0.0-coefficient.

5.27.4.4 Dual Values for PSD constraints

The PSD constraint on a matrix variable $\bar{X}$ is associated with a dual PSD matrix variable $\bar{Y}$. As GAMS is not aware of the PSD constraints, it is also not aware of the corresponding dual variables. Thus, there is no native way to pass the duals for the PSD constraints back to GAMS. (Recall, that the marginals for a PSDMATRIX variable `X` are used to store the dual values associated with the bound constraints on matrix entries.)

To work around this issue, the GAMS/MOSEK interface offers the option SDPSOLUFILE. This option allows to specify the name of a GDX file that stores primal values for all entries of a matrix variable and dual values of the corresponding PSD constraint. For a variable `X(i,j)`, the GDX file stores the primal matrix value for all entries (i, j) (i.e., not just the ones for which GAMS variables were created, therefor offering another workaround for the issue discussed in the previous section) as level values of variable `X` and the dual matrix for the PSD constraint as marginal values.

5.27.4.5 Infeasible and Unbounded SDPs

The GAMS/Mosek link currently does not pass on certificates for primal or dual infeasibility from Mosek to GAMS if PSD variables are present.

5.27.5 Detailed Descriptions of MOSEK Options

FIXOPTFILE *(string)*: Name of option file which is read just before solving the fixed problem. ↩

MSK_DPAR_ANA_SOL_INFEAS_TOL *(real)*: If a constraint violates its bound with an amount larger than this value, the constraint name, index and violation will be printed by the solution analyzer. ↩

Default: `1e-06`

MSK_DPAR_BASIS_REL_TOL_S *(real)*: Maximum relative dual bound violation allowed in an optimal basic solution. ↩

Default: `1e-12`

MSK_DPAR_BASIS_TOL_S *(real)*: Maximum absolute dual bound violation in an optimal basic solution. ↩

Range: [`1e-09`, ∞]

Default: `1e-06`

MSK_DPAR_BASIS_TOL_X *(real)*: Maximum absolute primal bound violation allowed in an optimal basic solution. ↩

Range: [`1e-09`, ∞]

Default: `1e-06`

MSK_DPAR_CHECK_CONVEXITY_REL_TOL *(real)*: Not in use. ↩

Default: `1e-10`

MSK_DPAR_DATA_SYM_MAT_TOL *(real)*: Absolute zero tolerance for elements in in symmetric matrices. ↩

If any value in a symmetric matrix is smaller than this parameter in absolute terms MOSEK will treat the values as zero and generate a warning.

Range: [`1e-16`, `1e-06`]

Default: `1e-12`

MSK_DPAR_DATA_SYM_MAT_TOL_HUGE *(real)*: An element in a symmetric matrix which is larger than this value in absolute size causes an error. ↩

Default: `1e+20`

MSK_DPAR_DATA_SYM_MAT_TOL_LARGE *(real)*: An element in a symmetric matrix which is larger than this value in absolute size causes a warning message to be printed. ↩

Default: `1e+10`

MSK_DPAR_DATA_TOL_AIJ_HUGE *(real)*: An element in the constraint matrix which is larger than this value in absolute size causes an error. ↩

Default: `1e+20`

MSK_DPAR_DATA_TOL_AIJ_LARGE *(real)*: An element in the constraint matrix which is larger than this value in absolute size causes a warning message. ↩

Default: `1e+10`

MSK_DPAR_DATA_TOL_BOUND_INF *(real)*: Any bound which in absolute value is greater than this parameter is considered infinite. ↩

Default: `1e+16`

MSK_DPAR_DATA_TOL_BOUND_WRN *(real)*: If a bound value is larger than this value in absolute size, then a warning message is issued. ↩

Default: `1e+08`

MSK_DPAR_DATA_TOL_CJ_LARGE *(real)*: A coefficient in the objective function which is larger than this value in absolute terms causes a warning message. ↩

Default: `1e+08`

MSK_DPAR_DATA_TOL_C_HUGE *(real)*: A coefficient in the objective function which is larger than the value in absolute terms is considered to be huge and generates an error. ↩

Default: `1e+16`

MSK_DPAR_DATA_TOL_QIJ *(real)*: Absolute zero tolerance for coefficients of quadratic terms. ↩

Default: `1e-16`

MSK_DPAR_DATA_TOL_X *(real)*: Zero tolerance for constraints and variables i.e. if the distance between the lower and upper bound is less than this value, then the lower and upper bound is considered identical. ↩

Default: `1e-08`

MSK_DPAR_INTPNT_CO_TOL_DFEAS *(real)*: Dual feasibility tolerance used by the interior-point optimizer for conic problems. ↩

Range: [0, 1]

Default: `1e-08`

See also: MSK_DPAR_INTPNT_CO_TOL_NEAR_REL.

MSK_DPAR_INTPNT_CO_TOL_INFEAS *(real)*: Infeasibility tolerance used by the interior-point optimizer for conic problems. ↩

Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: [0, 1]

Default: `1e-12`

MSK_DPAR_INTPNT_CO_TOL_MU_RED *(real)*: Relative complementarity gap tolerance used by the interior-point optimizer for conic problems. ↩

Range: [0, 1]

Default: 1e-08

MSK_DPAR_INTPNT_CO_TOL_NEAR_REL *(real)*: Termination tolerance multiplier that is used if no accurate solution can be found. ↩

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Range: [1, ∞]

Default: 1

MSK_DPAR_INTPNT_CO_TOL_PFEAS *(real)*: Primal feasibility tolerance used by the interior-point optimizer for conic problems. ↩

Range: [0, 1]

Default: 1e-08

See also: MSK_DPAR_INTPNT_CO_TOL_NEAR_REL.

MSK_DPAR_INTPNT_CO_TOL_REL_GAP *(real)*: Relative gap termination tolerance used by the interior-point optimizer for conic problems. ↩

Range: [0, 1]

Default: 1e-08

See also: MSK_DPAR_INTPNT_CO_TOL_NEAR_REL.

MSK_DPAR_INTPNT_QO_TOL_DFEAS *(real)*: Dual feasibility tolerance used when the interior-point optimizer is applied to a quadratic optimization problem. ↩

Range: [0, 1]

Default: 1e-08

See also: MSK_DPAR_INTPNT_QO_TOL_NEAR_REL.

MSK_DPAR_INTPNT_QO_TOL_INFEAS *(real)*: Infeasibility tolerance used by the interior-point optimizer for quadratic problems. ↩

Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: [0, 1]

Default: 1e-12

MSK_DPAR_INTPNT_QO_TOL_MU_RED *(real)*: Relative complementarity gap tolerance used by the interior-point optimizer for quadratic problems. ↩

Range: [0, 1]

Default: 1e-08

MSK_DPAR_INTPNT_QO_TOL_NEAR_REL *(real)*: Termination tolerance multiplier that is used if no accurate solution can be found. ↩

If MOSEK cannot compute a solution that has the prescribed accuracy, then it will multiply the termination tolerances with value of this parameter. If the solution then satisfies the termination criteria, then the solution is denoted near optimal, near feasible and so forth.

Range: [1, ∞]

Default: 1

MSK_DPAR_INTPNT_QO_TOL_PFEAS *(real)*: Primal feasibility tolerance used by the interior-point optimizer for quadratic problems. ↩

Range: [0, 1]

Default: 1e-08

See also: MSK_DPAR_INTPNT_QO_TOL_NEAR_REL.

MSK_DPAR_INTPNT_QO_TOL_REL_GAP *(real)*: Relative gap termination tolerance used by the interior-point optimizer for quadratic problems. ↩

Range: [0, 1]

Default: 1e-08

See also: MSK_DPAR_INTPNT_QO_TOL_NEAR_REL.

MSK_DPAR_INTPNT_TOL_DFEAS *(real)*: Dual feasibility tolerance used by the interior-point optimizer for linear problems. ↩

Range: [0, 1]

Default: 1e-08

MSK_DPAR_INTPNT_TOL_DSAFE *(real)*: Controls the initial dual starting point used by the interior-point optimizer. ↩

If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it might be worthwhile to increase this value.

Range: [0.0001, ∞]

Default: 1

MSK_DPAR_INTPNT_TOL_INFEAS *(real)*: Infeasibility tolerance used by the interior-point optimizer for linear problems. ↩

Controls when the interior-point optimizer declares the model primal or dual infeasible. A small number means the optimizer gets more conservative about declaring the model infeasible.

Range: [0, 1]

Default: 1e-10

MSK_DPAR_INTPNT_TOL_MU_RED *(real)*: Relative complementarity gap tolerance used by the interior-point optimizer for linear problems. ↩

Range: [0, 1]

Default: 1e-16

MSK_DPAR_INTPNT_TOL_PATH *(real)*: Controls how close the interior-point optimizer follows the central path. ↩

A large value of this parameter means the central path is followed very closely. On numerically unstable problems it may be worthwhile to increase this parameter.

Range: [0, 0.9999]

Default: 1e-08

MSK_DPAR_INTPNT_TOL_PFEAS *(real)*: Primal feasibility tolerance used by the interior-point optimizer for linear problems. ↩

Range: [0, 1]

Default: 1e-08

MSK_DPAR_INTPNT_TOL_PSAFE *(real)*: Controls the initial primal starting point used by the interior-point optimizer. ↩

If the interior-point optimizer converges slowly and/or the constraint or variable bounds are very large, then it may be worthwhile to increase this value.

Range: [0.0001, ∞]

Default: 1

MSK_DPAR_INTPNT_TOL_REL_GAP *(real)*: Relative gap termination tolerance used by the interior-point optimizer for linear problems. ↩

Range: [1e-14, ∞]

Default: 1e-08

MSK_DPAR_INTPNT_TOL_REL_STEP *(real)*: Relative step size to the boundary for linear and quadratic optimization problems. ↩

Range: [0.0001, 0.999999]

Default: 0.9999

MSK_DPAR_INTPNT_TOL_STEP_SIZE *(real)*: Step size tolerance. ↩

If the step size falls below the value of this parameter, then the interior-point optimizer assumes it is stalled. In other words the interior-point optimizer does not make any progress and therefore it is better stop.

Range: [0, 1]

Default: 1e-06

MSK_DPAR_LOWER_OBJ_CUT *(real)*: Lower objective limit. ↩

If either a primal or dual feasible solution is found proving that the optimal objective value is outside, the interval (MSK_DPAR_LOWER_OBJ_CUT, MSK_DPAR_UPPER_OBJ_CUT), then MOSEK is terminated.

Range: [-∞, ∞]

Default: -1e+30

See also: MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH.

MSK_DPAR_LOWER_OBJ_CUT_FINITE_TRH *(real)*: Lower objective limit threshold. ↩

If the lower objective cut (MSK_DPAR_LOWER_OBJ_CUT) is less than this value, then it is treated as infinity.

Range: [-∞, ∞]

Default: -5e+29

MSK_DPAR_MIO_MAX_TIME *(real)*: This parameter limits the maximum time spent by the mixed-integer optimizer. ↩

A negative number means infinity.

Range: [-∞, ∞]

Default: -1

MSK_DPAR_MIO_REL_GAP_CONST *(real)*: This value is used to compute the relative gap for the solution to an integer optimization problem. ↩

Range: [1e-15, ∞]

Default: 1e-10

MSK_DPAR_MIO_TOL_ABS_GAP *(real)*: Absolute optimality tolerance employed by the mixed-integer optimizer. ↩

Default: GAMS OptCA

MSK_DPAR_MIO_TOL_ABS_RELAX_INT *(real)*: Absolute relaxation tolerance of the integer constraints. ↩

That means if the fractional part of a discrete variable ($\min(|x| - \lfloor x \rfloor, \lceil x \rceil - |x|)$) is less than the tolerance, then the integer restriction is assumed to be satisfied.

Range: [1e-09, ∞]

Default: 1e-05

MSK_DPAR_MIO_TOL_FEAS *(real)*: Feasibility tolerance for mixed integer solver. ↩

Range: [1e-09, 0.001]

Default: 1e-06

MSK_DPAR_MIO_TOL_REL_DUAL_BOUND_IMPROVEMENT *(real)*: If the relative improvement of the dual bound is smaller than this value, the solver will terminate the root cut generation. ↩

A value of 0.0 means that the value is selected automatically.

Range: [0, 1]

Default: 0

MSK_DPAR_MIO_TOL_REL_GAP *(real)*: Relative optimality tolerance employed by the mixed-integer optimizer. ↩

Default: GAMS OptCR

MSK_DPAR_OPTIMIZER_MAX_TIME *(real)*: Maximum amount of time the optimizer is allowed to spent on the optimization. ↩

A negative number means infinity.

Range: [-∞, ∞]

Default: GAMS ResLim

MSK_DPAR_PRESOLVE_TOL_ABS_LINDEP *(real)*: Absolute tolerance employed by the linear dependency checker. ↩

Default: 1e-06

MSK_DPAR_PRESOLVE_TOL_AIJ *(real)*: Absolute zero tolerance employed for constraint coefficients in presolve. ↩

Range: [1e-15, ∞]

Default: 1e-12

MSK_DPAR_PRESOLVE_TOL_PRIMAL_INFEAS_PERTURBATION *(real)*: The presolve is allowed to perturb a bound on a constraint or variable by this amount if it removes an infeasibility. ↩

Default: 1e-06

MSK_DPAR_PRESOLVE_TOL_REL_LINDEP *(real)*: Relative tolerance employed by the linear dependency checker. ↩

Default: 1e-10

MSK_DPAR_PRESOLVE_TOL_S *(real)*: Absolute zero tolerance employed for dual variables in presolve. ↩

Default: 1e-08

MSK_DPAR_PRESOLVE_TOL_X *(real)*: Absolute zero tolerance employed for primal variables in presolve. ↩

Default: 1e-08

MSK_DPAR_QCQO_REFORMULATE_REL_DROP_TOL *(real)*: This parameter determines when columns are dropped in incomplete Cholesky factorization during reformulation of quadratic problems. ↩

Default: 1e-15

MSK_DPAR_SEMIDEFINITE_TOL_APPROX *(real)*: Tolerance to define a matrix to be positive semidefinite. ↩

Range: [1e-15, ∞]

Default: 1e-10

MSK_DPAR_SIMPLEX_ABS_TOL_PIV *(real)*: Absolute pivot tolerance employed by the simplex optimizers. ↩

Range: [1e-12, ∞]

Default: 1e-07

MSK_DPAR_SIM_LU_TOL_REL_PIV *(real)*: Relative pivot tolerance for LU factorization in simplex optimizers and basis identification. ↩

A value closer to 1.0 generally improves numerical stability but typically also implies an increase in the computational work.

Range: [1e-06, 0.999999]

Default: 0.01

MSK_DPAR_UPPER_OBJ_CUT *(real)*: Upper objective limit. ↩

If either a primal or dual feasible solution is found proving that the optimal objective value is outside the interval (MSK_DPAR_LOWER_OBJ_CUT, MSK_DPAR_UPPER_OBJ_CUT), then MOSEK is terminated.

Range: [-∞, ∞]

Default: 1e+30

See also: MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH.

MSK_DPAR_UPPER_OBJ_CUT_FINITE_TRH *(real)*: Upper objective limit threshold. ↩

If the upper objective cut MSK_DPAR_UPPER_OBJ_CUT is greater than this value, then it is treated as infinity.

Range: [-∞, ∞]

Default: 5e+29

MSK_IPAR_AUTO_SORT_A_BEFORE_OPT *(string)*: Controls whether the elements in each column of the coefficient matrix are sorted before an optimization is performed. ↩

This is not required but makes the optimization insusceptible to reorderings of variables.

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_BI_CLEAN_OPTIMIZER *(string)*: Controls which simplex optimizer is used in the clean-up phase. ↩

Anything else than primal or dual simplex is equivalent to free simplex.

Default: `MSK_OPTIMIZER_FREE`

value	meaning
`MSK_OPTIMIZER_FREE`	The optimizer is chosen automatically.
`MSK_OPTIMIZER_PRIMAL_SIMPLEX`	The primal simplex optimizer is used.
`MSK_OPTIMIZER_DUAL_SIMPLEX`	The dual simplex optimizer is used.
`MSK_OPTIMIZER_FREE_SIMPLEX`	One of the simplex optimizers is used.

MSK_IPAR_BI_IGNORE_MAX_ITER *(string)*: Controls if basis identification is performed under certain conditions. ↩

If the parameter MSK_IPAR_INTPNT_BASIS has the value `MSK_BI_NO_ERROR` and the interior-point optimizer has terminated due to maximum number of iterations, then basis identification is performed if this parameter has the value `MSK_ON`.

Default: `MSK_OFF`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_BI_IGNORE_NUM_ERROR *(string)*: Turns on basis identification if interior-point optimizer is terminated due to a numerical problem. ↩

If the parameter `MSK_IPAR_INTPNT_BASIS` has the value `MSK_BI_NO_ERROR` and the interior-point optimizer has terminated due to a numerical problem, then basis identification is performed if this parameter has the value `MSK_ON`.

Default: `MSK_ON`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_BI_MAX_ITERATIONS *(integer)*: Controls the maximum number of simplex iterations allowed to optimize a basis after the basis identification. ↩

Default: `1000000`

MSK_IPAR_INFEAS_PREFER_PRIMAL *(string)*: If both certificates of primal and dual infeasibility are supplied then only the primal is used when this option is turned on. ↩

Default: `MSK_ON`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_INFEAS_REPORT_AUTO *(string)*: Controls whether an infeasibility report is auto-

matically produced after the optimization if the problem is primal or dual infeasible. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_INFEAS_REPORT_LEVEL *(integer)*: Controls the amount of information presented in an infeasibility report. ↩

Higher values imply more information.

Default: 1

MSK_IPAR_INTPNT_BASIS *(string)*: Controls whether the interior-point optimizer also computes an optimal basis. ↩

Default: MSK_BI_ALWAYS

value	meaning
MSK_BI_NEVER	Never do basis identification.
MSK_BI_ALWAYS	Basis identification is always performed even if the interior-point optimizer terminates abnormally.
MSK_BI_NO_ERROR	Basis identification is performed if the interior-point optimizer terminates without an error.
MSK_BI_IF_FEASIBLE	Basis identification is not performed if the interior-point optimizer terminates with a problem status saying that the problem is primal or dual infeasible.

See also: MSK_IPAR_BI_CLEAN_OPTIMIZER, MSK_IPAR_BI_IGNORE_MAX_ITER, MSK_IPAR_BI_IGNORE_NUM_ERROR, MSK_IPAR_BI_MAX_ITERATIONS.

MSK_IPAR_INTPNT_DIFF_STEP *(string)*: Controls whether different step sizes are allowed in the primal and dual space. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_INTPNT_MAX_ITERATIONS *(integer)*: Controls the maximum number of iterations allowed in the interior-point optimizer. ↩

Default: GAMS IterLim

MSK_IPAR_INTPNT_MAX_NUM_COR *(integer)*: Controls the maximum number of correctors allowed by the multiple corrector procedure. ↩

A negative value means that MOSEK is making the choice.

Range: {-1, ..., ∞}

Default: -1

MSK_IPAR_INTPNT_MAX_NUM_REFINEMENT_STEPS *(integer)*: Maximum number of steps to be used by the iterative refinement of the search direction. ↩

A negative value implies that the optimizer chooses the maximum number of iterative refinement steps.

Range: {-∞, ..., ∞}

Default: `-1`

MSK_IPAR_INTPNT_OFF_COL_TRH *(integer)*: Controls how aggressively offending columns are detected in the Jacobian of the constraint matrix. ↩

0 means no detection, 1 means aggressive detection, and higher values mean less aggressive detection.

Default: `40`

MSK_IPAR_INTPNT_ORDER_GP_NUM_SEEDS *(integer)*: The GP ordering is dependent on a random seed. ↩

Therefore, trying several random seeds may lead to a better ordering. This parameter controls the number of random seeds tried. A value of 0 means that MOSEK makes the choice.

Default: `0`

MSK_IPAR_INTPNT_ORDER_METHOD *(string)*: Controls the ordering strategy used by the interior-point optimizer when factorizing the Newton equation system. ↩

Default: `MSK_ORDER_METHOD_FREE`

value	meaning
`MSK_ORDER_METHOD_FREE`	The ordering method is chosen automatically.
`MSK_ORDER_METHOD_APPMINLOC`	Approximate minimum local fill-in ordering is employed.
`MSK_ORDER_METHOD_TRY_GRAPHPAR`	Always try the graph partitioning based ordering.
`MSK_ORDER_METHOD_FORCE_GRAPHPAR`	Always use the graph partitioning based ordering even if it is worse than the approximate minimum local fill ordering.
`MSK_ORDER_METHOD_NONE`	No ordering is used. Note using this value almost always leads to a significantly slow down.

MSK_IPAR_INTPNT_REGULARIZATION_USE *(string)*: Controls whether regularization is allowed. ↩

Default: `MSK_ON`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_INTPNT_SCALING *(string)*: Controls how the problem is scaled before the interior-point optimizer is used. ↩

Default: MSK_SCALING_FREE

value	meaning
MSK_SCALING_FREE	The optimizer chooses the scaling heuristic.
MSK_SCALING_NONE	No scaling is performed.

MSK_IPAR_INTPNT_SOLVE_FORM *(string)*: Controls whether the primal or the dual problem is solved. ↩

Default: MSK_SOLVE_FREE

value	meaning
MSK_SOLVE_FREE	The optimizer is free to solve either the primal or the dual problem.
MSK_SOLVE_PRIMAL	The optimizer should solve the primal problem.
MSK_SOLVE_DUAL	The optimizer should solve the dual problem.

MSK_IPAR_INTPNT_STARTING_POINT *(string)*: Starting point used by the interior-point optimizer. ↩

Default: MSK_STARTING_POINT_FREE

value	meaning
MSK_STARTING_POINT_FREE	The starting point is chosen automatically.
MSK_STARTING_POINT_GUESS	The optimizer guesses a starting point.
MSK_STARTING_POINT_CONSTANT	The optimizer constructs a starting point by assigning a constant value to all primal and dual variables. This starting point is normally robust.

MSK_IPAR_LOG *(integer)*: Controls the amount of log information. ↩

The value 0 implies that all log information is suppressed. A higher level implies that more information is logged.

Default: 10

MSK_IPAR_LOG_BI *(integer)*: Controls the amount of output printed by the basis identification procedure. ↩

A higher level implies that more information is logged.

Default: 1

MSK_IPAR_LOG_BI_FREQ *(integer)*: Controls logging frequency of the basis identification ↩

Default: 2500

MSK_IPAR_LOG_FEAS_REPAIR *(integer)*: Controls the amount of output printed when performing feasibility repair. ↩

Default: 1

MSK_IPAR_LOG_INFEAS_ANA *(integer)*: Controls amount of output printed by the infeasibility analyzer procedures. ↩

A higher level implies that more information is logged.

Default: `1`

MSK_IPAR_LOG_INTPNT *(integer)*: Controls amount of output printed by the interior-point optimizer. ↩

A higher level implies that more information is logged.

Default: `1`

MSK_IPAR_LOG_MIO *(integer)*: Controls the log level for the mixed-integer optimizer. ↩

A higher level implies that more information is logged.

Default: `4`

MSK_IPAR_LOG_MIO_FREQ *(integer)*: Controls how frequent the mixed-integer optimizer prints the log line. ↩

It will print a line every time `MSK_INTPAR_LOG_MIO_FREQ` relaxations have been solved.

Range: {-∞, ..., ∞}

Default: `10`

MSK_IPAR_LOG_ORDER *(integer)*: If turned on, then factor lines are added to the log. ↩

Default: `1`

MSK_IPAR_LOG_PRESOLVE *(integer)*: Controls amount of output printed by the presolve procedure. ↩

A higher level implies that more information is logged.

Default: `1`

MSK_IPAR_LOG_RESPONSE *(integer)*: Controls amount of output printed when response codes are reported. ↩

A higher level implies that more information is logged.

Default: `0`

MSK_IPAR_LOG_SIM *(integer)*: Controls amount of output printed by the simplex optimizer. ↩

A higher level implies that more information is logged.

Default: `4`

MSK_IPAR_LOG_SIM_FREQ *(integer)*: Controls simplex optimizer logging frequency. ↩

Default: `1000`

MSK_IPAR_LOG_STORAGE *(integer)*: When turned on, MOSEK prints messages regarding the storage usage and allocation. ↩

Default: `0`

MSK_IPAR_MIO_BRANCH_DIR *(string)*: Controls whether the mixed-integer optimizer is branching up or down by default. ↩

Default: `MSK_BRANCH_DIR_FREE`

value	meaning
MSK_BRANCH_DIR_FREE	The mixed-integer optimizer decides which branch to choose.
MSK_BRANCH_DIR_UP	The mixed-integer optimizer always chooses the up branch first.
MSK_BRANCH_DIR_DOWN	The mixed-integer optimizer always chooses the down branch first.
MSK_BRANCH_DIR_NEAR	Branch in direction nearest to selected fractional variable.
MSK_BRANCH_DIR_FAR	Branch in direction farthest from selected fractional variable.
MSK_BRANCH_DIR_ROOT_LP	Chose direction based on root lp value of selected variable.
MSK_BRANCH_DIR_GUIDED	Branch in direction of current incumbent.
MSK_BRANCH_DIR_PSEUDOCOST	Branch based on the pseudocost of the variable.

MSK_IPAR_MIO_CONIC_OUTER_APPROXIMATION *(string)*: If this option is turned on outer approximation is used when solving relaxations of conic problems; otherwise interior point is used. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CONSTRUCT_SOL *(string)*: Whether to construct an initial solution from starting point ↩

If enabled and all integer variables have been given a value for which a feasible mixed integer solution exists, then MOSEK generates an initial solution to the mixed integer problem by fixing all integer values and solving the remaining problem.

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_CLIQUE *(string)*: Controls whether clique cuts should be generated. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_CMIR *(string)*: Controls whether mixed integer rounding cuts should be generated. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.

value	meaning
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_GMI *(string)*: Controls whether GMI cuts should be generated. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_IMPLIED_BOUND *(string)*: Controls whether implied bound cuts should be generated. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_KNAPSACK_COVER *(string)*: Controls whether knapsack cover cuts should be generated. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_LIPRO *(string)*: Controls whether lift-and-project cuts should be generated. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_CUT_SELECTION_LEVEL *(integer)*: Controls how aggressively generated cuts are selected to be included in the relaxation. ↩

Default: -1

value	meaning
-1	The optimizer chooses the level of cut selection.
0	Generated cuts less likely to be added to the relaxation.
1	Cuts are more aggressively selected to be included in the relaxation.

MSK_IPAR_MIO_DATA_PERMUTATION_METHOD *(string)*: Controls what problem data permutation method is appplied to mixed-integer problems. ↩

Default: `MSK_MIO_DATA_PERMUTATION_METHOD_NONE`

value	meaning
`MSK_MIO_DATA_PERMUTATION_METHOD_NONE`	No problem data permutation is applied.
`MSK_MIO_DATA_PERMUTATION_METHOD_CYCLIC_SHIFT`	A random cyclic shift is applied to permute the problem data.
`MSK_MIO_DATA_PERMUTATION_METHOD_RANDOM`	A random permutation is applied to the problem data.

MSK_IPAR_MIO_DUAL_RAY_ANALYSIS_LEVEL *(string)*: Controls the amount of symmetry detection and handling employed by the mixed-integer optimizer in presolve. ↩

Default: `-1`

value	meaning
`-1`	The optimizer chooses the level of dual ray analysis employed.
`0`	Dual ray analysis is disabled.
`1`	A lower amount of dual ray analysis is employed.
`2`	A higher amount of dual ray analysis is employed.

MSK_IPAR_MIO_FEASPUMP_LEVEL *(string)*: Controls the way the Feasibility Pump heuristic is employed by the mixed-integer optimizer. ↩

Default: `-1`

value	meaning
`-1`	The optimizer chooses how the Feasibility Pump is used.
`0`	The Feasibility Pump is disabled.
`1`	The Feasibility Pump is enabled with an effort to improve solution quality.
`2`	The Feasibility Pump is enabled with an effort to reach feasibility early.

MSK_IPAR_MIO_HEURISTIC_LEVEL *(integer)*: Controls the heuristic employed by the mixed-integer optimizer to locate an initial good integer feasible solution. ↩

A value of zero means the heuristic is not used at all. A larger value than 0 means that a gradually more sophisticated heuristic is used which is computationally more expensive. A negative value implies that the optimizer chooses the heuristic. Normally a value around 3 to 5 should be optimal.

Range: {-∞, ..., ∞}

Default: `-1`

MSK_IPAR_MIO_MAX_NUM_BRANCHES *(integer)*: Maximum number of branches allowed during the branch and bound search. ↩

A negative value means infinite.

Range: {-∞, ..., ∞}

Default: `-1`

MSK_IPAR_MIO_MAX_NUM_RELAXS *(integer)*: Maximum number of relaxations allowed during the branch and bound search. ↩

A negative value means infinite.

Range: {-∞, ..., ∞}

Default: `GAMS NodLim`

MSK_IPAR_MIO_MAX_NUM_RESTARTS *(integer)*: Maximum number of restarts allowed during the branch and bound search. ↩

Default: `0`

MSK_IPAR_MIO_MAX_NUM_ROOT_CUT_ROUNDS *(integer)*: Maximum number of cut separation rounds at the root node. ↩

Default: `100`

MSK_IPAR_MIO_MAX_NUM_SOLUTIONS *(integer)*: The mixed-integer optimizer can be terminated after a certain number of different feasible solutions has been located. ↩

If this parameter has the value $n > 0$, then the mixed-integer optimizer will be terminated when n feasible solutions have been located.

Range: {-∞, ..., ∞}

Default: `-1`

MSK_IPAR_MIO_MEMORY_EMPHASIS_LEVEL *(string)*: Controls how much emphasis is put on reducing memory usage. ↩

Being more conservative about memory usage may come at the cost of decreased solution speed.

Default: `0`

value	meaning
`0`	The optimizer chooses.
`1`	More emphasis is put on reducing memory usage and less on speed.

MSK_IPAR_MIO_MIN_REL *(integer)*: Number of times a variable must have been branched on for its pseudocost to be considered reliable. ↩

Default: `5`

MSK_IPAR_MIO_NODE_OPTIMIZER *(string)*: Controls which optimizer is employed at the non-root nodes in the mixed-integer optimizer. ↩

Default: `MSK_OPTIMIZER_FREE`

value	meaning
`MSK_OPTIMIZER_FREE`	The optimizer is chosen automatically.
`MSK_OPTIMIZER_INTPNT`	The interior-point optimizer is used.
`MSK_OPTIMIZER_CONIC`	The optimizer for problems having conic constraints.

value	meaning
MSK_OPTIMIZER_PRIMAL_SIMPLEX	The primal simplex optimizer is used.
MSK_OPTIMIZER_DUAL_SIMPLEX	The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE_SIMPLEX	One of the simplex optimizers is used.
MSK_OPTIMIZER_MIXED_INT	The mixed-integer optimizer.

MSK_IPAR_MIO_NODE_SELECTION *(string)*: Controls the node selection strategy employed by the mixed-integer optimizer. ↩

Default: MSK_MIO_NODE_SELECTION_FREE

value	meaning
MSK_MIO_NODE_SELECTION_FREE	The optimizer decides the node selection strategy.
MSK_MIO_NODE_SELECTION_FIRST	The optimizer employs a depth first node selection strategy.
MSK_MIO_NODE_SELECTION_BEST	The optimizer employs a best bound node selection strategy.
MSK_MIO_NODE_SELECTION_PSEUDO	The optimizer employs selects the node based on a pseudo cost estimate.

MSK_IPAR_MIO_NUMERICAL_EMPHASIS_LEVEL *(string)*: Controls how much emphasis is put on reducing numerical problems possibly at the expense of solution speed. ↩

Default: 0

value	meaning
0	The optimizer chooses.
1	More emphasis is put on reducing numerical problems.
2	Even more emphasis.

MSK_IPAR_MIO_PERSPECTIVE_REFORMULATE *(string)*: Enables or disables perspective reformulation in presolve. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_PRESOLVE_AGGREGATOR_USE *(string)*: Controls if the aggregator should be used. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_MIO_PROBING_LEVEL *(integer)*: Controls the amount of probing employed by the

mixed-integer optimizer in presolve. ↩

Default: `-1`

value	meaning
`-1`	The optimizer chooses the level of probing employed
`0`	Probing is disabled
`1`	A low amount of probing is employed
`2`	A medium amount of probing is employed
`3`	A high amount of probing is employed

MSK_IPAR_MIO_PROPAGATE_OBJECTIVE_CONSTRAINT *(string)*: Use objective domain propagation. ↩

Default: `MSK_OFF`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_MIO_QCQO_REFORMULATION_METHOD *(string)*: Controls what reformulation method is applied to mixed-integer quadratic problems. ↩

Default: `MSK_MIO_QCQO_REFORMULATION_METHOD_FREE`

value	meaning
`MSK_MIO_QCQO_REFORMULATION_METHOD_FREE`	The mixed-integer optimizer decides which reformulation method to apply.
`MSK_MIO_QCQO_REFORMULATION_METHOD_NONE`	No reformulation method is applied.
`MSK_MIO_QCQO_REFORMULATION_METHOD_LINEARIZATION`	Reformulation via linearization is applied.
`MSK_MIO_QCQO_REFORMULATION_METHOD_EIGEN_VAL_METHOD`	The eigenvalue method is applied.
`MSK_MIO_QCQO_REFORMULATION_METHOD_DIAG_SDP`	SDP perturbation of matrix diagonals via the solution of SDPs is applied.
`MSK_MIO_QCQO_REFORMULATION_METHOD_RELAX_SDP`	Reformulation based on the solution of an SDP-relaxation of the problem is applied.

MSK_IPAR_MIO_RINS_MAX_NODES *(integer)*: Controls the maximum number of nodes allowed in each call to the RINS heuristic. ↩

The default value of -1 means that the value is determined automatically. A value of zero turns off the heuristic.

Range: {`-1`, ..., ∞}

Default: `-1`

MSK_IPAR_MIO_ROOT_OPTIMIZER *(string)*: Controls which optimizer is employed at the root node in the mixed-integer optimizer. ↩

Default: `MSK_OPTIMIZER_FREE`

value	meaning
MSK_OPTIMIZER_FREE	The optimizer is chosen automatically.
MSK_OPTIMIZER_INTPNT	The interior-point optimizer is used.
MSK_OPTIMIZER_CONIC	The optimizer for problems having conic constraints.
MSK_OPTIMIZER_PRIMAL_SIMPLEX	The primal simplex optimizer is used.
MSK_OPTIMIZER_DUAL_SIMPLEX	The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE_SIMPLEX	One of the simplex optimizers is used.

MSK_IPAR_MIO_ROOT_REPEAT_PRESOLVE_LEVEL *(integer)*: Controls whether presolve can be repeated at root node. ↩

Default: -1

value	meaning
-1	The optimizer chooses whether presolve is repeated.
0	Never repeat presolve.
1	Always repeat presolve.

MSK_IPAR_MIO_SEED *(integer)*: Sets the random seed used for randomization in the mixed integer optimizer. ↩

Selecting a different seed can change the path the optimizer takes to the optimal solution.

Default: 42

MSK_IPAR_MIO_SYMMETRY_LEVEL *(string)*: Controls the amount of symmetry detection and handling employed by the mixed-integer optimizer in presolve. ↩

Default: -1

value	meaning
-1	The optimizer chooses the level of symmetry detection and handling employed.
0	Symmetry detection and handling is disabled.
1	A low amount of symmetry detection and handling is employed.
2	A medium amount of symmetry detection and handling is employed.
3	A high amount of symmetry detection and handling is employed .
4	An extremely high amount of symmetry detection and handling is employed.

MSK_IPAR_MIO_VAR_SELECTION *(string)*: Controls the variable selection strategy employed by the mixed-integer optimizer. ↩

Default: MSK_MIO_VAR_SELECTION_FREE

value	meaning
MSK_MIO_VAR_SELECTION_FREE	The optimizer decides the variable selection strategy.
MSK_MIO_VAR_SELECTION_PSEUDOCOST	The optimizer employs pseudocost variable selection.
MSK_MIO_VAR_SELECTION_STRONG	The optimizer employs strong branching variable selection.

MSK_IPAR_MIO_VB_DETECTION_LEVEL *(integer)*: Controls how much effort is put into detecting variable bounds. ↩

Default: `-1`

value	meaning
`-1`	The optimizer chooses.
`0`	No variable bounds are detected.
`1`	Only detect variable bounds that are directly represented in the problem.
`2`	Detect variable bounds in probing.

MSK_IPAR_NUM_THREADS *(integer)*: Controls the number of threads employed by the optimizer. ↩

If set to 0 the number of threads used will be equal to the number of cores detected on the machine.

Default: `GAMS Threads`

MSK_IPAR_OPF_WRITE_HEADER *(string)*: Write a text header with date and MOSEK version in an OPF file. ↩

Default: `MSK_ON`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_OPF_WRITE_HINTS *(string)*: Write a hint section with problem dimensions in the beginning of an OPF file. ↩

Default: `MSK_ON`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_OPF_WRITE_LINE_LENGTH *(integer)*: Aim to keep lines in OPF files not much longer than this. ↩

Default: `80`

MSK_IPAR_OPF_WRITE_PARAMETERS *(string)*: Write a parameter section in an OPF file. ↩

Default: `MSK_OFF`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_OPF_WRITE_PROBLEM *(string)*: Write objective, constraints, bounds etc. ↩

to an OPF file.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_SOLUTIONS *(string)*: Enable inclusion of solutions in the OPF files. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_SOL_BAS *(string)*: Whether to include basic solution in OPF files. ↩

If MSK_IPAR_OPF_WRITE_SOLUTIONS is MSK_ON and a basic solution is defined, include the basic solution in OPF files.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_SOL_ITG *(string)*: Whether to include integer solution in OPF files. ↩

If MSK_IPAR_OPF_WRITE_SOLUTIONS is MSK_ON and an integer solution is defined, write the integer solution to OPF files.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPF_WRITE_SOL_ITR *(string)*: Whether to include interior solution in OPF files. ↩

If MSK_IPAR_OPF_WRITE_SOLUTIONS is MSK_ON and an interior solution is defined, write the interior solution to OPF files.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_OPTIMIZER *(string)*: The parameter controls which optimizer is used to optimize the

task. ↩

Default: MSK_OPTIMIZER_FREE

value	meaning
MSK_OPTIMIZER_FREE	The optimizer is chosen automatically.
MSK_OPTIMIZER_INTPNT	The interior-point optimizer is used.
MSK_OPTIMIZER_CONIC	The optimizer for problems having conic constraints.
MSK_OPTIMIZER_PRIMAL_SIMPLEX	The primal simplex optimizer is used.
MSK_OPTIMIZER_DUAL_SIMPLEX	The dual simplex optimizer is used.
MSK_OPTIMIZER_FREE_SIMPLEX	One of the simplex optimizers is used.
MSK_OPTIMIZER_MIXED_INT	The mixed-integer optimizer.

MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_FILL *(integer)*: Controls the maximum amount of fill-in that can be created by one pivot in the elimination phase of presolve. ↩

A negative value means the parameter value is selected automatically.

Range: {-∞, ..., ∞}

Default: -1

MSK_IPAR_PRESOLVE_ELIMINATOR_MAX_NUM_TRIES *(integer)*: Control the maximum number of times the eliminator is tried. ↩

A negative value implies MOSEK decides.

Range: {-∞, ..., ∞}

Default: -1

MSK_IPAR_PRESOLVE_LINDEP_ABS_WORK_TRH *(integer)*: Controls the linear dependency check, which is potentially computationally expensive. ↩

Range: {-∞, ..., ∞}

Default: 100

MSK_IPAR_PRESOLVE_LINDEP_NEW *(string)*: Controls whether a new experimental linear dependency checker is employed. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_PRESOLVE_LINDEP_REL_WORK_TRH *(integer)*: Controls the linear dependency check, which is potentially computationally expensive. ↩

Range: {-∞, ..., ∞}

Default: 100

MSK_IPAR_PRESOLVE_LINDEP_USE *(string)*: Controls whether the linear constraints are checked for linear dependencies. ↩

Default: `MSK_ON`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_PRESOLVE_MAX_NUM_PASS *(integer)*: Control the maximum number of times presolve passes over the problem. ↩

A negative value implies MOSEK decides.

Range: {-∞, ..., ∞}

Default: `-1`

MSK_IPAR_PRESOLVE_MAX_NUM_REDUCTIONS *(integer)*: Controls the maximum number of reductions performed by the presolve. ↩

The value of the parameter is normally only changed in connection with debugging. A negative value implies that an infinite number of reductions are allowed.

Range: {-∞, ..., ∞}

Default: `-1`

MSK_IPAR_PRESOLVE_USE *(string)*: Controls whether the presolve is applied to a problem before it is optimized. ↩

Default: `MSK_PRESOLVE_MODE_FREE`

value	meaning
`MSK_PRESOLVE_MODE_OFF`	The problem is not presolved before it is optimized.
`MSK_PRESOLVE_MODE_ON`	The problem is presolved before it is optimized.
`MSK_PRESOLVE_MODE_FREE`	It is decided automatically whether to presolve before the problem is optimized.

MSK_IPAR_PTF_WRITE_PARAMETERS *(string)*: If enabled, then the parameters section is written. ↩

Default: `MSK_OFF`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_PTF_WRITE_SOLUTIONS *(string)*: If enabled, then the solution section is written if any solutions are available, otherwise solution section is not written even if solutions are available. ↩

Default: `MSK_OFF`

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_PTF_WRITE_TRANSFORM *(string)*: If enabled, then constraint blocks with identifiable conic slacks are transformed into conic constraints and the slacks are eliminated. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_SIM_BASIS_FACTOR_USE *(string)*: Controls whether an LU factorization of the basis is used in a hot-start. ↩

Forcing a refactorization sometimes improves the stability of the simplex optimizers, but in most cases there is a performance penalty.

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_SIM_DEGEN *(string)*: Controls how aggressively degeneration is handled. ↩

Default: MSK_SIM_DEGEN_FREE

value	meaning
MSK_SIM_DEGEN_NONE	The simplex optimizer should use no degeneration strategy.
MSK_SIM_DEGEN_FREE	The simplex optimizer chooses the degeneration strategy.
MSK_SIM_DEGEN_AGGRESSIVE	The simplex optimizer should use an aggressive degeneration strategy.
MSK_SIM_DEGEN_MODERATE	The simplex optimizer should use a moderate degeneration strategy.
MSK_SIM_DEGEN_MINIMUM	The simplex optimizer should use a minimum degeneration strategy.

MSK_IPAR_SIM_DUAL_CRASH *(integer)*: Controls whether crashing is performed in the dual simplex optimizer. ↩

In general if a basis consists of more than (100*MSK_IPAR_SIM_DUAL_CRASH) percent fixed variables, then a crash will be performed.

Default: 90

MSK_IPAR_SIM_DUAL_RESTRICT_SELECTION *(integer)*: Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the dual simplex. ↩

The dual simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the dual simplex optimizer first choose a subset of all the potential outgoing variables. Next, for some time it will choose the outgoing variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Range: {0, ..., 100}

Default: 50

MSK_IPAR_SIM_DUAL_SELECTION *(string)*: Controls the choice of the incoming variable, known as the selection strategy, in the dual simplex optimizer. ↩

Default: MSK_SIM_SELECTION_FREE

value	meaning
MSK_SIM_SELECTION_FREE	The optimizer chooses the pricing strategy.
MSK_SIM_SELECTION_FULL	The optimizer uses full pricing.
MSK_SIM_SELECTION_ASE	The optimizer uses approximate steepest-edge pricing.
MSK_SIM_SELECTION_DEVEX	The optimizer uses devex steepest-edge pricing. If it is not available an approximate steep-edge selection is chosen.
MSK_SIM_SELECTION_SE	The optimizer uses steepest-edge selection. If it is not available an approximate steep-edge selection is chosen.
MSK_SIM_SELECTION_PARTIAL	The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

MSK_IPAR_SIM_EXPLOIT_DUPVEC *(string)*: Controls if the simplex optimizers are allowed to exploit duplicated columns. ↩

Default: MSK_SIM_EXPLOIT_DUPVEC_OFF

value	meaning
MSK_SIM_EXPLOIT_DUPVEC_ON	Allow the simplex optimizer to exploit duplicated columns.
MSK_SIM_EXPLOIT_DUPVEC_OFF	Disallow the simplex optimizer to exploit duplicated columns.
MSK_SIM_EXPLOIT_DUPVEC_FREE	The simplex optimizer can choose freely.

MSK_IPAR_SIM_HOTSTART *(string)*: Controls the type of hot-start that the simplex optimizer perform. ↩

Default: MSK_SIM_HOTSTART_FREE

value	meaning
MSK_SIM_HOTSTART_NONE	The simplex optimizer performs a coldstart.
MSK_SIM_HOTSTART_FREE	The simplex optimize chooses the hot-start type.
MSK_SIM_HOTSTART_STATUS_KEYS	Only the status keys of the constraints and variables are used to choose the type of hot-start.

MSK_IPAR_SIM_HOTSTART_LU *(string)*: Determines if the simplex optimizer should exploit the initial factorization. ↩

Default: `MSK_ON`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_SIM_MAX_ITERATIONS *(integer)*: Maximum number of iterations that can be used by a simplex optimizer. ↩

Default: `GAMS IterLim`

MSK_IPAR_SIM_MAX_NUM_SETBACKS *(integer)*: Controls how many set-backs are allowed within a simplex optimizer. ↩

A set-back is an event where the optimizer moves in the wrong direction. This is impossible in theory but may happen due to numerical problems.

Default: `250`

MSK_IPAR_SIM_NON_SINGULAR *(string)*: Controls if the simplex optimizer ensures a non-singular basis, if possible. ↩

Default: `MSK_ON`

value	meaning
`MSK_ON`	Switch the option on.
`MSK_OFF`	Switch the option off.

MSK_IPAR_SIM_PRIMAL_CRASH *(integer)*: Controls whether crashing is performed in the primal simplex optimizer. ↩

In general if a basis consists of more than `(100*MSK_IPAR_SIM_PRIMAL_CRASH)` percent fixed variables, then a crash will be performed.

Default: `90`

MSK_IPAR_SIM_PRIMAL_RESTRICT_SELECTION *(integer)*: Controls how aggressively a restricted selection/pricing strategy is used to choose the outgoing variable in the primal simplex. ↩

The primal simplex optimizer can use a so-called restricted selection/pricing strategy to chooses the outgoing variable. Hence, if restricted selection is applied, then the primal simplex optimizer first choose a subset of all the potential incoming variables. Next, for some time it will choose the incoming variable only among the subset. From time to time the subset is redefined. A larger value of this parameter implies that the optimizer will be more aggressive in its restriction strategy, i.e. a value of 0 implies that the restriction strategy is not applied at all.

Range: {`0`, ..., `100`}

Default: `50`

MSK_IPAR_SIM_PRIMAL_SELECTION *(string)*: Controls the choice of the incoming variable, known as the selection strategy, in the primal simplex optimizer. ↩

Default: `MSK_SIM_SELECTION_FREE`

value	meaning
MSK_SIM_SELECTION_FREE	The optimizer chooses the pricing strategy.
MSK_SIM_SELECTION_FULL	The optimizer uses full pricing.
MSK_SIM_SELECTION_ASE	The optimizer uses approximate steepest-edge pricing.
MSK_SIM_SELECTION_DEVEX	The optimizer uses devex steepest-edge pricing. If it is not available an approximate steep-edge selection is chosen.
MSK_SIM_SELECTION_SE	The optimizer uses steepest-edge selection. If it is not available an approximate steep-edge selection is chosen.
MSK_SIM_SELECTION_PARTIAL	The optimizer uses a partial selection approach. The approach is usually beneficial if the number of variables is much larger than the number of constraints.

MSK_IPAR_SIM_REFORMULATION *(string)*: Controls if the simplex optimizers are allowed to reformulate the problem. ↩

Default: MSK_SIM_REFORMULATION_OFF

value	meaning
MSK_SIM_REFORMULATION_ON	Allow the simplex optimizer to reformulate the problem.
MSK_SIM_REFORMULATION_OFF	Disallow the simplex optimizer to reformulate the problem.
MSK_SIM_REFORMULATION_FREE	The simplex optimizer can choose freely.
MSK_SIM_REFORMULATION_AGGRESSIVE	The simplex optimizer should use an aggressive reformulation strategy.

MSK_IPAR_SIM_SAVE_LU *(string)*: Controls if the LU factorization stored should be replaced with the LU factorization corresponding to the initial basis. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_SIM_SCALING *(string)*: Controls how much effort is used in scaling the problem before a simplex optimizer is used. ↩

Default: MSK_SCALING_FREE

value	meaning
MSK_SCALING_FREE	The optimizer chooses the scaling heuristic.
MSK_SCALING_NONE	No scaling is performed.

MSK_IPAR_SIM_SCALING_METHOD *(string)*: Controls how the problem is scaled before a simplex optimizer is used. ↩

Default: MSK_SCALING_METHOD_POW2

value	meaning
MSK_SCALING_METHOD_POW2	Scales only with power of 2 leaving the mantissa untouched.
MSK_SCALING_METHOD_FREE	The optimizer chooses the scaling heuristic.

MSK_IPAR_SIM_SEED *(integer)*: Sets the random seed used for randomization in the simplex optimizers. ↩

Range: {0, ..., 32749}

Default: 23456

MSK_IPAR_SIM_SOLVE_FORM *(string)*: Controls whether the primal or the dual problem is solved by the primal-/dual-simplex optimizer. ↩

Default: MSK_SOLVE_FREE

value	meaning
MSK_SOLVE_FREE	The optimizer is free to solve either the primal or the dual problem.
MSK_SOLVE_PRIMAL	The optimizer should solve the primal problem.
MSK_SOLVE_DUAL	The optimizer should solve the dual problem.

MSK_IPAR_SIM_STABILITY_PRIORITY *(integer)*: Controls how high priority the numerical stability should be given. ↩

Range: {0, ..., 100}

Default: 50

MSK_IPAR_SIM_SWITCH_OPTIMIZER *(string)*: Controls the simplex behavior. ↩

The simplex optimizer sometimes chooses to solve the dual problem instead of the primal problem. This implies that if you have chosen to use the dual simplex optimizer and the problem is dualized, then it actually makes sense to use the primal simplex optimizer instead. If this parameter is on and the problem is dualized and furthermore the simplex optimizer is chosen to be the primal (dual) one, then it is switched to the dual (primal).

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_TIMING_LEVEL *(integer)*: Controls the amount of timing performed inside MOSEK. ↩

Default: 1

MSK_IPAR_WRITE_COMPRESSION *(integer)*: Controls whether the data file is compressed while it is written. ↩

0 means no compression while higher values mean more compression.

Default: 9

MSK_IPAR_WRITE_DATA_PARAM *(string)*: If this option is turned on the parameter settings are written to the data file as parameters. ↩

Default: `MSK_OFF`

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_GENERIC_NAMES *(string)*: Controls whether generic names should be used instead of user-defined names when writing to the data file. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_GENERIC_NAMES_IO *(integer)*: Index origin used in generic names. ↩

Default: 1

MSK_IPAR_WRITE_IGNORE_INCOMPATIBLE_ITEMS *(string)*: Controls if the writer ignores incompatible problem items when writing files. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_JSON_INDENTATION *(string)*: When set, the JSON task and solution files are written with indentation for better readability. ↩

Default: MSK_OFF

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_LP_FULL_OBJ *(string)*: Write all variables, including the ones with 0-coefficients, in the objective. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_LP_LINE_WIDTH *(integer)*: Maximum width of line in an LP file written by MOSEK. ↩

Range: {40, ..., ∞}

Default: 80

MSK_IPAR_WRITE_MPS_FORMAT *(string)*: Controls in which format the MPS is written. ↩

Default: MSK_MPS_FORMAT_FREE

value	meaning
MSK_MPS_FORMAT_STRICT	It is assumed that the input file satisfies the MPS format strictly.
MSK_MPS_FORMAT_RELAXED	It is assumed that the input file satisfies a slightly relaxed version of the MPS format.
MSK_MPS_FORMAT_FREE	It is assumed that the input file satisfies the free MPS format. This implies that spaces are not allowed in names. Otherwise the format is free.
MSK_MPS_FORMAT_CPLEX	The CPLEX compatible version of the MPS format is employed.

MSK_IPAR_WRITE_MPS_INT *(string)*: Controls if marker records are written to the MPS file to indicate whether variables are integer restricted. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_TASK_INC_SOL *(string)*: Controls whether the solutions are stored in the task file too. ↩

Default: MSK_ON

value	meaning
MSK_ON	Switch the option on.
MSK_OFF	Switch the option off.

MSK_IPAR_WRITE_XML_MODE *(string)*: Controls if linear coefficients should be written by row or column when writing in the XML file format. ↩

Default: MSK_WRITE_XML_MODE_ROW

value	meaning
MSK_WRITE_XML_MODE_ROW	Write in row order.
MSK_WRITE_XML_MODE_COL	Write in column order.

MSK_SPAR_DATA_FILE_NAME *(string)*: If set, problem data is written to this file. File extension determines format. ↩

Synonym: writeprob

MSK_SPAR_PARAM_READ_FILE_NAME *(string)*: Modifications to the parameter database is read from this file. ↩

MSK_SPAR_PARAM_WRITE_FILE_NAME *(string)*: The parameter database is written to this file. ↩

MSK_SPAR_WRITE_LP_GEN_VAR_NAME *(string)*: Sometimes when an LP file is written additional variables must be inserted. ↩

They will have the prefix denoted by this parameter.

Default: `"xmskgen"`

QEXTRACTALG *(integer)*: Switch to choose extraction algorithm for quadratic equations in GAMS interface. ↩

Default: `0`

value	meaning
`0`	Choose automatically.
`1`	ThreePass: uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage
`2`	DoubleForward: uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.
`3`	Concurrent: Uses ThreePass and DoubleForward in parallel. As soon as one finishes, the other one stops.

SDPCHECKVARS *(boolean)*: Switch to disable checking that for every entry of a PSD matrix variable also a corresponding GAMS variable is present. ↩

Default: `1`

SDPSOLUFILE *(string)*: Name of GDX file to write primal solution of all PSD matrix variables and dual solution for PSD constraints. ↩

SOLVEFINAL *(boolean)*: Switch to resolve the problem with fixed discrete variables after the MOSEK optimizer finished. ↩

Default: `1`

value	meaning
`0`	Do not solve the fixed problem.
`1`	Solve the fixed problem and return duals.

5.27.6 Setting up a GAMS/MOSEK-Link license

To use the GAMS/MOSEK solver with a GAMS/MOSEK-Link license you have save a MOSEK standalone license and point to the MOSEK license file via environment variable `MOSEKLM_LICENSE_FILE`.

5.28 NLPEC

5.28.1 Introduction

The GAMS/NLPEC solver, developed jointly by Michael Ferris of UW-Madison and GAMS Development, solves MPEC and MCP models via reformulation of the complementarity constraints. The resulting sequence of NLP models are parameterized by a scalar μ and solved by existing NLP solvers. The resulting solutions are used to recover an MPEC or MCP solution.

GAMS/NLPEC serves a number of purposes. In many cases, it is an effective tool for solving MPEC models, the first such tool available within GAMS. It also serves as a way to experiment with the many reformulation strategies proposed for solving MPEC and MCP models. Without something like NLPEC (and a library of models to test with) a comprehensive and thorough test and comparison of the various reformulation strategies would not be possible. To better serve these purposes, NLPEC has an open architecture. The model reformulations are written out as GAMS source for solution via an NLP solver, so it is possible to view this source and modify it if desired.

A brief note about notation is in order. The GAMS keyword `positive` is used to indicate nonnegative variables. The same holds for nonpositive variables and the GAMS keyword `negative`.

5.28.2 Usage

GAMS/NLPEC can solve models of two types: MPEC and MCP. If you did not specify NLPEC as the default MPEC or MCP solver, use the following statement in your GAMS model before the solve statement:

```
option MPEC=nlpec; { or MCP }
```

You can also make NLPEC the default solver via the command line:

```
gams nash  MPEC=nlpec MCP=nlpec
```

You can use NLPEC with its default strategy and formulation, but most users will want to use an options file (Section Options) after reading about the different types of reformulations possible (Section Reformulation). In addition, an understanding of the architecture of NLPEC (Section Open Architecture) will be helpful in understanding how GAMS options are treated. Although NLPEC doesn't use the GAMS options `workspace`, `workfactor`, `optcr`, `optca`, `reslim`, `iterlim`, and `domlim` directly, it passes these options on in the reformulated model so they are available to the NLP subsolver.

5.28.3 Reformulation

In this section we describe the different ways that the NLPEC solver can reformulate an MPEC as an NLP. The description also applies to MCP models: just consider MCP to be an MPEC with a constant objective. The choice of reformulation, and the subsidiary choices each reformulation entails, are controlled by the options described in the section on Setting the Reformulation Options and referenced throughout this section.

The original MPEC model is given as:

$$\min_{x \in \mathbf{R}^n, y \in \mathbf{R}^m} f(x, y) \tag{1}$$

subject to the constraints

$$g(x,y) \leq 0 \tag{2}$$

and

$$y \text{ solves MCP}(h(x,\cdot),\mathbf{B}). \tag{3}$$

In most of the reformulations, the objective function (1) is included in the reformulated model without change. In some cases, it may be augmented with a penalty function. The variables x are typically called upper level variables (because they are associated with the upper level optimization problem) whereas the variables y are sometimes termed lower level variables.

The constraints (2) are standard nonlinear programming constraints specified in GAMS in the standard fashion. In particular, these constraints may be less than inequalities as shown above, or equalities or greater than inequalities. The constraints will be unaltered by all our reformulations. These constraints may involve both x and y, or just x or just y, or may not be present at all in the problem.

The constraints of interest are the equilibrium constraints (3), where (3) signifies that $y \in \mathbf{R}^m$ is a solution to the mixed complementarity problem (MCP) defined by the function $h(x,\cdot)$ and the box $\mathbf{B}$ containing (possibly infinite) simple bounds on the variables y. A point y with $a_i \leq y_i \leq b_i$ solves (3) if for each i at least one of the following holds:

$$\begin{array}{c} h_i(x,y) = 0 \\ h_i(x,y) \geq 0,\ y_i = a_i; \\ h_i(x,y) \leq 0,\ y_i = b_i. \end{array} \tag{4}$$

As a special case of (4), consider the case where $a = 0$ and $b = +\infty$. Since y_i can never be $+\infty$ at a solution, (4) simplifies to the nonlinear complementarity problem (NCP):

$$0 \leq h_i(x,y), 0 \leq y_i \text{ and } y_i h_i(x,y) = 0, i = 1,\ldots,m \tag{5}$$

namely that h and y are nonnegative vectors with h perpendicular to y. This motivates our shorthand for (4), the "perp to" symbol $\perp$:

$$h_i(x,y) \quad \perp \quad y_i \in [a_i, b_i] \tag{6}$$

The different ways to force (6) to hold using (smooth) NLP constraints are the basis of the NLPEC solver.

We introduce a simple example now that we will use throughout this document for expositional purposes:

$$\begin{array}{cc} \min\limits_{x_1,x_2,y_1,y_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & y_1 - y_2 + 1 \leq x_1 \perp y_1 \geq 0 \\ & x_2 + y_2 \perp y_2 \in [-1,1] \end{array}$$

This problem has the unique solution $x_1 = 0$, $x_2 = -1$, $y_1 = 0$, $y_2 = 1$. Note that $f(x,y) = x_1 + x_2$ and $g(x,y) = x_1^2 + x_2^2 - 1$ are the objective function and the standard nonlinear programming constraints for this problem. The function $h(x,y)$ is given by:

$$h(x,y) = \begin{bmatrix} x_1 - y_1 + y_2 - 1 \\ x_2 + y_2 \end{bmatrix}$$

and

$$a = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad b = \begin{bmatrix} \infty \\ 1 \end{bmatrix}.$$

This example is written very succinctly in GAMS notation as:

```
$TITLE simple mpec example

variable f, x1, x2, y1, y2;
positive variable y1;
y2.lo = -1;
y2.up =  1;

equations cost, g, h1, h2;

cost..  f =E= x1 + x2;
g..     sqr(x1) + sqr(x2) =L= 1;
h1..    x1 =G= y1 - y2 + 1;
h2..    x2 + y2 =N= 0;

model example / cost, g, h1.y1, h2.y2 /;
solve example using mpec min f;
```

Note that the equation `cost` is used to define f, the constraint `g` defines the function g, and h is defined by `h1` and `h2`. The complementarity constraints utilize the standard GAMS convention of specifying the orthogonality relationship between h and y in the `model` statement. The interpretation of the "." relies on the bounds a and b that are specified using `positive`, `negative`, or `lo` and `up` keywords in GAMS. Note that since `h2` really specifies a function h_2 and not a constraint $h_2(x, y) = 0$, we use the GAMS syntax `=N=` to ensure this is clear here. Since the relationships satisfied by h_1 and h_2 are determined by the bounds, `=G=` could also be replaced by `=N=` in `h1`.

In describing the various reformulations for (6), it is convenient to partition the y variables into free $\mathcal{F}$, lower bounded $\mathcal{L}$, upper bounded $\mathcal{U}$ and doubly bounded $\mathcal{B}$ variables respectively, that is:

$$\mathbf{B} := \{y = (y_{\mathcal{F}}, y_{\mathcal{L}}, y_{\mathcal{U}}, y_{\mathcal{B}}) : a_{\mathcal{L}} \leq y_{\mathcal{L}},\ y_{\mathcal{U}} \leq b_{\mathcal{U}},\ a_{\mathcal{B}} \leq y_{\mathcal{B}} \leq b_{\mathcal{B}}\}.$$

We will assume (without loss of generality) that $a_{\mathcal{B}} < b_{\mathcal{B}}$. If $a_i = b_i$ then (6) holds trivially for the index i and we can remove the constraint h_i and its corresponding (fixed) variable y_i from the model. The complementarity condition for variables in $y_i \in \mathcal{F}$ is simply the equality $h_i(x, y) = 0$ so these equality constraints are moved directly into the NLP constraints g of the original model as equalities. Thus, NLPEC needs only to treat the singly-bounded variables in $\mathcal{L}$ and $\mathcal{U}$ and the doubly-bounded variables in $\mathcal{B}$. In the above example, $\mathcal{L} = \{1\}$, $\mathcal{U} = \emptyset$ and $\mathcal{B} = \{2\}$.

5.28.3.1 Product reformulations

Product reformulations all involve products of y_i with h_i, or products of y_i with some auxiliary or slack variables that are set equal to h_i. The underlying point is that the constraints (3) are entirely equivalent to the following system of equalities and inequalities:

$$\begin{gathered} w_{\mathcal{L}} = h_{\mathcal{L}}(x, y),\ a_{\mathcal{L}} \leq y_{\mathcal{L}},\ w_{\mathcal{L}} \geq 0 \text{ and } (y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = 0 \\ v_{\mathcal{U}} = -h_{\mathcal{U}}(x, y),\ y_{\mathcal{U}} \leq b_{\mathcal{U}},\ v_{\mathcal{U}} \geq 0 \text{ and } (b_{\mathcal{U}} - y_{\mathcal{U}})^T v_{\mathcal{U}} = 0 \\ w_{\mathcal{B}} - v_{\mathcal{B}} = h_{\mathcal{B}}(x, y),\ a_{\mathcal{B}} \leq y_{\mathcal{B}} \leq b_{\mathcal{B}},\ w_{\mathcal{B}} \geq 0,\ v_{\mathcal{B}} \geq 0 \\ (y_{\mathcal{B}} - a_{\mathcal{B}})^T w_{\mathcal{B}} = 0,\ (b_{\mathcal{B}} - y_{\mathcal{B}})^T v_{\mathcal{B}} = 0. \end{gathered} \tag{7}$$

Note that each inner product is a summation of products of nonnegative terms: a slack variable and the difference between a variable and its bound. In each of these products, either the slack variable or its complement must be zero in order to have a solution. Complementarity is forced by the multiplication of these two terms. The above reformulation is specified using option `reftype mult`.

There are a number of variations on this theme, all of which can be specified via an options file. All of the inner products could be put into the same equation, left as in (7) above, or broken out into individual products (one for each $i \in \mathcal{L} \cup \mathcal{U}$, two for each $i \in \mathcal{B}$). For example, the complementarity constraints

associated with lower bounded variables involve nonnegativity of $w_{\mathcal{L}}$, $y_{\mathcal{L}} \geq a_{\mathcal{L}}$ and either of the following alternatives:

$$(y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = \sum (i \in \mathcal{L}(y_i - a_i)w_i = 0$$

or

$$(y_i - a_i)w_i = 0,\ i = 1, \ldots, m$$

These different levels of aggregation are chosen using option `aggregate none|partial|full`.

Since all of the inner products in (7) involve nonnegative terms, we can set the inner products equal to zero or set them ≤ 0 without changing the feasible set. To choose one or the other, use the option `constraint equality|inequality`.

As a concrete example, consider the option file

```
reftype mult
aggregate none
constraint inequality
```

applied to the simple example given above. Such an option file generates the nonlinear programming model:

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 \leq \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1, 1] \\ & (y_2 + 1)w_2 \leq \mu,\ (1 - y_2)v_2 \leq \mu \end{array} \tag{8}$$

By default, a single model is generated with the value μ set to 0. There are many examples (e.g. interior point codes, many LP and NLP packages, published results on reformulation approaches to MPEC) that illustrate the value of starting with a "nearly-complementary" solution and pushing the complementarity gap down to zero. For this reason, the inner products in (7) above are always set equal to (or $\leq$) a scalar μ instead of zero. By default μ is zero, but options exist to start μ at a positive value (e.g. `InitMu 1e-2`), to decrease it by a constant factor in a series of looped solves (e.g. `NumSolves 4`, `UpdateFac 0.1`), and to solve one last time with a final value for μ (e.g. `FinalMu 0`). If the following lines are added to the option file

```
initmu 1.0
numsolves 4
```

then five consecutive solves of the nonlinear program (8) are performed, the first one using $\mu = 1$ and each subsequent solve dividing μ by 10 (and starting the NLP solver at the solution of the previous model in this sequence).

As a final example, we use a combination of these options to generate a sequence of nonlinear programs whose solutions attempt to trace out the "central path" favored by interior point and barrier algorithms:

```
reftype mult
constraint equality
initmu 1.0
numsolves 4
updatefac 0.1
finalmu 1e-6
```

produces 6 nonlinear programs of the form

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 = \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1, 1],\ (y_2 + 1)w_2 = \mu,\ (y_2 - 1)v_2 = \mu \end{array}$$

for values of $\mu = 1, 0.1, 0.01, 0.001, 0.0001$ and $1e-6$.

Slacks and doubly bounded variables

Slack variables can be used to reduce the number of times a complex nonlinear expression appears in the nonlinear programming model, as was carried out in (7). For a simpler illustrative example the NCP constraints (5) are equivalent to the constraints:

$$w_i = h_i(x, y), 0 \leq w_i, 0 \leq y_i \text{ and } y_i w_i = 0, i = 1, \ldots, m$$

This reformulation has an additional equality constraint, and additional variables w, but the expression h_i only appears once. There are cases when this formulation will be preferable, and the simple option `slack none|positive` controls the use of the w variables.

When there are doubly bounded variables present, these two slack options work slightly differently. For the `positive` case, the reformulation introduces two nonnegative variables w_i and v_i that take on the positive and negative parts of h_i at the solution as shown in (7). Since this is the default value of the option `slack`, the example (8) shows what ensues to both singly and doubly bounded variables under this setting.

For the case `slack none`, Scholtes proposed a way to use a multiplication to force complementarity that requires no slack variables:

$$h_i \quad \perp \quad a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i,\ (y_i - a_i)h_i \leq \mu,\ (y_i - b_i)h_i \leq \mu \tag{9}$$

Note that unlike the inner products in Section Reformulation, we can expect that one of the inequalities in (9) is unlikely to be binding at a solution (i.e. when h_i is nonzero). Therefore, we cannot use an equality in this reformulation, and furthermore the products must not be aggregated. Thus, if you use this option, the reformulation automatically enforces the additional options `constraint inequality` and `aggregate none` on the doubly bounded variables, even if the user specifies a conflicting option. Thus the option file

```
reftype mult
slack none
```

results in the model

$$\begin{array}{ll} \min\limits_{x_1,x_2,y_1,y_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & x_1 - y_1 + y_2 - 1 \geq 0, y_1 \geq 0 \\ & (x_1 - y_1 + y_2 - 1)y_1 = \mu \\ & y_2 \in [-1, 1],\ (y_2 + 1)(x_2 + y_2) \leq \mu,\ (y_2 - 1)(x_2 + y_2) \leq \mu \end{array}$$

Note that the complementarity constraint associated with y_1 is an equality (the default) while the constraints associated with y_2 are inequalities for the reasons outlined above.

In the case of doubly bounded variables, a third option is available for the slack variables, namely `slack one`. In this case, only one slack is introduced, and this slack removes the need to write the function h_i twice in the reformulated model as follows:

$$h_i(x, y) \quad \perp \quad a_i \leq y_i \leq b_i \iff a_i \leq y_i \leq b_i,\ w_i = h_i(x, y),\ (y_i - a_i)w_i \leq \mu,\ (y_i - b_i)w_i \leq \mu$$

Note that the slack variable w that is introduced is a free variable. It is not known before solving the problem whether w_i will be positive or negative at the solution.

We take this opportunity to introduce a simple extension to our option mechanism, namely the ability to set the options for singly and doubly bounded variables differently. For example, the option file

```
reftype mult
slack positive one
```

sets the option `slack positive` for the singly bounded variables and the option `slack one` for the doubly bounded variables resulting in the model

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_1,w_2} & x_1 + x_2 \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0, y_1 \geq 0 \\ & w_1 y_1 = \mu_1 \\ & w_2 = x_2 + y_2,\ y_2 \in [-1, 1],\ (y_2 + 1)w_2 \leq \mu_2,\ (y_2 - 1)w_2 \leq \mu_2 \end{array}$$

Additional options such as

```
initmu 1.0 3.0
numsolves 2
updatefac 0.1 0.2
```

allow the values of μ for the singly and doubly bounded variables to be controlled separately. In this case μ_1 takes on values of 1, 0.1 and 0.01, while μ_2 takes on values 3.0, 0.6 and 0.12 in each of the three nonlinear programming models generated.

5.28.3.2 NCP functions

An NCP-function is a function $\phi(r, s)$ with the following property:

$$\phi(r, s) = 0 \Longleftrightarrow r \geq 0, s \geq 0, rs = 0$$

Clearly, finding a zero of an NCP-function solves a complementarity problem in (r, s). We can replace the inner products of nonnegative vectors in (7) with a vector of NCP functions whose arguments are complementary pairs, e.g. $(y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} = 0$ becomes $\phi(y_i - a_i, w_i) = 0, i \in \mathcal{L}$ and arrive at another way to treat the complementarity conditions. Note that an NCP function forces both nonnegativity and complementarity, so constraints to explicitly force nonnegativity are not required, though they can be included.

Examples of NCP functions include the min function, `min(r,s)`, and the Fischer-Burmeister function

$$\phi(r, s) = \sqrt{r^2 + s^2} - r - s$$

There is no requirement that an NCP function be nonnegative everywhere (it may be strictly negative at some points), so there is little point in setting the option `constraint`; it will automatically take on the value `constraint equality`. NCP functions cannot be aggregated, so the `aggregate` option will always be set to `none`.

Since the arguments to the NCP functions are going to be nonnegative at solution, we cannot use the functions h_i directly in the case of doubly-bounded variables. We must use slacks $w - v = h_i$ to separate h_i into its positive and negative parts (but see Section Doubly bounded variables below). The slacks can be `positive` or `free`, since the NCP function will force positivity at solution. For the singly-bounded variables, slacks are optional, and can also be `positive` or `free`.

Both of the NCP functions mentioned above suffer from being non-differentiable at the origin (and at points where $r = s$ for the min function). Various smoothed NCP-functions have been proposed that are differentiable. These smooth functions are parameterized by μ, and approach the true NCP-function as the smoothing parameter approaches zero. For example, the Fischer-Burmeister function includes a perturbation μ that guarantees differentiability:

$$\phi_{FB}(r, s) := \sqrt{r^2 + s^2 + 2\mu} - (r + s). \tag{10}$$

You can choose these particular NCP functions using option `RefType min|FB|fFB`. The difference between the last two is that `RefType FB` writes out GAMS code to compute the function ϕ_{FB}, while `RefType fFB` makes use of the GAMS intrinsic function NCPFB(r,s,mu) that computes ϕ_{FB} internally. In general, using the GAMS intrinsic function should work better since the intrinsic can guard against overflow, scale the arguments before computing the function, and use alternative formulas that give more accurate results for certain input ranges.

As an example, the option file

```
reftype fFB
slack free
initmu 1e-2
```

generates the reformulation

$$\begin{aligned} \min_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1 \\ & \phi_{FB}(w_1, y_1, \mu) = 0 \\ & w_2 - v_2 = x_2 + y_2 \\ & \phi_{FB}(y_2 + 1, w_2, \mu) = 0,\ \phi_{FB}(1 - y_2, v_2, \mu) = 0 \end{aligned}$$

with a value of $\mu = 0.01$. Following a path of solutions for decreasing values of μ is possible using the options discussed above.

Each of the two arguments to the NCP function will be nonnegative at solution, but for each argument we have the option of including a nonnegativity constraint explicitly as well. This results in the 4 values for the option `NCPBounds none|all|function|variable`. When no slacks are present, this option controls whether to bound the function h_i as well as including it in the NCP function, e.g. $h_i \geq 0, \phi(h_i, y_i - a_i) = 0$. When slacks are present, we require that the slack setting be consistent with the bound setting for the function argument to the NCP function, where `NCPBounds none|variable` is consistent with free slack variables and `NCPBounds all|function` is consistent with positive slack variables.

Thus, the option file

```
reftype min
slack positive
NCPBounds function
```

generates the reformulation

$$\begin{aligned} \min_{x_1,x_2,y_1,y_2,w_1,w_2,v_2} \quad & x_1 + x_2 \\ \text{subject to} \quad & x_1^2 + x_2^2 \leq 1 \\ & w_1 = x_1 - y_1 + y_2 - 1, w_1 \geq 0 \\ & \min(w_1, y_1) = \mu \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0 \\ & \min(y_2 + 1, w_2) = \mu,\ \min(1 - y_2, v_2) = \mu \end{aligned}$$

The `NCPBounds function` option means that the variable argument to the NCP function (in this case y) does not have its bounds explicitly enforced. It should be noted that this nonlinear program has nondifferentiable constraints for every value of μ. For this reason, the model is constructed as a `dnlp` model (instead of an `nlp` model) in GAMS.

A smoothed version of the min function was proposed by Chen & Mangasarian:

$$\phi_{CM}(r, s) := r - \mu \log(1 + \exp((r - s)/\mu)). \tag{11}$$

This function is not symmetric in its two arguments, so $\phi_{CM}(r, s) \neq \phi_{CM}(s, r)$. For this reason, we distinguish between the two cases. Unlike the Fischer-Burmeister function ϕ_{FB}, ϕ_{CM} is not defined in the limit (i.e. for $\mu = 0$) if you use GAMS code to compute it. However, the GAMS intrinsic NCPCM(r,s,mu) handles this limit case internally. The option `RefType CMxf|CMfx|fCMxf|fCMfx` chooses a reformulation based on the function ϕ_{CM}. Again, the last two choices use the GAMS intrinsic function.

Doubly bounded variables

Like the mult reformulation (7), reformulations using NCP functions are appropriate as long as we split the function h_i matching a doubly-bounded variable into its positive and negative parts $w_i - v_i = h_i$. To avoid this, Billups has proposed using a composition of NCP functions to treat the doubly-bounded case:

$$h_i \quad \perp \quad a_i \leq y_i \leq b_i \iff \phi_{FB}(y_i - a_i, \phi_{FB}(b_i - y_i, -h_i)) = 0 \tag{12}$$

Use option `RefType Bill|fBill` to choose such a reformulation for the doubly-bounded variables. The first option value writes out the function in explicit GAMS code, while the second writes it out using the GAMS intrinsic function NCPFB.

5.28.3.3 Penalty functions

All of the reformulations discussed so far have reformulated the complementarity conditions as constraints. It is also possible to treat these by moving them into the objective function with a penalty parameter $1/\mu$: as μ goes to zero, the relative weight placed on complementarity increases. Ignoring the NLP constraints, we can rewrite the original MPEC problem as

$$\min_{x \in \mathbf{R}^n, y \in \mathbf{R}^m} f(x,y) + \frac{1}{\mu}((y_{\mathcal{L}} - a_{\mathcal{L}})^T w_{\mathcal{L}} + (b_{\mathcal{U}} - y_{\mathcal{U}})^T v_{\mathcal{U}} + (y_{\mathcal{B}} - a_{\mathcal{B}})^T w_{\mathcal{B}} + (b_{\mathcal{B}} - y_{\mathcal{B}})^T v_{\mathcal{B}}) \tag{13}$$

subject to the constraints

$$\begin{gathered} g(x,y) \leq 0 \\ w_{\mathcal{L}} = h_{\mathcal{L}}(x,y),\ a_{\mathcal{L}} \leq y_{\mathcal{L}},\ w_{\mathcal{L}} \geq 0 \\ v_{\mathcal{U}} = -h_{\mathcal{U}}(x,y),\ y_{\mathcal{U}} \leq b_{\mathcal{U}},\ v_{\mathcal{U}} \geq 0 \\ w_{\mathcal{B}} - v_{\mathcal{B}} = h_{\mathcal{B}}(x,y)\ a_{\mathcal{B}} \leq y_{\mathcal{B}} \leq b_{\mathcal{B}},\ w_{\mathcal{B}} \geq 0, v_{\mathcal{B}} \geq 0 \end{gathered} \tag{14}$$

Choose this treatment using option `refType penalty`. The options `aggregate` and `constraint` are ignored, since the inner products here are all aggregated and there are no relevant constraints. It is possible to do a similar reformulation without using slacks, so the options `slack none|positive` can be used in conjunction with this reformulation type.

The following option file shows the use of the `penalty` reformulation, but also indicates how to use a different reformulation for the singly and doubly bounded variables:

```
reftype penalty mult
slack none *
initmu 1.0
numsolves 2
updatefac 0.1 0.2
```

The "$*$" value allows the `slack` option to take on its existing value, in this case `positive`. Applied to our simple example given above, such an option file generates the nonlinear programming model:

$$\begin{array}{rc} \min\limits_{x_1,x_2,y_1,y_2,w_2,v_2} & x_1 + x_2 + \frac{1}{\mu_1} y_1 (x_1 - y_1 + y_2 - 1) \\ \text{subject to} & x_1^2 + x_2^2 \leq 1 \\ & x_1 - y_1 + y_2 - 1 \geq 0, y_1 \geq 0 \\ & w_2 - v_2 = x_2 + y_2, w_2, v_2 \geq 0, y_2 \in [-1,1] \\ & (y_2 + 1) w_2 \leq \mu_2,\ (1 - y_2) v_2 \leq \mu_2 \end{array}$$

The penalty parameter μ_1 is controlled separately from the doubly bounded constraint parameter μ_2. For consistency with other options, the penalty parameter in the objective is $1/\mu$ meaning that as μ_1 tends to zero, the penalty increases. The option `initmu` has only one value, so both the singly and doubly bounded μ values are initialized to 1. In the above example, three solves are performed with $\mu_1 = 1, 0.1$ and 0.01 and $\mu_2 = 1, 0.2$ and 0.04.

5.28.3.4 Testing for complementarity

In some cases a solution to the reformulated model may not satisfy the complementarity constraints of the original MPEC, e.g. if a large penalty parameter is used in the reformulation. It can also happen that the solution tolerances used in the NLP solver allow solutions with small error in the NLP model but large error in the original MPEC. For example if $x = f(x) = .001$ then the NLP constraint $xf(x) = 0$ may satisfy the NLP feasibility tolerance but it's not so easy to claim that either x or $f(x)$ is zero. The NLPEC solver includes a check that the proposed solution does in fact satisfy the complementarity constraints. The complementarity gap is computed using the definition common to all GAMS MCP solvers in computing the *objval* model attribute for an MCP model. The tolerance used for this complementarity gap can be adjusted using the `testtol` option.

5.28.4 Options

For details on how to create and use an option file, see the introductory chapter on solver usage.

For most GAMS solvers, the use of an options file is discouraged, at least for those unfamiliar with the solver. For NLPEC, however, we expect that most users will want to use an options file from the very beginning. NLPEC is as much a tool for experimentation as it is a solver, and as such use of the options file is encouraged.

Option values can take many different types (e.g. strings, integers, or reals). Perhaps the most important option to remember is one with no value at all: the `help` option. `Help` prints a list of the available options, along with their possible values and some helpful text. The options file is read sequentially, so in case an option value is set twice, the latter value takes precedence. However, any consistency checks performed on the options values (e.g. `RefType fBill` cannot be used with `aggregate full`) are made after the entire options file is read in, so the order in which different options appear is not important, provided the options are not specified twice.

5.28.4.1 Setting the Reformulation Options

While NLPEC has many options, there is a small set of five options that, taken together, serve to define the type of reformulation used. Listed in order of importance (highest priority items first), these *reformulation options* are the `RefType`, `slack`, `constraint`, `aggregate` and `NCPBounds` options. In some cases, setting the highest-priority option `RefType` is enough to completely define a reformulation (e.g. `RefType penalty` in the case of doubly-bounded variables). In most cases though, the lower-priority options play a role in defining or modifying a reformulation. It's useful to consider the reformulation options in priority order when creating option files to define reformulations.

Some of the combinations of the reformulation options don't make sense. For example, the use of an NCP function to force complementarity between its two input arguments requires a separate function for each complementary pair, so setting both `RefType min` and `aggregate full` is inconsistent. NLPEC implements consistency checks on the reformulation options using the priority order: Given a consistent setting of the higher priority options, the next-highest priority option is checked and, if necessary, reset to be consistent with the items of higher priority. The end result is a set of consistent options that will result in a working reformulation. NLPEC prints out the pre- and post-checked sets of reformulation options, as well as warning messages about changes made. In case you want to use an option that NLPEC doesn't think is consistent, you can use the `NoCheck` option: this supresses the consistency checks.

Each of the reformulation options in the table below takes two values - one for the singly-bounded variables in $\mathcal{L} \cup \mathcal{U}$ and another for the doubly-bounded variables in $\mathcal{B}$. If one option value appears, it sets both option values. When setting both option values, use an asterisk "$*$" to indicate no change. So for example, an option file

```
RefType  fCMxf
RefType  *  fBill
```

first sets the `RefType` to `fCMxf` for all variable types, and then resets the `RefType` to `fBill` for doubly-bounded variables.

5.28.4.2 Reformulation Options

Option	Description	Default
aggregate	controls constraint aggregation Determines if certain constraints are aggregated or not. E.g. to force `w >= 0` and `y >= 0` to be complementary we can write either `w^T y <= 0` or `w_i^T y_i <= 0, for all i`. `none`: use no aggregation `partial`: aggregate doubly-bounded variables separately from others `full`: use maximum aggregation possible	`none`
constraint	controls use of equality/inequality Determines if certain constraints are written down using equalities or inequalities. E.g. to force `w >= 0` and `y >= 0` to be complementary we can write either `w^T y <= 0` or `w^T y = 0`. This option only plays a role when bounding a quantity whose sign cannot be both positive and negative and which must be 0 at a solution. `equality`: use =E= constraints `inequality`: use =L= constraints	`equality`
NCPBounds	sets explicit bounds on arguments of NCP functions Determines which of the two arguments to an NCP function `Phi(r,s)` are explicitly constrained to be nonnegative. The explicit constraints are in addition to those imposed by the constraint `Phi(r,s) = 0`, which implies nonnegativity of `r` and `s`. `none`: no explicit bounds `function`: explicit bound on function/slack argument `variable`: explicit bound on variable argument `all`: explicit bound on both function and variable arguments	`none`
refType	reformulation type Determines the type of reformulation used. Our notation and descriptions are taken from a special case of the MPEC, the NCP: find `x >= 0, f(x) >= 0, x^T f(x) = 0`. `mult`: inner product <x,f> = 0 `min`: NCP function min(x,f) `CMxf`: Chen-Mangasarian NCP function, explicit `CMfx`: Chen-Mangasarian NCP function, explicit `fCMxf`: Chen-Mangasarian NCP function, intrinsic `fCMfx`: Chen-Mangasarian NCP function, intrinsic `FB`: Fischer-Burmeister NCP function, explicit `fFB`: Fischer-Burmeister NCP function, intrinsic `FB_neg`: Fischer-Burmeister NCP function negated, explicit `fFB_neg`: Fischer-Burmeister NCP function negated, intrinsic `Bill`: Billups function for doubly-bounded variables, explicit `fBill`: Billups function for doubly-bounded variables, intrinsic `penalty`: weighted penalization of non-complementarity in objective `median`: median function for doubly-bounded variables, explicit `fVUsin`: Veelken-Ulbrich NCP function (smoothed min), intrinsic `fVUpow`: Veelken-Ulbrich NCP function (smoothed min), intrinsic	`mult`
slack	control use of slacks for function values Determines if slacks are used to treat the functions `h_i`. For single-bounded variables, we use at most one slack (either free or positive) for each `h_i`. For doubly-bounded variables, we can have no slacks, one slack (necessarily free), or two slacks (either free or positive) for each `h_i`. `none`: no slacks will be used `free`: free slacks will be used `positive`: non-negative slacks will be used `one`: one free slack will be used for each h_i in the doubly bounded case	`positive`

5.28.4.3 General Options

Option	Description	Default
allSolves	do all solves in a loop regardless of previous failure In case multiple (looped) solves are specified, the default is to skip subsequent solves when any solve terminates without getting a solution. Setting this flag removes the check and all solves are done, regardless of previous failures.	0
dotGams	name of gams source file for scalar model	auto
dumpValid	dump valid reformulation options to a GDX file and exit	0
equReform	outdated and deprecated Range: {0, ..., 33}	0
finalMu	final value of parameter mu If specified, an extra solve is carried out with mu set to this value. Can be set independently for singly and doubly bounded variables.	none
initMu	initial value of parameter mu A single solve of the nonlinear program is carried out for this value. Note that mu must be positive for some settings of reftype, e.g. penalty. Can be set independently for singly and doubly bounded variables.	0
initSLo	lower bound for artificials added to the problem Range: [-∞, ∞]	0
initSUp	upper bound for artificials added to the problem Range: [-∞, ∞]	+inf
noCheck	do not check consistency of reformulation options	0
numSolves	number of looped solves This should be set in conjunction with the updateFac option, so that the mu values are lowered successively.	0
parmFile	extra GAMS options for running scalar model	
subSolver	controls what subsolver to run If this option is not specified, the usual GAMS rules for selecting the solver to run are used.	auto
subSolverOpt	optfile value to pass to the subsolver	0
terminate	terminate after generating scalar GAMS source code	0
testTol	tolerance for complementarity check in MPEC/MCP	1e-005
updateFac	update factor for mu The factor that multiplies mu before each of the extra solves triggered by the numsolves option. Can be set independently for singly and doubly bounded variables. Range: [1e-280, 1.0]	1e-1

5.28.4.4 The Outdated equreform Option

In the early versions of NLPEC the only way to set the reform type was via the equreform option. Each valid equreform value represented a preselected combination of the options from Section Setting the Reformulation Options. This made it difficult to experiment with combinations not preselected, so the options in Section Setting the Reformulation Options were added. Be default, the equreform option has value 0 and is not used. To get the old behavior, set equreform to a positive value - this will force the options in Section Setting the Reformulation Options to be ignored. The general options in Section General Options are used no matter how the reformulation type is selected - via RefType or equreform.

Option	Description	Default
equreform	Old way to set the type of reformulation used.	0

The values allowed for equreform and their implications are given by the following table.

	L/U				B			
equref	reftype	sign	slacks	free-y	reftype	sign	slacks	free-y
1	$<,>_i$	$=\mu$	bnd		$<,>_i$	$=\mu$	bnd	
2	$<,>_i$	$<=\mu$	bnd		$<,>_i$	$<=\mu$	bnd	
3	$<,>_i$	$=\mu$	bnd		Scholtes	$<=\mu$	one	
4	$<,>_{L+U+B}$	$=\mu$	bnd		$<,>_{L+U+B}$	$=\mu$	bnd	
5	$<,>_{L+U+B}$	$=\mu$	none		$<,>_{L+U+B}$	$=\mu$	bnd	
6	$<,>_{L+U}$	$=\mu$	none		Scholtes	$<=\mu$	one	
7	$<,>_{L+U}$	$<=\mu$	none		Scholtes	$<=\mu$	none	
8	$<,>_i$	$=\mu$	none		Scholtes	$<=\mu$	none	
9	$<,>_{\text{obj}}$		bnd		$<,>_{\text{obj}}$		bnd	
10	$<,>_{\text{obj}}$		none		$<,>_{\text{obj}}$		bnd	
11	$<,>_i$	$=\mu$	none		$<,>_i$	$=\mu$	bnd	
12	F-B	$=0$	none	free	F-B	$=0$	free	free
13	F-B	$=0$	none	free	Billups	$=0$	none	free
14	min	$=\mu$	free	free	min	$=\mu$	free	free
15	min	$<=\mu$	bnd		min	$<=\mu$	bnd	
16	C-M(x,f)	$=0$	free	free	C-M(x,f)	$=0$	free	free
17	C-M(x,f)	$=0$	bnd		C-M(x,f)	$=0$	bnd	
18	$<,>_{L,U}$	$<=\mu$	none		$<,>_B$	$<=\mu$	bnd	
19	$<,>_i$	$<=\mu$	none		$<,>_i$	$<=\mu$	bnd	
20	$<,>_{L,U}$	$<=\mu$	bnd		$<,>_B$	$<=\mu$	bnd	
21	$<,>_{L+U+B}$	$<=\mu$	bnd		$<,>_{L+U+B}$	$<=\mu$	bnd	
22	F-B	$=0$	bnd		F-B	$=0$	bnd	
23	F-B	$=0$	free	free	F-B	$=0$	free	free
24	C-M(f,x)	$=0$	none	free	C-M(f,x)	$=0$	free	free
25	C-M(f,x)	$=0$	none		C-M(f,x)	$=0$	bnd	
26	NCPF	$=0$	none	free	NCPF	$=0$	free	
27	NCPF	$=0$	none	free	Billups†	$=0$	none	free
28	NCPF	$=0$	bnd		NCPF	$=0$	bnd	
29	NCPF	$=0$	free	free	NCPF	$=0$	free	free
30	NCPCM(x,f)	$=0$	free	free	C-M(x,f)	$=0$	free	free
31	NCPCM(x,f)	$=0$	bnd		C-M(x,f)	$=0$	bnd	
32	NCPCM(f,x)	$=0$	none	free	NCPCM(f,x)	$=0$	free	free
33	NCPCM(f,x)	$=0$	none		NCPCM(f,x)	$=0$	bnd	

Table 1: Values for equreform option.

5.28.5 Open Architecture

In this section we describe the architecture of the NLPEC solver, i.e. the way the solver is put together. This should be useful to anybody using NLPEC for experiments or to those wanting to know the details of how NLPEC works.

The foundation for the NLPEC solver is the software library (also used in the GAMS/CONVERT solver) that allows us to write out a scalar GAMS model that is mathematically equivalent to the original, or

to write out selected pieces of such a model. Using this software, NLPEC creates a GAMS NLP model using one of the reformulation strategies from Section Reformulation. This model may contain many new variables and/or equations, but it will surely contain the (non)linear expressions defining the original model as well. Once the scalar model has been created, NLPEC calls GAMS to solve this model, using the current NLP solver. The option parmFile can be used to pass on additional options to this GAMS job. After the model has solved, NLPEC reads the NLP solution, extracts the MPEC solution from it, and passes this MPEC solution back to GAMS as it terminates.

There are a number of advantages to this architecture. First, its openness makes it easy to see exactly what reformulation is being done. The intermediate scalar GAMS NLP model can be made available after the run by either saving the scratch directory (i.e. run with `keep=1`) or using the `dotGams` option to select an alternate file name. This intermediate model contains all the details of the reformulation. It can be used for debugging in case things didn't work out as expected. It is also possible to modify this file to do some quick and dirty experiments with similar reformulation strategies. Another advantage is the variety of NLP solvers that can be plugged in to solve the reformulated model. There is no need to program (and debug) an interface to an NLP package to run experiments with an NLP solver - the existing GAMS link is all that is needed. It is also easy to experiment with non-default solver options that may be more appropriate for reformulated MPEC models or for a particular choice of reformulation.

5.29 ODHCPLEX

5.29.1 Introduction

GAMS/ODHCPLEX is a solver from Optimization Direct Inc. that implements a set of heuristic methods (named ODHeuristics) for finding feasible solutions to Mixed Integer Programming (MIP and MIQCP) models that uses IBM CPLEX as its underlying solver engine. It is designed for large-scale models which a MIP solver would find intractable: either by it being unable to find feasible solutions at all or; more usually, by being unable to find feasible solutions of adequate quality in the time available to its user.

It is intended for users who are familiar with MIP modelling and have some knowledge of using the GAMS/CPLEX solver. GAMS/ODHCPLEX does not demand expert specialism in this field.

ODHCPLEX can be used in two ways: it is implemented as a stand-alone ODHeuristic engine, which can be used on its own (ODHeuristicMethod=STANDALONE); and also within the CPLEX optimizer, within which it can supply and receive solutions from the main CPLEX caller (ODHeuristicMethod=ODH-CPLEX (default)) thereby accelerating optimization compared with GAMS/CPLEX run on its own.

The ODHeuristic engine has a heuristic method for finding an initial feasible solution that it designed to complement, those of CPLEX. Since its main algorithmic procedure works by improving an incumbent feasible solution, getting an initial one is important and may consume a significant part of its total runtime. When used on its own (i.e. ODHeuristicMethod=STANDALONE), users should experiment to discover whether ODHeuristics' or CPLEX's initial feasible solution methods work best, but within ODH-CPLEX (i.e. ODHeuristicMethod=ODH-CPLEX) both methods are run in parallel and the winner is chosen automatically.

ODHeuristics' principal algorithm works by solving a sequence of sub-models. An innovative aspect of this process is its ability to use the model's symbolic structure to achieve the sub-model decomposition. It does this by analyzing the symbolic names that the user gives to the decision variables and careful specification of how this should be done this is worthwhile. ODHCPLEX can work without this analysis, but it usually takes about twice as much runtime.

5.29.2 Specifying Model Structure

The ODHeuristic method needs to break the model down into sub-models. It can do this in one of three ways:

1. Automatically using its decomposition heuristic;
2. Using information specified by the user in the IndexKey parameter;
3. By simply assigning each variable to a different block (or *key*); or
4. By using the dot option notatation with the option .key.

By default, the program will use information specified by the IndexKey parameter if it is set and its automatic decomposition heuristic otherwise. This may be overridden by the Decomposition parameter. If it is set to 0 (zero), option 3 above is selected. If it is set to 1, its automatic decomposition method is used, and if it is set to 2, option 4 above is selected.

Whilst the automatic decomposition method often works well, there may be an advantage to specifying decomposition through the IndexKey parameter. After performing the decomposition in whatever way, the program analyses the decomposition and displays statistics showing the maximum and minimum number of variables in each key or block and showing a percentage score to the decomposition as a whole. A typical display is of the form:

```
Variables/key 205.58 (+/-304.79), max/min variables/key 933(32) / 60(113).
There are 227 keys (4149 keys were dropped) with 46872 values.
Decomposition score 13.66%, graph score 2074/3135232.
```

Other things (such as the distribution of variables in keys and the number of keys) being equal, the smaller the percentage decomposition score, the better the decomposition is and the more effective the program will be.

5.29.2.1 Using the IndexKey parameter

The program needs to associate sets of variables with distinct values of a single index. The user can specify this association with a pattern to which some or all of the variables conform. The pattern is in standard C scanf format (see, for example, Kernighan and Ritchie's 2nd edition of The C Programming Language, section B1.3 *Formatted Input*). Currently allowable index values must be non-negative integers, so the pattern must include *d*. For example, if we have variables `x` whose first index names starts with a number of letters then an underscore followed by a numeric index value (like `x(firstone_1)`, `x(another_1)`, `x(firstone_2)`, `x(another_2)`, ..) the pattern

```
x(%*[a-z]_%d)
```

associates those variables whose name ends in _1 with index value 1, those whose name ends in _2 with index value 2 and so on. The pattern is called an *index key* referred to by the program as the option parameter IndexKey, for example

```
IndexKey=%*[xy](t_%d)
```

It may, however, be desirable to consider variables whose names are, say, `x(t_2)` and `y(t_2)`, to belong to different index values, i.e. to belong to different groups. One way of doing this is to identify them with separate index keys. These can be supplied to IndexKey as two fields separated by a semi-colon, for example

```
IndexKey=x(t_%d);y(t_%d)
```

Up to 10 fields can be specified in this way.

On the other hand, it may be desirable to consider such variables as having the same index key and their nomenclature may not permit their identification by a single key field. For example, suppose there are variables `john(t_1)`, `john(t_2)`,.., `jane(t_1)`, `jane(t_2)`,.. and `johnny(t_1)`, `johnny(t_2)`,.. and we want to associate `john(t_x)` and `jane(t_x)` as belonging to key value x, but want to ignore `johnny(t_x)`.

Two key fields can be used to do this by

```
INDEXKEY=john(t_%d);jane(t_%d)
```

By default `john(t_2)` and `jane(t_2)` would not share the same index value, but if the option parameter KeyType, is set to 1, the heuristic will group them together so as to share the same index.

If IndexKey is not specified, the program uses a default decomposition.

The program divides the model up into parts associated with different values of the IndexKey (if specified), using an integer interval divisor. Initially this is a number not less than 2 and it is increased as the search progresses. When no improved solution is found after a number, MaxRepeat, of attempts with the maximum interval divisor, MaxInterDiv, the program terminates. Default values are provided for MaxRepeat and MaxInterDiv, so these do not have to be specified by the user.

5.29.3 Heuristic Parameters

There are a number of other parameters which control the behaviour of the heuristic program. These may be left at their default values or specified on the GAMS/ODHCPLEX option file. In addition, many GAMS/CPLEX options can be supplied in the GAMS/ODHCPLEX option file to tweak the CPLEX behavior, see tables below. Note however that - although they can be technically set - these GAMS/CPLEX options may not have any influence on a GAMS/ODHCPLEX solve. For detailed descriptions of the GAMS/CPLEX options, see Detailed Descriptions of CPLEX Options.

Parameter names and their meanings are summarized in the table below.

5.29.3.1 GAMS/ODHCPLEX: General Options

Option	Description	Default
addcuts	Indicator for adding cuts from CPLEX master solve	2
cpxpresolve	Applies CPLEX presolve to full model	0
decompdensity	Matrix density above which automatic decomposition assigns each variable to a separate key	0.3
decomposition	Model decomposition method.	-1
deterministic	Specifies whether the solution improvement heuristic is run in deterministic or opportunistic (i.e. non-deterministic) mode	1

Option	Description	Default
divisor	Initial divisor for sub models	automatic
dynamicsearch	Search strategy for CPLEX caller and sub-model solves	-1
extracplexlog	Write addition CPLEX output to log	0
feastol	Feasibility tolerance	1e-6
firstfeas	Use first feasible heuristic for finding an initial feasible solution	-1
firstfeascontinue	Whether first feasible heuristic continues when it achieves feasibility	0
firstfeaseffort	Effort limit on first feasible heuristic	-500
firstfeaslpitlim	Limit on number of relaxed re-solves in first feasible heuristic	10
firstfeasrelaxcrit	Smallest sum of infeasibilites/row where relaxed solution used	0.01
firstfeasshift	First feasible heuristic variable shifting in found solutions	0
globalbounds	Use of global bounds from CPLEX caller	2
ignoresetslvrparams	Flag to control whether solver parameters can be dynamically altered by ODH	0
indexkey	Pattern used to match variable names for grouping into sub-models discussed above	
integeronly	Variables to include in INDEXKEY	-1
integertol	Integrality tolerance for variable values	1e-5
interdiv	Initial divisor value	4
.key	Variable block or key number	0
keypartition	Use of solver partition information	0
keysminimum	Minimum number of keys that the automatic decomposition method attempts to find	512
keytype	Treatment of multiple INDEXKEYs	0
localsearch	Indicator for local search heuristic	1
loosefeastol	Loose feasibility tolerance	1e-5
maxbacktrack	The maximum number of backtracks permitted in sub-model solves	-1
maxbound	The largest(smallest) non infinite bound value ODH will accept for upper(lower) bounds	1e+9
maxinfrepeat	Maximum divisor value when solution is infeasible	0
maxinterdiv	Maximum divisor value	0
maxrepeat	Maximum number of sub-model repeat solves for each divisor value	0
objtarget	Target objective value	0
objthreshold	Threshold for absolute value of objective coefficients	1e99
odheuristicmethod	ODHeuristic method section	ODH-CPLEX
odhfeasopt	Optimization method for sub-models in phaseI	0

Option	Description	Default
odhthreads	The number of heuristic threads used by ODH-CPLEX or STANDALONE	-1
odhtimelimit	Elapsed time limit in seconds	GAMS ResLim
penalty	The objective function coefficient value for penalties	-1
penperturb	Perturbation tolerance for penalties coefficients	0
phase12	Specifies whether to use a phaseI/phaseII method to remove infeasibilities	1
processorlock	Thread allocation	0
quickfirstsolve	Accelerate initial CPLEX solve	0
recurse	Recurse using heuristic to solve sub-models when a feasible solution has been obtained	0
recursedecomp	Recursed model decomposition method	0
recurseiterlim	Recursed heuristic iteration limit for sub-solves	40
recurselog	Write thread log files for recursed sub-solves	0
recurseminiterlim	Recursed heuristic minimum iterations before a solution is found in sub-solves	10
recursesoliterlim	Recursed heuristic sub-solves quit after these iterations if a solution is found	maxint
rejectinfsol	Reject infeasible solutions to sub-models	2
relaxsos2	Treatment of SOS2 members	1
seed	Initial random number seed	12345
sosfind	Automatic detection of Special Ordered Sets (SOSs)	0
sosinkey	Assign each SOS to its own sub-model component (key)	0
sosmember	Automatically detect SOSs whose variable member names match this pattern	
sosovar	Automatically detect SOSs whose output variable name matches this pattern	
sosselect	Select sub-set of SOS members	
sosselect16	Select sub-set of SOS members for sets with 16 or fewer members only	
soswvar	Automatically detect SOSs whose input(weight) variable name matches this pattern	
strategy	ODH-Cplex Strategy	1
strictdeterministic	Terminate ODH deterministically when improvement heuristic finishes	0
subcheckfreq	Frequency with which sub-model LPs are interrupted for mutual communication	10
subnodelimit	Node limit for submodel searches	-1
suborder	Use of priority order in sub-solves	1
sub_cpx_threads	Threads availble for the solves within ODHeuristic	1
syncfreq	Thread synchronization frequency in deterministic parallel mode	1

Option	Description	Default
threadlog	Write thread log files	0
threadzerosync	Which CPLEX threads to use for synchronization	0
tightenprebounds	Level of bound tightening in ODH presolved model	1
trialbound	Trial bound heuristic	0
trialboundfile	The trial bound file	
trialboundsetsize	Size adjustment to automatically generated trial bound sets	0
usehistory	Use of past sub-model selections in current selection	-1
varcleanlpmethod	Method used to solve variable cleaning LPs	-1
variableclean	Clean variable values from sub-models	1
zerotol	Zero tolerance for variable values	1e-9

5.29.3.2 GAMS/CPLEX: Preprocessing and General Options

Option	Description	Default
advind	advanced basis use	1
aggfill	aggregator fill parameter	10
aggind	aggregator on/off	-1
calcqcpduals	calculate the dual values of a quadratically constrained problem	1
clocktype	clock type for computation time	2
coeredind	coefficient reduction on/off	-1
cpumask	switch and mask to bind threads to processors (Linux only)	auto
cutoff	GMO cutoff	0.0
datacheck	controls data consistency checking and modeling assistance	0
depind	dependency checker on/off	-1
dettilim	deterministic time limit	1.0e+75
domlim	domain violation number	0
eprelax	relaxation for feasOpt	1.0e-06
feasoptmode	mode of FeasOpt	0
fixoptfile	name of option file which is read just before solving the fixed problem	
folding	LP folding will be attempted during the preprocessing phase	-1
freegamsmodel	preserves memory by dumping the GAMS model instance representation temporarily to disk	0
indicoptstrict	abort in case of an error in indicator constraint in solver option file	1
lpmethod	algorithm to be used for LP problems	0

Option	Description	Default
memoryemphasis	reduces use of memory	0
multobjdisplay	level of display during multiobjective optimization	1
names	load GAMS names into Cplex	1
numericalemphasis	emphasizes precision in numerically unstable or difficult problems	0
optimalitytarget	type of optimality that Cplex targets	0
parallelmode	parallel optimization mode	0
paramdisplay	display the nondefault parameters before optimization	1
predual	give dual problem to the optimizer	0
preind	turn presolver on/off	1
prelinear	linear reduction indicator	1
prepass	number of presolve applications to perform	-1
prereform	set presolve reformulations	3
printoptions	list values of all options to GAMS listing file	0
qextractalg	quadratic extraction algorithm in GAMS interface	0
qpmethod	algorithm to be used for QP problems	0
qtolin	linearization of the quadratic terms in the objective function of a QP or MIQP model	-1
randomseed	sets the random seed differently for diversity of solutions	202009243
record	Records invocations of Callable Library routines	0
reduce	primal and dual reduction type	3
relaxfixedinfeas	accept small infeasibilties in the solve of the fixed problem	0
relaxpreind	presolve for initial relaxation on/off	-1
rerun	rerun problem if presolve infeasible or unbounded	nono
scaind	matrix scaling on/off	0
solutiontype	type of solution (basic or non basic) for an LP or QP	0
threads	global default thread count	0
tilim	overrides the GAMS ResLim option	1.0e+75
tryint	GMO tryint	0.0
tuningdettilim	tuning deterministic time limit per model or suite	1.0e+75
tuningdisplay	level of information reported by the tuning tool	1
tuningmeasure	measure for evaluating progress for a suite of models	1
tuningrepeat	number of times tuning is to be repeated on perturbed versions	1
tuningtilim	tuning time limit per model or suite	1.0e+75
usebasis	GMO usebasis	0

Option	Description	Default
warninglimit	determines how many times warnings of a specific type (datacheck=2) will be displayed	10
workdir	directory for working files	.
workmem	memory available for working storage	2048.0

5.29.3.3 GAMS/CPLEX: Simplex Algorithmic Options

Option	Description	Default
conflictalg	algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model	0
conflictdisplay	decides how much information CPLEX reports when the conflict refiner is working	1
craind	crash strategy (used to obtain starting basis)	1
dpriind	dual simplex pricing	0
dynamicrows	switch for dynamic management of rows	-1
epper	perturbation constant	1.0e-06
netfind	attempt network extraction	2
netppriind	network simplex pricing	0
perind	force initial perturbation	0
perlim	number of stalled iterations before perturbation	0
ppriind	primal simplex pricing	0
pricelim	pricing candidate list	0
reinv	refactorization frequency	0
sifting	switch for sifting from simplex optimization	1

5.29.3.4 GAMS/CPLEX: Simplex Limit Options

Option	Description	Default
itlim	iteration limit	2147483647
netitlim	iteration limit for network simplex	2147483647
objllim	objective function lower limit	-1.0e+75
objulim	objective function upper limit	1.0e+75
singlim	limit on singularity repairs	10

5.29.3.5 GAMS/CPLEX: Simplex Tolerance Options

Option	Description	Default
epmrk	Markowitz pivot tolerance	0.01
epopt	optimality tolerance	1.0e-06
eprhs	feasibility tolerance	1.0e-06
netepopt	optimality tolerance for the network simplex method	1.0e-06
neteprhs	feasibility tolerance for the network simplex method	1.0e-06

5.29.3.6 GAMS/CPLEX: Barrier Specific Options

Option	Description	Default
baralg	algorithm selection	0
barcolnz	dense column handling	0
barcrossalg	barrier crossover method	0
barepcomp	convergence tolerance	1.0e-08
bargrowth	unbounded face detection	1.0e+12
baritlim	iteration limit	2147483647
barmaxcor	maximum correction limit	-1
barobjrng	maximum objective function	1.0e+20
barorder	row ordering algorithm selection	0
barqcpepcomp	convergence tolerance for the barrier optimizer for QCPs	1.0e-07
barstartalg	barrier starting point algorithm	1

5.29.3.7 GAMS/CPLEX: Sifting Specific Options

Option	Description	Default
siftalg	sifting subproblem algorithm	0
siftitlim	limit on sifting iterations	2147483647

5.29.3.8 GAMS/CPLEX: MIP Algorithmic Options

Option	Description	Default
bbinterval	best bound interval	7
bendersstrategy	Benders decomposition algorithm as a strategy	0
bndstrenind	bound strengthening	-1
bqpcuts	boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality	0

Option	Description	Default
brdir	set branching direction	0
bttol	backtracking limit	1.0
cliques	clique cut generation	0
covers	cover cut generation	0
cutlo	lower cutoff for tree search	-1.0e+75
cuts	default cut generation	0
cutsfactor	cut limit	-1.0
cutup	upper cutoff for tree search	1.0e+75
disjcuts	disjunctive cuts generation	0
divetype	MIP dive strategy	0
eachcutlim	sets a limit for each type of cut	2100000000
flowcovers	flow cover cut generation	0
flowpaths	flow path cut generation	0
fpheur	feasibility pump heuristic	0
fraccuts	Gomory fractional cut generation	0
gubcovers	GUB cover cut generation	0
heurfreq	heuristic frequency	0
heuristiceffort	the effort that CPLEX spends on heuristics during a MIP solve	1.0
implbd	implied bound cut generation	0
.lazy	Lazy constraints activation	0
lazyconstraints	Indicator to use lazy constraints	0
lbheur	local branching heuristic	0
liftprojcuts	lift-and-project cuts	0
localimplied	generation of locally valid implied bound cuts	0
mcfcuts	multi-commodity flow cut generation	0
mipemphasis	MIP solution tactics	0
mipkappastats	MIP kappa computation	0
mipordind	priority list on/off	1
mipordtype	priority order generation	0
mipsearch	search strategy for mixed integer programs	0
mipstart	use mip starting values	0
miqcpstrat	MIQCP relaxation choice	0
mircuts	mixed integer rounding cut generation	0

Option	Description	Default
multimipstart	use multiple mipstarts provided via gdx files	
nodecuts	decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree	0
nodefileind	node storage file indicator	1
nodesel	node selection strategy	1
preslvnd	node presolve selector	0
probe	perform probing before solving a MIP	0
qpmakepsdind	adjust MIQP formulation to make the quadratic matrix positive-semi-definite	1
repeatpresolve	reapply presolve at root after preprocessing	-1
rinsheur	relaxation induced neighborhood search frequency	0
rltcuts	Reformulation Linearization Technique (RLT) cuts	0
solvefinal	switch to solve the problem with fixed discrete variables	1
sos1reform	automatic logarithmic reformulation of special ordered sets of type 1 (SOS1)	0
sos2reform	automatic logarithmic reformulation of special ordered sets of type 2 (SOS2)	0
startalg	MIP starting algorithm	0
strongcandlim	size of the candidates list for strong branching	10
strongitlim	limit on iterations per branch for strong branching	0
subalg	algorithm for subproblems	0
submipnodelim	limit on number of nodes in an RINS subMIP	500
submipscale	scale the problem matrix when CPLEX solves a subMIP during MIP optimization	0
submipstartalg	starting algorithm for a subMIP of a MIP	0
submipsubalg	algorithm for subproblems of a subMIP of a MIP	0
symmetry	symmetry breaking cuts	-1
varsel	variable selection strategy at each node	0
workeralgorithm	set method for optimizing benders subproblems	0
zerohalfcuts	zero-half cuts	0

5.29.3.9 GAMS/CPLEX: MIP Limit Options

Option	Description	Default
aggcutlim	aggregation limit for cut generation	3
auxrootthreads	number of threads for auxiliary tasks at the root node	0
cutpass	maximum number of cutting plane passes	0

Option	Description	Default
fraccand	candidate limit for generating Gomory fractional cuts	200
fracpass	maximum number of passes for generating Gomory fractional cuts	0
intsollim	maximum number of integer solutions	2147483647
nodelim	maximum number of nodes to solve	2147483647
polishafterdettime	deterministic time before starting to polish a feasible solution	1.0e+75
polishafterepagap	absolute MIP gap before starting to polish a feasible solution	0.0
polishafterepgap	relative MIP gap before starting to polish a solution	0.0
polishafterintsol	MIP integer solutions to find before starting to polish a feasible solution	2147483647
polishafternode	nodes to process before starting to polish a feasible solution	2147483647
polishaftertime	time before starting to polish a feasible solution	1.0e+75
probedettime	deterministic time spent probing	1.0e+75
probetime	time spent probing	1.0e+75
repairtries	try to repair infeasible MIP start	0
trelim	maximum space in memory for tree	1.0e+75

5.29.3.10 GAMS/CPLEX: MIP Solution Pool Options

Option	Description	Default
.divflt	solution pool range filter coefficients	0
divfltlo	lower bound on diversity	mindouble
divfltup	upper bound on diversity	maxdouble
populatelim	limit of solutions generated for the solution pool by populate method	20
readflt	reads Cplex solution pool filter file	
solnpool	solution pool file name	
solnpoolagap	absolute tolerance for the solutions in the solution pool	1.0e+75
solnpoolcapacity	limits of solutions kept in the solution pool	2100000000
solnpoolgap	relative tolerance for the solutions in the solution pool	1.0e+75
solnpoolintensity	solution pool intensity for ability to produce multiple solutions	0
solnpoolmerge	solution pool file name for merged solutions	
solnpoolnumsym	maximum number of variable symbols when writing merged solutions	10
solnpoolprefix	file name prefix for GDX solution files	soln
solnpoolreplace	strategy for replacing a solution in the solution pool	0

5.29.3.11 GAMS/CPLEX: MIP Tolerance Options

Option	Description	Default
bendersfeascuttol	Tolerance for whether a feasibility cut has been violated in Benders decomposition	1.0e-06
bendersoptcuttol	Tolerance for optimality cuts in Benders decomposition	1.0e-06
epagap	absolute stopping tolerance	1.0e-06
epgap	relative stopping tolerance	1.0e-04
epint	integrality tolerance	1.0e-05
eplin	degree of tolerance used in linearization	0.001
objdif	overrides GAMS Cheat parameter	0.0
relobjdif	relative cheat parameter	0.0

5.29.3.12 GAMS/CPLEX: Output Options

Option	Description	Default
bardisplay	progress display level	1
clonelog	enable clone logs	0
mipdisplay	progress display level	2
mipinterval	progress display interval	0
mpslongnum	MPS file format precision of numeric output	1
netdisplay	network display level	2
quality	write solution quality statistics	0
siftdisplay	sifting display level	1
simdisplay	simplex display level	1
writeannotation	produce a Cplex annotation file	
writebas	produce a Cplex basis file	
writeflt	produce a Cplex solution pool filter file	
writelp	produce a Cplex LP file	
writemps	produce a Cplex MPS file	
writemst	produce a Cplex mst file	
writeord	produce a Cplex ord file	
writeparam	produce a Cplex parameter file with all active options	
writepre	produce a Cplex LP/MPS/SAV file of the presolved problem	
writeprob	produce a Cplex problem file and inferrs the type from the extension	
writesav	produce a Cplex binary problem file	

5.29.4 Parallel execution using multiple threads

Both ODHeuristicMethods `STANDALONE` and `ODH-CPLEX` can use multiple simultaneous threads. `ODH-CPLEX` must use separate threads for the main CPLEX solve and for the ODHeuristics engine. The `STANDALONE` just uses the ODHeuristics engine which may use multiple simultaneous threads. Thus the processing capability of multi-core hardware can be exploited effectively.

GAMS/ODHCPLEX will ignore the GAMS threads parameter and use its own default. The default ODHeuristic method (i.e. `ODH-CPLEX`) requires multiple threads to works and with the GAMS threads default of 1 this will not work.

Whilst there are good defaults for allocating available threads, it may be worthwhile to give some attention to the allocation of threads between the main CPLEX solver and the ODHeuristics engine for ODH-CPLEX and STANDALONE.

If the option ODHThreads is set to n, n threads are allocated in total, otherwise the total number of threads allocated for both ODH-CPLEX and STANDALONE is set to the number of physical processors available on the computer. If the ODHThreads option is set to a number greater than the number of available processors, multiple threads will have to share the same processor, which may severely degrade performance.

In general, the more threads allocated to the main CPLEX solver, the faster it will run, and similarly, the more allocated to the ODHeuristics engine, the faster it will run. The best balance depends on the model being solved and whether it is intended to run to optimality or to an optimality gap of (say) 0.05 or 0.1. If the GAMS/Cplex Threads is not set, ODH-CPLEX defaults to allocating a quarter of the threads to the ODHeuristics engine and the remainder to the main CPLEX solve. Otherwise it allocates the specified number of threads to the main CPLEX solver and the remainder to the ODHeuristics engine.

Within the ODHeuristics engine, the principal heuristic algorithm can run in parallel on multiple threads. Each algorithmic thread uses CPLEX to solve sub-models and each such instance of CPLEX can itself run on multiple threads. So some attention needs to be given to the allocation of threads between them. If SUB_CPX_THREADS is not set, the CPLEX solver will use one thread for each available logical processor to solve the sub-models. This means that only one thread will be available for the solution improvement heuristic. If the option SUB_CPX_THREADS is set, then by default the heuristic engine sets its number of algorithmic threads to

```
number_of_available_processors / SUB_CPX_THREADS
```

where number_of_available_processors is: the number of logical processors for STANDALONE; and for ODH-CPLEX it is this number less those allocated to the main CPLEX solver.

Many Intel and compatible processors support hyperthreading (where this is enabled on the computer and operating system) and if so there will be two logical processors for every physical core. Using them can severely degrade performance, so if they are enabled it is often a good idea to set ODHThreads to the number of physical processors. Note that on machines with a large number of processors (cores), the principal bottleneck for large scale optimization is usually memory access. In practice it is often better to use only about half of the available cores on (say) a 24 core Intel Xeon system. This is model dependent and some experimentation is worthwhile.

Although the operating system's scheduler usually allocates threads to logical processors so that they run on separate physical cores where possible, it will have more threads to manage than those of the heuristic or CPLEX and so will change this allocation as the heuristic and CPLEX run so as to balance its workload effectively. There is a performance penalty to doing this from the perspective of the heuristic run time. For the ODHeuristics STANDALONE, this can be avoided by locking the heuristic threads to specific processors by setting the heuristic option parameter ProcessorLock to 1. It is not supported for ODH-CPLEX. Under Windows, beware that the threads need to be locked at an above normal priority so this may have a negative impact on other programs concurrently running on the machine.

5.29.5 Determinism

Many users require that repeated runs of their applications under the same conditions give the same results, albeit in slightly variable times. The heuristic runs in this way by default. However, there is a performance penalty that has to be paid for synchronizing the threads. On average, performance can be considerably improved performance at the expense of non-repeatable execution by setting the heuristic option parameter Deterministic to 0. This is often preferred by users with particularly large and difficult models.

5.29.6 Detailed Descriptions of ODHCLPEX Options

addcuts *(integer)*: Indicator for adding cuts from CPLEX master solve ↩

Default: `2`

value	meaning
`0`	Do not add cuts
`1`	Add cuts from CPLEX master solve at root
`2`	Add cuts from CPLEX master solve at root and in tree

advind *(integer)*: advanced basis use ↩

Default: `1`

aggcutlim *(integer)*: aggregation limit for cut generation ↩

Default: `3`

aggfill *(integer)*: aggregator fill parameter ↩

Default: `10`

aggind *(integer)*: aggregator on/off ↩

Default: `-1`

auxrootthreads *(integer)*: number of threads for auxiliary tasks at the root node ↩

Default: `0`

baralg *(integer)*: algorithm selection ↩

Default: `0`

barcolnz *(integer)*: dense column handling ↩

Default: `0`

barcrossalg *(integer)*: barrier crossover method ↩

Default: `0`

bardisplay *(integer)*: progress display level ↩

Default: `1`

barepcomp *(real)*: convergence tolerance ↩

Default: `1.0e-08`

bargrowth *(real)*: unbounded face detection ↩

Default: `1.0e+12`

baritlim *(integer)*: iteration limit ↩

Default: `2147483647`

barmaxcor *(integer)*: maximum correction limit ↩

Default: `-1`

barobjrng *(real)*: maximum objective function ↩

Default: `1.0e+20`

barorder *(integer)*: row ordering algorithm selection ↩

Default: `0`

barqcpepcomp *(real)*: convergence tolerance for the barrier optimizer for QCPs ↩

Default: `1.0e-07`

barstartalg *(integer)*: barrier starting point algorithm ↩

Default: `1`

bbinterval *(integer)*: best bound interval ↩

Default: `7`

bendersfeascuttol *(real)*: Tolerance for whether a feasibility cut has been violated in Benders decomposition ↩

Default: `1.0e-06`

bendersoptcuttol *(real)*: Tolerance for optimality cuts in Benders decomposition ↩

Default: `1.0e-06`

bendersstrategy *(integer)*: Benders decomposition algorithm as a strategy ↩

Default: `0`

bndstrenind *(integer)*: bound strengthening ↩

Default: `-1`

bqpcuts *(integer)*: boolean quadric polytope cuts for nonconvex QP or MIQP solved to global optimality ↩

Default: `0`

brdir *(integer)*: set branching direction ↩

Default: `0`

bttol *(real)*: backtracking limit ↩

Default: `1.0`

calcqcpduals *(integer)*: calculate the dual values of a quadratically constrained problem ↩

Default: `1`

cliques *(integer)*: clique cut generation ↩

Default: `0`

clocktype *(integer)*: clock type for computation time ↩

Default: `2`

clonelog *(integer)*: enable clone logs ↩

Default: `0`

coeredind *(integer)*: coefficient reduction on/off ↩

Default: `-1`

conflictalg *(integer)*: algorithm CPLEX uses in the conflict refiner to discover a minimal set of conflicting constraints in an infeasible model ↩

Default: `0`

conflictdisplay *(integer)*: decides how much information CPLEX reports when the conflict refiner is working ↩

Default: `1`

covers *(integer)*: cover cut generation ↩

Default: `0`

cpumask *(string)*: switch and mask to bind threads to processors (Linux only) ↩

Default: `auto`

cpxpresolve *(integer)*: Applies CPLEX presolve to full model ↩

Default: `0`

value	meaning
`-1`	Apply only for first feasible heuristic
`-2`	Do not apply at all
`0`	Automatically determined
`1`	Always applied

craind *(integer)*: crash strategy (used to obtain starting basis) ↩

Default: `1`

cutlo *(real)*: lower cutoff for tree search ↩

Default: `-1.0e+75`

cutoff *(real)*: GMO cutoff ↩

Default: `0.0`

cutpass *(integer)*: maximum number of cutting plane passes ↩

Default: `0`

cuts *(string)*: default cut generation ↩

Default: `0`

cutsfactor *(real)*: cut limit ↩

Default: `-1.0`

cutup *(real)*: upper cutoff for tree search ↩

Default: `1.0e+75`

datacheck *(integer)*: controls data consistency checking and modeling assistance ↩

Default: `0`

decompdensity *(real)*: Matrix density above which automatic decomposition assigns each variable to a separate key ↩

Default: `0.3`

decomposition *(integer)*: Model decomposition method. ↩

Default: `-1`

value	meaning
-1	Automatically determined
0	Assign each variable to a separate key
1	Use automatic decomosition method
2	Use decomposition based on dot option .key

depind *(integer)*: dependency checker on/off ↩

Default: `-1`

deterministic *(boolean)*: Specifies whether the solution improvement heuristic is run in deterministic or opportunistic (i.e. non-deterministic) mode ↩

Default: `1`

value	meaning
0	Opportunistic
1	Deterministic

dettilim *(real)*: deterministic time limit ↩

Default: 1.0e+75

disjcuts *(integer)*: disjunctive cuts generation ↩

Default: 0

divetype *(integer)*: MIP dive strategy ↩

Default: 0

.divflt *(real)*: solution pool range filter coefficients ↩

Default: 0

divfltlo *(real)*: lower bound on diversity ↩

Default: mindouble

divfltup *(real)*: upper bound on diversity ↩

Default: maxdouble

divisor *(integer)*: Initial divisor for sub models ↩

Initial sub model size is model_size times 1 over Divisor. Ignored if InterDiv is set.

Default: automatic

domlim *(integer)*: domain violation number ↩

Default: 0

dpriind *(integer)*: dual simplex pricing ↩

Default: 0

dynamicrows *(integer)*: switch for dynamic management of rows ↩

Default: -1

dynamicsearch *(integer)*: Search strategy for CPLEX caller and sub-model solves ↩

Default: -1

value	meaning
-1	Automatically determined
0	Use traditional branch & cut
1	Use dynamic search

eachcutlim *(integer)*: sets a limit for each type of cut ↩

Default: `2100000000`

epagap *(real)*: absolute stopping tolerance ↩

Synonym: optca

Default: `1.0e-06`

epgap *(real)*: relative stopping tolerance ↩

Synonym: optcr

Default: `1.0e-04`

epint *(real)*: integrality tolerance ↩

Default: `1.0e-05`

eplin *(real)*: degree of tolerance used in linearization ↩

Default: `0.001`

epmrk *(real)*: Markowitz pivot tolerance ↩

Default: `0.01`

epopt *(real)*: optimality tolerance ↩

Default: `1.0e-06`

epper *(real)*: perturbation constant ↩

Default: `1.0e-06`

eprelax *(real)*: relaxation for feasOpt ↩

Default: `1.0e-06`

eprhs *(real)*: feasibility tolerance ↩

Default: `1.0e-06`

extracplexlog *(boolean)*: Write addition CPLEX output to log ↩

Default: `0`

value	meaning
0	Do not write extra CPLEX informtion
1	Write extra CPLEX information

feasoptmode *(integer)*: mode of FeasOpt ↩

Default: `0`

feastol *(real)*: Feasibility tolerance ↩

Range: [0, 1.0]

Default: 1e-6

firstfeas *(integer)*: Use first feasible heuristic for finding an initial feasible solution ↩

The default for ODH-CPLEX is 1 while for the heuristic engine the default is -1.

Default: -1

value	meaning
-1	Do not use
0	Use if no solution found during initial presolve
1	Always use

firstfeascontinue *(integer)*: Whether first feasible heuristic continues when it achieves feasibility ↩

Default: 0

value	meaning
0	Do not continue
1	Continue
2	Use in sub-model solves

firstfeaseffort *(integer)*: Effort limit on first feasible heuristic ↩

If the option value is positive the exact value is used as the level of effort. If the value is negative no more than the absolute value of the option is used as the level of effort. The larger the effort level, the more effort is expended before giving up.

Default: -500

value	meaning
>0	Use this level of effort
<0	Use no more than -firstfeaseffort effort

firstfeaslpitlim *(integer)*: Limit on number of relaxed re-solves in first feasible heuristic ↩

Default: 10

value	meaning
-1	No limit
0	No relaxed re-solves
>0	Limit

firstfeasrelaxcrit *(real)*: Smallest sum of infeasibilites/row where relaxed solution used ↩

Default: 0.01

firstfeasshift *(integer)*: First feasible heuristic variable shifting in found solutions ↩

Default: `0`

value	meaning
0	Do not shift
1	Moderate shifting
2	Aggressive shifting

fixoptfile *(string)*: name of option file which is read just before solving the fixed problem ↩

flowcovers *(integer)*: flow cover cut generation ↩

Default: 0

flowpaths *(integer)*: flow path cut generation ↩

Default: 0

folding *(integer)*: LP folding will be attempted during the preprocessing phase ↩

Default: -1

fpheur *(integer)*: feasibility pump heuristic ↩

Default: 0

fraccand *(integer)*: candidate limit for generating Gomory fractional cuts ↩

Default: 200

fraccuts *(integer)*: Gomory fractional cut generation ↩

Default: 0

fracpass *(integer)*: maximum number of passes for generating Gomory fractional cuts ↩

Default: 0

freegamsmodel *(boolean)*: preserves memory by dumping the GAMS model instance representation temporarily to disk ↩

Default: 0

globalbounds *(integer)*: Use of global bounds from CPLEX caller ↩

Default: 2

value	meaning
0	Never use
5	Always use
1-4	Intensity of use

gubcovers *(integer)*: GUB cover cut generation ↩

Default: 0

heurfreq *(integer)*: heuristic frequency ↩

Default: `0`

heuristiceffort *(real)*: the effort that CPLEX spends on heuristics during a MIP solve ↩

Default: `1.0`

ignoresetslvrparams *(boolean)*: Flag to control whether solver parameters can be dynamically altered by ODH ↩

Default: `0`

implbd *(integer)*: implied bound cut generation ↩

Default: `0`

indexkey *(string)*: Pattern used to match variable names for grouping into sub-models discussed above ↩

indicoptstrict *(boolean)*: abort in case of an error in indicator constraint in solver option file ↩

Default: `1`

integeronly *(integer)*: Variables to include in INDEXKEY ↩

Default: `-1`

value	meaning
`-1`	Automatically determined
`0`	All variables
`1`	Only non-continuous variables

integertol *(real)*: Integrality tolerance for variable values ↩

Default: `1e-5`

interdiv *(integer)*: Initial divisor value ↩

Default: `4`

intsollim *(integer)*: maximum number of integer solutions ↩

Default: `2147483647`

itlim *(integer)*: iteration limit ↩

Synonym: iterlim

Default: `2147483647`

.key *(integer)*: Variable block or key number ↩

Default: `0`

keypartition *(integer)*: Use of solver partition information ↩

Default: `0`

value	meaning
0	Do not use
1	Use solver partition information to generate keys
2	Use keys to generate sub-solver partition information

keysminimum *(integer)*: Minimum number of keys that the automatic decomposition method attempts to find ↩

Default: 512

keytype *(boolean)*: Treatment of multiple INDEXKEYs ↩

Default: 0

value	meaning
0	Considered separately e.g. INDEXKEY=x_d;y_d means x_2 and y_2 belong to separate groups
1	Considered together e.g. INDEXKEY=x_d;y_d means x_2 and y_2 belong to the same group

.lazy *(boolean)*: Lazy constraints activation ↩

Default: 0

lazyconstraints *(boolean)*: Indicator to use lazy constraints ↩

Default: 0

lbheur *(boolean)*: local branching heuristic ↩

Default: 0

liftprojcuts *(integer)*: lift-and-project cuts ↩

Default: 0

localimplied *(integer)*: generation of locally valid implied bound cuts ↩

Default: 0

localsearch *(boolean)*: Indicator for local search heuristic ↩

Default: 1

loosefeastol *(real)*: Loose feasibility tolerance ↩

Default: 1e-5

lpmethod *(integer)*: algorithm to be used for LP problems ↩

Default: 0

maxbacktrack *(integer)*: The maximum number of backtracks permitted in sub-model solves ↩

Default: -1

value	meaning
-1	Automatically determined
0	Infinite
>0	Use this value if a better solution is available

maxbound *(real)*: The largest(smallest) non infinite bound value ODH will accept for upper(lower) bounds ↩

If this value is positive, bounds exceeding MAXBOUND are reduced to MAXBOUND; if this value is negative, bounds exceeding -MAXBOUND are ignored.

Default: 1e+9

maxinfrepeat *(integer)*: Maximum divisor value when solution is infeasible ↩

Default: 0

value	meaning
0	Automatically determined
>0	Use this value

maxinterdiv *(integer)*: Maximum divisor value ↩

Default: 0

maxrepeat *(integer)*: Maximum number of sub-model repeat solves for each divisor value ↩

Default: 0

value	meaning
0	Automatically determined
>0	Use this value

mcfcuts *(integer)*: multi-commodity flow cut generation ↩

Default: 0

memoryemphasis *(boolean)*: reduces use of memory ↩

Default: 0

mipdisplay *(integer)*: progress display level ↩

Default: 2

mipemphasis *(integer)*: MIP solution tactics ↩

Default: 0

mipinterval *(integer)*: progress display interval ↩

Default: 0

mipkappastats *(integer)*: MIP kappa computation ↩

Default: `0`

mipordind *(boolean)*: priority list on/off ↩

Synonym: prioropt

Default: `1`

mipordtype *(integer)*: priority order generation ↩

Default: `0`

mipsearch *(integer)*: search strategy for mixed integer programs ↩

Default: `0`

mipstart *(integer)*: use mip starting values ↩

Default: `0`

miqcpstrat *(integer)*: MIQCP relaxation choice ↩

Default: `0`

mircuts *(integer)*: mixed integer rounding cut generation ↩

Default: `0`

mpslongnum *(boolean)*: MPS file format precision of numeric output ↩

Default: `1`

multimipstart *(string)*: use multiple mipstarts provided via gdx files ↩

multobjdisplay *(integer)*: level of display during multiobjective optimization ↩

Default: `1`

names *(boolean)*: load GAMS names into Cplex ↩

Default: `1`

netdisplay *(integer)*: network display level ↩

Default: `2`

netepopt *(real)*: optimality tolerance for the network simplex method ↩

Default: `1.0e-06`

neteprhs *(real)*: feasibility tolerance for the network simplex method ↩

Default: `1.0e-06`

netfind *(integer)*: attempt network extraction ↩

Default: 2

netitlim *(integer)*: iteration limit for network simplex ↩

Default: 2147483647

netppriind *(integer)*: network simplex pricing ↩

Default: 0

nodecuts *(integer)*: decide whether or not cutting planes are separated at the nodes of the branch-and-bound tree ↩

Default: 0

nodefileind *(integer)*: node storage file indicator ↩

Default: 1

nodelim *(integer)*: maximum number of nodes to solve ↩

Synonym: nodlim

Default: 2147483647

nodesel *(integer)*: node selection strategy ↩

Default: 1

numericalemphasis *(boolean)*: emphasizes precision in numerically unstable or difficult problems ↩

Default: 0

objdif *(real)*: overrides GAMS Cheat parameter ↩

Synonym: cheat

Default: 0.0

objllim *(real)*: objective function lower limit ↩

Default: -1.0e+75

objtarget *(real)*: Target objective value ↩

ODHeuristics terminates when this value is reached. Defaults to -infinity for minimization or +infinity for maximization models.

Default: 0

objthreshold *(real)*: Threshold for absolute value of objective coefficients ↩

Variables with coefficients greater than this value are always considered for optimization.

Default: 1e99

objulim *(real)*: objective function upper limit ↩

Default: 1.0e+75

odheuristicmethod *(string)*: ODHeuristic method section ↩

Default: ODH-CPLEX

value	meaning
ODH-CPLEX	ODHeuristic within the CPLEX optimizer
STANDALONE	Stand-alone ODHeuristic engine

odhfeasopt *(boolean)*: Optimization method for sub-models in phaseI ↩

Default: 0

odhthreads *(integer)*: The number of heuristic threads used by ODH-CPLEX or STANDALONE ↩

Default: -1

value	meaning
-1	Automatically determined
0	Run in serial mode
>0	Use the specified number of threads

odhtimelimit *(real)*: Elapsed time limit in seconds ↩

Synonym: reslim

Default: GAMS ResLim

optimalitytarget *(integer)*: type of optimality that Cplex targets ↩

Default: 0

parallelmode *(integer)*: parallel optimization mode ↩

Default: 0

paramdisplay *(boolean)*: display the nondefault parameters before optimization ↩

Default: 1

penalty *(real)*: The objective function coefficient value for penalties ↩

The objective function coefficient value for the penalties introduced to deal with infeasibilities in the solution improvement heuristic. It is set by default when required and if not specified.

Default: -1

penperturb *(real)*: Perturbation tolerance for penalties coefficients ↩

Default: 0

perind *(boolean)*: force initial perturbation ↩

Default: 0

perlim *(integer)*: number of stalled iterations before perturbation ↩

Default: 0

phase12 *(boolean)*: Specifies whether to use a phaseI/phaseII method to remove infeasibilities ↩

Default: 1

value	meaning
0	Use composite objective method
1	Use phaseI/phaseII method

polishafterdettime *(real)*: deterministic time before starting to polish a feasible solution ↩

Default: 1.0e+75

polishafterepagap *(real)*: absolute MIP gap before starting to polish a feasible solution ↩

Default: 0.0

polishafterepgap *(real)*: relative MIP gap before starting to polish a solution ↩

Default: 0.0

polishafterintsol *(integer)*: MIP integer solutions to find before starting to polish a feasible solution ↩

Default: 2147483647

polishafternode *(integer)*: nodes to process before starting to polish a feasible solution ↩

Default: 2147483647

polishaftertime *(real)*: time before starting to polish a feasible solution ↩

Default: 1.0e+75

populatelim *(integer)*: limit of solutions generated for the solution pool by populate method ↩

Default: 20

ppriind *(integer)*: primal simplex pricing ↩

Default: 0

predual *(integer)*: give dual problem to the optimizer ↩

Default: 0

preind *(boolean)*: turn presolver on/off ↩

Default: 1

prelinear *(boolean)*: linear reduction indicator ↩

Default: 1

prepass *(integer)*: number of presolve applications to perform ↩

Default: -1

prereform *(integer)*: set presolve reformulations ↩

Default: 3

preslvnd *(integer)*: node presolve selector ↩

Default: 0

pricelim *(integer)*: pricing candidate list ↩

Default: 0

printoptions *(boolean)*: list values of all options to GAMS listing file ↩

Default: 0

probe *(integer)*: perform probing before solving a MIP ↩

Default: 0

probedettime *(real)*: deterministic time spent probing ↩

Default: 1.0e+75

probetime *(real)*: time spent probing ↩

Default: 1.0e+75

processorlock *(boolean)*: Thread allocation ↩

Default: 0

value	meaning
0	Do not lock threads to processors
1	Lock threads to processors

qextractalg *(integer)*: quadratic extraction algorithm in GAMS interface ↩

Default: 0

qpmakepsdind *(boolean)*: adjust MIQP formulation to make the quadratic matrix positive-semi-definite ↩

Default: 1

qpmethod *(integer)*: algorithm to be used for QP problems ↩

Default: 0

qtolin *(integer)*: linearization of the quadratic terms in the objective function of a QP or MIQP model ↩

Default: -1

quality *(boolean)*: write solution quality statistics ↩

Default: 0

quickfirstsolve *(boolean)*: Accelerate initial CPLEX solve ↩

Default: 0

value	meaning
0	Do not unless presolve applied to full model
1	Use existing presolved model

randomseed *(integer)*: sets the random seed differently for diversity of solutions ↩

Default: 202009243

readflt *(string)*: reads Cplex solution pool filter file ↩

record *(boolean)*: Records invocations of Callable Library routines ↩

Default: 0

recurse *(integer)*: Recurse using heuristic to solve sub-models when a feasible solution has been obtained ↩

Default: 0

value	meaning
0	Do not recurse
1	Recurse thread 0 only
2	Recurse odd numbered threads
3	Recurse all threads
<0	Recurse when working with an infeasible solution. values are negated (e.g. -3 = recursion of all threads)

recursedecomp *(integer)*: Recursed model decomposition method ↩

Default: 0

value	meaning
-1	Use initial model decomposition
0	Assign each variable to a separate key
1	Use automatic decomposition method

recurseiterlim *(integer)*: Recursed heuristic iteration limit for sub-solves ↩

Default: 40

recurselog *(boolean)*: Write thread log files for recursed sub-solves ↩

Default: 0

recurseminiterlim *(integer)*: Recursed heuristic minimum iterations before a solution is found in sub-solves ↩

Default: 10

recursesoliterlim *(integer)*: Recursed heuristic sub-solves quit after these iterations if a solution is found ↩

Default: `maxint`

reduce *(integer)*: primal and dual reduction type ↩

Default: `3`

reinv *(integer)*: refactorization frequency ↩

Default: `0`

rejectinfsol *(integer)*: Reject infeasible solutions to sub-models ↩

Default: `2`

value	meaning
0	Do not check feasibility or reject
1	Check feasibility and warn if infeasible, but accept
2	Check feasibility and reject if infeasible

relaxfixedinfeas *(boolean)*: accept small infeasibilties in the solve of the fixed problem ↩

Default: `0`

relaxpreind *(integer)*: presolve for initial relaxation on/off ↩

Default: `-1`

relaxsos2 *(integer)*: Treatment of SOS2 members ↩

Default: `1`

value	meaning
0	Aggressive use in reducing sub-model size
1	Moderate use in reducing sub-model size
2	Ignored in sub-model creation

relobjdif *(real)*: relative cheat parameter ↩

Default: `0.0`

repairtries *(integer)*: try to repair infeasible MIP start ↩

Default: `0`

repeatpresolve *(integer)*: reapply presolve at root after preprocessing ↩

Default: `-1`

rerun *(string)*: rerun problem if presolve infeasible or unbounded ↩

Default: `nono`

rinsheur *(integer)*: relaxation induced neighborhood search frequency ↩

Default: `0`

rltcuts *(integer)*: Reformulation Linearization Technique (RLT) cuts ↩

Default: `0`

scaind *(integer)*: matrix scaling on/off ↩

Default: `0`

seed *(integer)*: Initial random number seed ↩

Default: `12345`

siftalg *(integer)*: sifting subproblem algorithm ↩

Default: `0`

siftdisplay *(integer)*: sifting display level ↩

Default: `1`

sifting *(boolean)*: switch for sifting from simplex optimization ↩

Default: `1`

siftitlim *(integer)*: limit on sifting iterations ↩

Default: `2147483647`

simdisplay *(integer)*: simplex display level ↩

Default: `1`

singlim *(integer)*: limit on singularity repairs ↩

Default: `10`

solnpool *(string)*: solution pool file name ↩

solnpoolagap *(real)*: absolute tolerance for the solutions in the solution pool ↩

Default: `1.0e+75`

solnpoolcapacity *(integer)*: limits of solutions kept in the solution pool ↩

Default: `2100000000`

solnpoolgap *(real)*: relative tolerance for the solutions in the solution pool ↩

Default: `1.0e+75`

solnpoolintensity *(integer)*: solution pool intensity for ability to produce multiple solutions ↩

Default: `0`

solnpoolmerge *(string)*: solution pool file name for merged solutions ↩

solnpoolnumsym *(integer)*: maximum number of variable symbols when writing merged solutions ↩

Default: `10`

solnpoolprefix *(string)*: file name prefix for GDX solution files ↩

Default: `soln`

solnpoolreplace *(integer)*: strategy for replacing a solution in the solution pool ↩

Default: `0`

solutiontype *(integer)*: type of solution (basic or non basic) for an LP or QP ↩

Default: `0`

solvefinal *(boolean)*: switch to solve the problem with fixed discrete variables ↩

Default: `1`

sos1reform *(integer)*: automatic logarithmic reformulation of special ordered sets of type 1 (SOS1) ↩

Default: `0`

sos2reform *(integer)*: automatic logarithmic reformulation of special ordered sets of type 2 (SOS2) ↩

Default: `0`

sosfind *(boolean)*: Automatic detection of Special Ordered Sets (SOSs) ↩

Default: `0`

sosinkey *(integer)*: Assign each SOS to its own sub-model component (key) ↩

Default: `0`

value	meaning
`0`	Treat SOS members as normal i.e. in keys iff declared non-continuous or integeronly =0;
`1`	SOS members always in keys
`2`	SOS members always in keys and members of each SOS to their own key

sosmember *(string)*: Automatically detect SOSs whose variable member names match this pattern ↩

sosovar *(string)*: Automatically detect SOSs whose output variable name matches this pattern ↩

sosselect *(string)*: Select sub-set of SOS members ↩

sosselect16 *(string)*: Select sub-set of SOS members for sets with 16 or fewer members only ↩

soswvar *(string)*: Automatically detect SOSs whose input(weight) variable name matches this pattern ↩

startalg *(integer)*: MIP starting algorithm ↩

Default: 0

strategy *(integer)*: ODH-Cplex Strategy ↩

The aggressive setting attempt to make more progress with each sub-model solve at the cost of more expensive sub solves. Amongst other changes, it sets InterDiv, MaxInterDiv and MaxRepeat if they are not explicitly set by the user.

Default: 1

value	meaning
0	Conservative
1	Normal
2	Aggressiv

strictdeterministic *(integer)*: Terminate ODH deterministically when improvement heuristic finishes ↩

Default: 0

value	meaning
0	Terminate ODH as soon as possible which can violate determinism
1	Terminate ODH deterministically

strongcandlim *(integer)*: size of the candidates list for strong branching ↩

Default: 10

strongitlim *(integer)*: limit on iterations per branch for strong branching ↩

Default: 0

subalg *(integer)*: algorithm for subproblems ↩

Default: 0

subcheckfreq *(integer)*: Frequency with which sub-model LPs are interrupted for mutual communication ↩

Default: 10

value	meaning
0	Do not interrupt sub-model LPs
>0	Interrupt sub-model LPs. The smaller this value then more often they are interrupted.

submipnodelim *(integer)*: limit on number of nodes in an RINS subMIP ↩

Default: 500

submipscale *(integer)*: scale the problem matrix when CPLEX solves a subMIP during MIP optimization ↩

Default: 0

submipstartalg *(integer)*: starting algorithm for a subMIP of a MIP ↩

Default: 0

submipsubalg *(integer)*: algorithm for subproblems of a subMIP of a MIP ↩

Default: 0

subnodelimit *(integer)*: Node limit for submodel searches ↩

Default: -1

value	meaning
-1	Automatically determined
>0	Set node limit to this value

suborder *(integer)*: Use of priority order in sub-solves ↩

Default: 1

value	meaning
0	Do not use any supplied priority order information in sub-solves
1	Use any supplied priority order information in sub-solves

sub_cpx_threads *(integer)*: Threads availble for the solves within ODHeuristic ↩

Default: 1

symmetry *(integer)*: symmetry breaking cuts ↩

Default: -1

syncfreq *(integer)*: Thread synchronization frequency in deterministic parallel mode ↩

Default: 1

value	meaning
0	Low frequency
1	High frequency

threadlog *(boolean)*: Write thread log files ↩

Default: 0

threads *(integer)*: global default thread count ↩

Synonym: gthreads

Default: `0`

threadzerosync *(integer)*: Which CPLEX threads to use for synchronization ↩

Default: `0`

value	meaning
0	Synchronize with multiple CPLEX threads
1	Only synchronize with CPLEX thread 0

tightenprebounds *(integer)*: Level of bound tightening in ODH presolved model ↩

Default: `1`

value	meaning
0	Do not tighten bounds
1	Least aggressive
2	Moderate aggressive
3	Most aggressive

tilim *(real)*: overrides the GAMS ResLim option ↩

Default: `1.0e+75`

trelim *(real)*: maximum space in memory for tree ↩

Default: `1.0e+75`

trialbound *(integer)*: Trial bound heuristic ↩

Default: `0`

value	meaning
-1	As option 1 plus retain trial bounds for whole optimization
-2	As option 2 plus retain trial bounds for whole optimization
0	Read trial bounds from trialboundfile if it is specified
1	Read trial bounds from trialboundfile
2	Auto generate trial bounds and write them to trialboundfile if it is specified

trialboundfile *(string)*: The trial bound file ↩

Bounds are used to attempt finding an initial feasible solution

trialboundsetsize *(integer)*: Size adjustment to automatically generated trial bound sets ↩

Default: `0`

value	meaning
0	Do not adjust set size
>0	Use no more than this number of set members
<0	Decrease set size by this value

tryint *(real)*: GMO tryint ↩

Default: 0.0

tuningdettilim *(real)*: tuning deterministic time limit per model or suite ↩

Default: 1.0e+75

tuningdisplay *(integer)*: level of information reported by the tuning tool ↩

Default: 1

tuningmeasure *(integer)*: measure for evaluating progress for a suite of models ↩

Default: 1

tuningrepeat *(integer)*: number of times tuning is to be repeated on perturbed versions ↩

Default: 1

tuningtilim *(real)*: tuning time limit per model or suite ↩

Default: 1.0e+75

usebasis *(boolean)*: GMO usebasis ↩

Default: 0

usehistory *(integer)*: Use of past sub-model selections in current selection ↩

Default: -1

value	meaning
-1	Automatically determined
0	Never use historical information
1	Always use historical information

varcleanlpmethod *(integer)*: Method used to solve variable cleaning LPs ↩

Default: -1

value	meaning
-1	Automatically determined by solver
0	Primal simplex
1	Dual simplex
2	Barrier

variableclean *(integer)*: Clean variable values from sub-models ↩

Default: `1`

value	meaning
0	No cleaning
1	Quick cleaning and allow feasible uncleaned solutions if unable to clean
2	Quick cleaning and disallow uncleaned solutions
3	Thorough cleaning

varsel *(integer)*: variable selection strategy at each node ↩

Default: `0`

warninglimit *(integer)*: determines how many times warnings of a specific type (datacheck=2) will be displayed ↩

Default: `10`

workdir *(string)*: directory for working files ↩

Default: `.`

workeralgorithm *(integer)*: set method for optimizing benders subproblems ↩

Default: `0`

workmem *(real)*: memory available for working storage ↩

Default: `2048.0`

writeannotation *(string)*: produce a Cplex annotation file ↩

writebas *(string)*: produce a Cplex basis file ↩

writeflt *(string)*: produce a Cplex solution pool filter file ↩

writelp *(string)*: produce a Cplex LP file ↩

writemps *(string)*: produce a Cplex MPS file ↩

writemst *(string)*: produce a Cplex mst file ↩

writeord *(string)*: produce a Cplex ord file ↩

writeparam *(string)*: produce a Cplex parameter file with all active options ↩

writepre *(string)*: produce a Cplex LP/MPS/SAV file of the presolved problem ↩

writeprob *(string)*: produce a Cplex problem file and inferrs the type from the extension ↩

writesav *(string)*: produce a Cplex binary problem file ↩

zerohalfcuts *(integer)*: zero-half cuts ↩

Default: `0`

zerotol *(real)*: Zero tolerance for variable values ↩

Default: `1e-9`

5.30 PATHNLP

5.30.1 Introduction

This document describes the GAMS/PATHNLP solver for non-linear programs and the options unique to this solver.

PATHNLP solves an NLP by internally constructing the Karush-Kuhn-Tucker (KKT) system of first-order optimality conditions associated with the NLP and solving this system using the PATH solver for complementarity problems. The solution to the original NLP is extracted from the KKT solution and returned to GAMS. All of this takes place automatically - no special syntax or user reformulation is required.

Typically, PATHNLP works very well for convex models. It also has a comparative advantage on models whose solution via reduced gradient methods results in a large number of superbasic variables, since the PATH solver won't construct a dense reduced Hessian in the space of the superbasic variables as reduced gradient solvers do. For nonconvex models, however, PATHNLP is not as robust as the reduced gradient methods.

The theory relating NLP to their KKT systems is well-known: assuming differentiability without convexity, and assuming a constraint qualification holds, then a solution to the NLP must also be a solution to the KKT system. If we also assume convexity, then a solution to the KKT system is also a solution to the NLP - no further constraint qualification is required.

In case PATH fails to find a solution to the KKT system for the NLP, a phase I / phase II method is used in which the phase I objective is simply the feasibility error and the original objective is ignored. If a feasible point is found in phase I then phase II, an attempt to solve the KKT system for the NLP using the current feasible point, is entered.

PATHNLP is installed automatically with your GAMS system. With a demo or community license, it will solve small models only). If your GAMS license includes PATH, this size restriction is removed.

5.30.2 Usage

If you have installed the system and configured PATHNLP as the default NLP solver, all NLP models without a specific solver option will be solved with PATHNLP. If you installed another solver as the default, you can explicitly request that a particular model be solved using PATHNLP by inserting the statement

```
option NLP = pathnlp;
```

somewhere before the `solve` statement. Similar comments hold for the other model types (LP, RMINLP, QCP, etc.) PATHNLP can handle.

The standard GAMS model options `iterlim` and `reslim` can be used to control PATHNLP. A description of these options can be found in the GAMS Options section of the chapter on basic solver usage. In general this is enough to use PATHNLP effectively. In some cases, however, you may want to use some of the PATH or PATHNLP options to gain further performance improvements or for other reasons. The rules for using an option file are described in the chapter on basic solver usage. The options used to control PATH can also be used to control PATHNLP. There are also some options unique to PATHNLP.

5.30.3 Options

The tables that follow describe the options unique to PATHNLP as well as the options shared with the PATH solver for MCP models.

5.30.3.1 General options

Option	Description	Default
chen_lambda	lambda parameter for Chen-Chen-Kanzow residual Range: [0, 1]	0.8
convergence_tolerance	stopping criterion	1e-6
crash_iteration_limit	maximum iterations allowed in crash	50
crash_merit_function	merit function used in crash method `normal`: Use the normal map `fischer`: Use the Fischer function	fischer
crash_method	pnewton or none `pnewton`: Use projected Newton method `none`	pnewton
crash_minimum_dimension	minimum problem dimension to perform crash	1
crash_nbchange_limit	number of changes to the basis allowed	1
crash_perturb	perturb the problem using pnewton crash	1
crash_searchtype	search type to use in the crash method `line`: Use a linesearch `arc`: Use an arcsearch	line
cumulative_iteration_limit	maximum minor iterations allowed	10000
gradient_searchtype	search type to use on a gradient step `line`: Use a linesearch `arc`: Use an arcsearch	arc
gradient_step_limit	gradient steps allowed before restarting	5
interrupt_limit	ctrl-C's required before killing job Range: {1, ..., ∞}	5

Option	Description	Default
major_iteration_limit	maximum major iterations allowed	500
merit_function	merit function to use (normal or fischer) `normal`: Use the normal map `fischer`: Use the Fischer function	fischer
minor_iteration_limit	minor iterations allowed in each major iteration	1000
nms	allow line searching, watch-dogging, and nonmonotone descent	1
nms_initial_reference_factor	controls size of initial reference value	20
nms_maximum_watchdogs	maximum number of watchdog steps allowed	5
nms_memory_size	number of reference values kept	10
nms_mstep_frequency	frequency at which m-steps are performed	10
nms_searchtype	search type to use `line`: Use a linesearch `arc`: Use an arcsearch	line
option_file	option file name for PATHLIB to read	
preprocess	turns preprocessing on/off	1
proximal_perturbation	initial perturbation	0
time_limit	number of seconds algorithm is allowed to run	
lemke_rank_deficiency_iterations	number of attempts made to fix rank-deficient basis during Lemke start	10
lemke_start	frequency of lemke starts `always`: Use a Lemke start for each LCP subproblem `automatic`: Determined by algorithm `first`: Use a Lemke start for the first LCP subproblem	automatic
lemke_start_type	type of lemke start `advanced`: Start Lemke method using an advanced basis `slack`: Start Lemke method using and all-slack basis	slack

5.30.3.2 NLP-specific options

Option	Description	Default
allow_reform	substitute out objective var and equ when possible Many models have an objective variable and equation that can be substituted out of the model, e.g. `f(x) =E= z;` If this option is true, PATHNLP will substitute out the objective variable and equation where possible.	1
gmo_hess_factor	maximum multiples of Jacobian size to allow Hessian storage: 0=no limit	0
nlp_lambda	linesearch factor when using the NLP objective If `nlp_objective` is true and `nlp_lambda` is positive, the PATH linesearch will be altered to take the objective function into account.	0
nlp_objective	treat NLP objective differently in PATH linesearch	0
output_memory	output breakdown of where memory is used	0

Option	Description	Default
skip_kkt	go right to Phase I / Phase II method If true, PATHNLP will skip the initial attempt to solve the KKT system for the NLP and go directly into a Phase I / Phase II method that first attempts to get feasible and then attempts to solve the KKT system starting from the feasible point found in Phase I.	0

5.30.3.3 Output options

Option	Description	Default
output_crash_iterations	output information on crash iterations	1
output_crash_iterations_frequency	frequency at which crash iteration log is printed Range: {1, ..., ∞}	1
output_errors	output error messages	1
output_final_degeneracy_statistics	print information regarding degeneracy at the solution	0
output_final_point	output final point returned from PATH	0
output_final_point_statistics	output information about the point, function, and Jacobian at the final point	1
output_final_scaling_statistics	display matrix norms on the Jacobian at the final point	0
output_final_statistics	output evaluation of available merit functions at the final point	1
output_final_summary	output summary information	1
output_initial_point	output initial point given to PATH	0
output_initial_point_statistics	output information about the point, function, and Jacobian at the initial point	1
output_initial_scaling_statistics	display matrix norms on the Jacobian at the initial point	1
output_initial_statistics	output evaluation of available merit functions at the initial point	0
output_linear_model	output linear model at each major iteration	0
output_major_iterations	output information on major iterations	1
output_major_iterations_frequency	frequency at which major iteration log is printed Range: {1, ..., ∞}	1
output_maximum_zero_listing	limits zero columns reported to listing file	1000
output_maximum_zero_log	limits zero columns reported to log file	10
output_minor_iterations	output information on minor iterations	1
output_minor_iterations_frequency	frequency at which minor iteration log is printed Range: {1, ..., ∞}	500
output_options	output all options and their values	0
output	no turns all output off	1
output_preprocess_level	control output of preprocessing information	1
output_restart_log	output options during restarts	1

Option	Description	Default
output_time	output breakdown of where time is spent	0
output_warnings	output warning messages	0

5.31 PATH

Author

Michael C. Ferris

Todd S. Munson

5.31.1 Complementarity

A fundamental problem of mathematics is to find a solution to a square system of nonlinear equations. Two generalizations of nonlinear equations have been developed, a constrained nonlinear system which incorporates bounds on the variables, and the complementarity problem (MCP). This document is primarily concerned with the complementarity problem.

The PATH solver for MCP models is a Newton-based solver that combines a number of the most effective variations, extensions, and enhancements of this powerful technique. See PATH vs MILES for a comparison with MILES. Algorithmic details can also be found in papers and technical reports by Dirkse, Ferris, and Munson on Ferris' Home Page.

The complementarity problem adds a combinatorial twist to the classic square system of nonlinear equations, thus enabling a broader range of situations to be modeled. In its simplest form, the combinatorial problem is to choose from $2n$ inequalities a subset of n that will be satisfied as equations. These problems arise in a variety of disciplines including engineering and economics [60] where we might want to compute Wardropian and Walrasian equilibria, and optimization where we can model the first order optimality conditions for nonlinear programs [102] [114] . Other examples, such as bimatrix games [117] and options pricing [97] , abound.

Our development of complementarity is done by example. We begin by looking at the optimality conditions for a transportation problem and some extensions leading to the nonlinear complementarity problem. We then discuss a Walrasian equilibrium model and use it to motivate the more general mixed complementarity problem. We conclude this chapter with information on solving the models using the PATH solver and interpreting the results.

5.31.1.1 Transportation Problem

The transportation model is a linear program where demand for a single commodity must be satisfied by suppliers at minimal transportation cost. The underlying transportation network is given as a set $\mathcal{A}$ of arcs, where $(i,j) \in \mathcal{A}$ means that there is a route from supplier i to demand center j. The problem variables are the quantities $x_{i,j}$ shipped over each arc $(i,j) \in \mathcal{A}$. The linear program can be written mathematically as

$$\begin{array}{rl} \min_{x \geq 0} & \sum_{(i,j)\in\mathcal{A}} c_{i,j} x_{i,j} \\ \text{subject to} & \sum_{j:(i,j)\in\mathcal{A}} x_{i,j} \leq s_i, \ \forall i \\ & \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, \ \forall j. \end{array} \tag{1}$$

where $c_{i,j}$ is the unit shipment cost on the arc (i,j), s_i is the available supply at i, and d_j is the demand at j.

The derivation of the optimality conditions for this linear program begins by associating with each constraint a multiplier, alternatively termed a dual variable or shadow price. These multipliers represent the marginal price on changes to the corresponding constraint. We label the prices on the supply constraint p^s and those on the demand constraint p^d. Intuitively, for each supply node i

$$0 \leq p_i^s, \; s_i \geq \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}.$$

Consider the case when $s_i > \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}$, that is there is excess supply at i. Then, in a competitive marketplace, no rational person is willing to pay for more supply at node i; it is already over-supplied. Therefore, $p_i^s = 0$. Alternatively, when $s_i = \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}$, that is node i clears, we might be willing to pay for additional supply of the good. Therefore, $p_i^s \geq 0$. We write these two conditions succinctly as:

$$0 \leq p_i^s \quad \perp \quad s_i \geq \textstyle\sum_{j:(i,j)\in\mathcal{A}} x_{i,j}, \quad \forall i$$

where the $\perp$ notation is understood to mean that at least one of the adjacent inequalities must be satisfied as an equality. For example, either $0 = p_i^s$, the first case, or $s_i = \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}$, the second case.

Similarly, at each node j, the demand must be satisfied in any feasible solution, that is

$$\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j.$$

Furthermore, the model assumes all prices are nonnegative, $0 \leq p_j^d$. If there is too much of the commodity supplied, $\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} > d_j$, then, in a competitive marketplace, the price p_j^d will be driven down to 0. Summing these relationships gives the following complementarity condition:

$$0 \leq p_j^d \quad \perp \quad \textstyle\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, \quad \forall j.$$

The supply price at i plus the transportation cost $c_{i,j}$ from i to j must exceed the market price at j. That is, $p_i^s + c_{i,j} \geq p_j^d$. Otherwise, in a competitive marketplace, another producer will replicate supplier i increasing the supply of the good in question which drives down the market price. This chain would repeat until the inequality is satisfied. Furthermore, if the cost of delivery strictly exceeds the market price, that is $p_i^s + c_{i,j} > p_j^d$, then nothing is shipped from i to j because doing so would incur a loss and $x_{i,j} = 0$. Therefore,

$$0 \leq x_{i,j} \quad \perp \quad p_i^s + c_{i,j} \geq p_j^d, \quad \forall (i,j) \in \mathcal{A}.$$

We combine the three conditions into a single problem,

$$\begin{array}{rcll} 0 \leq p_i^s & \perp & s_i \geq \sum_{j:(i,j)\in\mathcal{A}} x_{i,j}, & \forall i \\ 0 \leq p_j^d & \perp & \sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j, & \forall j \\ 0 \leq x_{i,j} & \perp & p_i^s + c_{i,j} \geq p_j^d, & \forall (i,j) \in \mathcal{A}. \end{array} \tag{2}$$

This model defines a linear complementarity problem that is easily recognized as the complementary slackness conditions [37] of the linear program (1). For linear programs the complementary slackness conditions are both necessary and sufficient for x to be an optimal solution of the problem (1). Furthermore, the conditions (2) are also the necessary and sufficient optimality conditions for a related problem in the variables (p^s, p^d)

$$\begin{array}{rl} \max_{p^s,p^d\geq 0} & \sum_j d_j p_j^d - \sum_i s_i p_i^s \\ \text{subject to} & c_{i,j} \geq p_j^d - p_i^s, \; \forall (i,j) \in \mathcal{A} \end{array}$$

termed the dual linear program (hence the nomenclature "dual variables").

Looking at (2) a bit more closely we can gain further insight into complementarity problems. A solution of (2) tells us the arcs used to transport goods. A priori we do not need to specify which arcs to use, the solution itself indicates them. This property represents the key contribution of a complementarity problem over a system of equations. If we know what arcs to send flow down, we can just solve a simple system of linear equations. However, the key to the modeling power of complementarity is that it chooses which of the inequalities in (2) to satisfy as equations. In economics we can use this property to generate a model with different regimes and let the solution determine which ones are active. A regime shift could, for example, be a back stop technology like windmills that become profitable if a CO_2 tax is increased.

GAMS Code

The GAMS code for the complementarity version of the transportation problem is given in Figure 1; the actual data for the model is assumed to be given in the file `transmcp.dat`. Note that the model written corresponds very closely to (2). In GAMS, the $\perp$ sign is replaced in the `model` statement with a ".". It is precisely at this point that the pairing of variables and equations shown in (2) occurs in the GAMS code. For example, the function defined by `rational` is complementary to the variable `x`. To inform a solver of the bounds, the standard GAMS statements on the variables can be used, namely (for a declared variable $z(i)$):

```
z.lo(i) = 0;
```

or alternatively

```
positive variable z;
```

Further information on the GAMS syntax can be found in [155]. *Note that GAMS requires the modeler to write `F(z) =g= 0` whenever the complementary variable is lower bounded, and does not allow the alternative form `0 =l= F(z)`.*

Figure 1: A simple MCP model in GAMS, `transmcp.gms`

```
sets    i   canning plants,
        j   markets ;

parameter
        s(i)    capacity of plant i in cases,
        d(j)    demand at market j in cases,
        c(i,j)  transport cost in thousands of dollars per case ;

$include transmcp.dat

positive variables
        x(i,j)           shipment quantities in cases
        p_demand(j)      price at market j
        p_supply(i)      price at plant i;

equations
        supply(i)      observe supply limit at plant i
        demand(j)      satisfy demand at market j
        rational(i,j);

supply(i) ..     s(i) =g= sum(j, x(i,j)) ;

demand(j) ..     sum(i, x(i,j))  =g=  d(j) ;

rational(i,j) .. p_supply(i) + c(i,j) =g= p_demand(j) ;

model transport / rational.x, demand.p_demand, supply.p_supply /;

solve transport using mcp;
```

Extension: Model Generalization

While many interior point methods for linear programming exploit this complementarity framework (so-called primal-dual methods [205]), the real power of this modeling format is the new problem instances it enables a modeler to create. We now show some examples of how to extend the simple model (2) to investigate other issues and facets of the problem at hand.

Demand in the model of Figure 1 is independent of the prices p. Since the prices p are variables in the complementarity problem (2), we can easily replace the constant demand d with a function $d(p)$ in the complementarity setting. Clearly, any algebraic function of p that can be expressed in GAMS can now be added to the model given in Figure 1. For example, a linear demand function could be expressed using

$$\sum_{i:(i,j)\in\mathcal{A}} x_{i,j} \geq d_j(1 - p_j^d), \ \forall j.$$

Note that the demand is rather strange if p_j^d exceeds 1. Other more reasonable examples for $d(p)$ are easily derived from Cobb-Douglas or CES utilities. For those examples, the resulting complementarity problem becomes nonlinear in the variables p. Details of complementarity for more general transportation models can be found in [48] , [62] .

Another feature that can be added to this model are tariffs or taxes. In the case where a tax is applied at the supply point, the third general inequality in (2) is replaced by

$$p_i^s(1 + t_i) + c_{i,j} \geq p_j^d, \ \forall (i,j) \in \mathcal{A}.$$

The taxes can be made endogenous to the model, details are found in [155] .

The key point is that with either of the above modifications, the complementarity problem is not just the optimality conditions of a linear program. In many cases, there is no optimization problem corresponding to the complementarity conditions.

Nonlinear Complementarity Problem

We now abstract from the particular example to describe more carefully the complementarity problem in its mathematical form. All the above examples can be cast as nonlinear complementarity problems (NCPs) defined as follows:

(NCP) Given a function $F : \mathbf{R}^n \to \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that

$$0 \leq z \perp F(z) \geq 0.$$

Recall that the $\perp$ sign signifies that one of the inequalities is satisfied as an equality, so that componentwise, $z_i F_i(z) = 0$. We frequently refer to this property as z_i is *complementary* to F_i. A special case of the NCP that has received much attention is when F is a linear function, the linear complementarity problem [40] .

5.31.1.2 Walrasian Equilibrium

A Walrasian equilibrium can also be formulated as a complementarity problem (see [132]). In this case, we want to find a price $p \in \mathbf{R}^m$ and an activity level $y \in \mathbf{R}^n$ such that

$$\begin{array}{rcrcl} 0 \leq y & \perp & L(p) & := & -A^T p \geq 0 \\ 0 \leq p & \perp & S(p,y) & := & b + Ay - d(p) \geq 0 \end{array} \tag{3}$$

where $S(p, y)$ represents the excess supply function and $L(p)$ represents the loss function. Complementarity allows us to choose the activities y_j to run (i.e. only those that do not make a loss). The second set of inequalities state that the price of a commodity can only be positive if there is no excess supply. These conditions indeed correspond to the standard exposition of Walras' law which states that supply equals demand if we assume all prices p will be positive at a solution. Formulations of equilibria as systems of equations do not allow the model to choose the activities present, but typically make an a priori assumption on this matter.

GAMS Code

A GAMS implementation of (3) is given in Figure 2. Many large scale models of this nature have been developed. An interested modeler could, for example, see how a large scale complementarity problem was used to quantify the effects of the Uruguay round of talks [95] .

Figure 2: Walrasian equilibrium as an NCP, `walras1.gms`

```
$include walras.dat

positive variables p(i), y(j);
equations S(i), L(j);

S(i)..   b(i) + sum(j, A(i,j)*y(j)) - c(i)*sum(k, g(k)*p(k)) / p(i)
         =g= 0;

L(j)..   -sum(i, p(i)*A(i,j)) =g= 0;

model walras / S.p, L.y /;
solve walras using mcp;
```

Extension: Intermediate Variables

In many modeling situations, a key tool for clarification is the use of intermediate variables. As an example, the modeler may wish to define a variable corresponding to the demand function $d(p)$ in the Walrasian equilibrium (3). The syntax for carrying this out is shown in Figure 3 where we use the variables `d` to store the demand function referred to in the excess supply equation. The model `walras` now contains a mixture of equations and complementarity constraints. Since constructs of this type are prevalent in many practical models, the GAMS syntax allows such formulations.

Figure 3: Walrasian equilibrium as an MCP, `walras2.gms`

```
$include walras.dat

positive variables p(i), y(j);
variables d(i);
equations S(i), L(j), demand(i);

demand(i)..
         d(i) =e= c(i)*sum(k, g(k)*p(k)) / p(i) ;

S(i)..   b(i) + sum(j, A(i,j)*y(j)) - d(i) =g= 0 ;

L(j)..   -sum(i, p(i)*A(i,j)) =g= 0 ;

model walras / demand.d, S.p, L.y /;
solve walras using mcp;
```

Note that positive variables are paired with inequalities, while free variables are paired with equations. A crucial point misunderstood by many modelers (new and experienced alike) is that *the bounds on the variable determine the relationships satisfied by the function F*. Thus in Figure 3, d is a free variable and therefore its paired equation `demand` is an equality. Similarly, since p is nonnegative, its paired relationship `S` is a (greater-than) inequality. See the MCP definition below for details.

A simplification is allowed to the model statement in Figure 3. It is not required to match free variables explicitly to equations; we only require that there are the same number of free columns (i.e. single variables) as unmatched rows (i.e. single equations). Thus, in the example of Figure 3, the model statement could be replaced by

```
model walras / demand, S.p, L.y /;
```

This extension allows existing GAMS models consisting of a square system of nonlinear equations to be easily recast as a complementarity problem - the model statement is unchanged. GAMS generates a list of all variables appearing in the equations found in the model statement, performs explicitly defined pairings and then checks that the number of remaining equality rows equals the number of remaining free columns. However, if an explicit match is given, the PATH solver can frequently exploit the information for better solution. Note that all variables that are not free and all inequalities must be explicitly matched.

Mixed Complementarity Problem

A mixed complementarity problem (MCP) is specified by three pieces of data, namely the lower bounds ℓ, the upper bounds u and the function F.

(MCP) Given lower bounds $\ell \in \{\mathbf{R} \cup \{-\infty\}\}^n$, upper bounds $u \in \{\mathbf{R} \cup \{\infty\}\}^n$ and a function $F : \mathbf{R}^n \to \mathbf{R}^n$, find $z \in \mathbf{R}^n$ such that *precisely* one of the following holds for each $i \in \{1, \ldots, n\}$:

$$\begin{array}{lll} F_i(z) = 0 & \text{and} & \ell_i \leq z_i \leq u_i \\ F_i(z) > 0 & \text{and} & z_i = \ell_i \\ F_i(z) < 0 & \text{and} & z_i = u_i. \end{array}$$

These relationships define a general MCP (sometimes termed a rectangular variational inequality [94]). We will write these conditions compactly as

$$\ell \leq x \leq u \quad \perp \quad F(x).$$

As the MCP model type as defined above is the one used in GAMS, there are some consequences of the definition worth emphasizing here. Firstly, the bounds on the variable determine the relationships satisfied by the function F: the constraint type (e.g. `=G=`, `=L=`) chosen plays no role in defining the problem. This being the case, it is acceptable (and perhaps advisable!) to use an `=N=` in the definition of all equations used in complementarity models to highlight the dependence on the variable bounds in defining the MCP. This is not required, however. The constraint types `=G=`, `=L=` and `=E=` are also allowed. If the variable bounds are inconsistent with the constraint type chosen, GAMS will either reject the model (e.g. an `=L=` row matched with a lower-bounded variable) or silently and temporarily ignore the issue (e.g. an `=E=` row matched with a lower-bounded variable). In the latter case, if equality does not hold at solution (allowable if the variable is at lower bound) the is marked in the listing file with a redef.

Secondly, note the contrast between *complementarity conditions* like $g(z) \leq 0, z \geq 0$ and a *complementarity problem* like $F(x) \perp 0 \leq x \leq \infty$. The former requires a trivial transformation in order to fit into the MCP framework: $-g(z) \perp 0 \leq z \leq \infty$. This simple issue is a frequent source of problems and worth looking out for.

The nonlinear complementarity problem of Nonlinear Complementarity Problem is a special case of the MCP. For example, to formulate an NCP in the GAMS/MCP format we set

```
z.lo(I) = 0;
```

or declare

```
positive variable z;
```

Another special case is a square system of nonlinear equations

(NE) Given a function $F : \mathbf{R}^n \to \mathbf{R}^n$ find $z \in \mathbf{R}^n$ such that

$$F(z) = 0.$$

In order to formulate this in the GAMS/MCP format we just declare

```
free variable z;
```

In both the above cases, we *must not* modify the lower and upper bounds on the variables later (unless we wish to drastically change the problem under consideration).

An advantage of the extended formulation described above is the pairing between "fixed" variables (ones with equal upper and lower bounds) and a component of F. If a variable z_i is fixed, then $F_i(z)$ is unrestricted since precisely one of the three conditions in the MCP definition automatically holds when $z_i = \ell_i = u_i$. Thus if a variable is fixed in a GAMS model, the paired equation is completely dropped from the model. This convenient modeling trick can be used to remove particular constraints from a model at generation time. As an example, in economics, fixing a level of production will remove the zero-profit condition for that activity.

Simple bounds on the variables are a convenient modeling tool that translates into efficient mathematical programming tools. For example, specialized codes exist for the bound constrained optimization problem

$$\min f(x) \text{ subject to } \ell \leq x \leq u.$$

The first order optimality conditions for this problem class are precisely MCP($\nabla f(x), [\ell, u]$). We can easily see this condition in a one dimensional setting. If we are at an unconstrained stationary point, then $\nabla f(x) = 0$. Otherwise, if x is at its lower bound, then the function must be increasing as x increases, so $\nabla f(x) \geq 0$. Conversely, if x is at its upper bound, then the function must be increasing as x decreases, so that $\nabla f(x) \leq 0$. The MCP allows such problems to be easily and efficiently processed.

Upper bounds can be used to extend the utility of existing models. For example, in Figure 3 it may be necessary to have an upper bound on the activity level y. In this case, we simply add an upper bound to y in the model statement, and we can replace the loss equation with the following definition:

```
y.up(j) = 10;
L(j)..   -sum(i, p(i)*A(i,j)) =e= 0 ;
```

Here, for bounded variables, we do not know beforehand if the constraint will be satisfied as an equation, less-than inequality or greater-than inequality, since this determination depends on the values of the solution variables. However, let us interpret the relationships that the above change generates. If $y_j = 0$, the loss function can be positive since we are not producing in the jth sector. If y_j is strictly between its bounds, then the loss function must be zero by the definition of complementarity; this is the competitive assumption. Finally, if y_j is at its upper bound, then the loss function can be negative. Of course, if the market does not allow free entry, some firms may operate at a profit (negative loss). For more examples of problems, the interested reader is referred to [45] , [59] , [60] .

5.31.1.3 Solution

We will assume that a file named `transmcp.gms` has been created using the GAMS syntax which defines an MCP model `transport` as developed in Transportation Problem . The modeler has a choice of the complementarity solver to use. We are going to further assume that the modeler wants to use PATH.

There are two ways to ensure that PATH is used as opposed to any other GAMS/MCP solver. These are as follows:

1. Add the following line to the `transmcp.gms` file prior to the `solve` statement

   ```
   option mcp = path;
   ```

   PATH will then be used instead of the default solver provided.

2. Choose PATH as the default solver for MCP (via the IDE on Windows or elsewhere by rerunning `gamsinst` from the GAMS system directory).

To solve the problem, the modeler executes the command:

```
gams transmcp
```

where `transmcp` can be replaced by any filename containing a GAMS model. Many other command line options for GAMS exist; the reader is referred to List of Command Line Parameters for further details.

At this stage, control is handed over to the solver which creates a log providing information on what the solver is doing as time elapses. See Section PATH for details about the log file. After the solver terminates, a listing file is generated containing the solution to the problem. We now describe the listing file output specifically related to complementarity problems.

Listing File

The listing file is the standard GAMS mechanism for reporting model results. This file contains information regarding the compilation process, the form of the generated equations in the model, and a report from the solver regarding the solution process.

We now detail the last part of this output, an example of which is given in Figure 4 . We use "..." to indicate where we have omitted continuing similar output.

Figure 4: Listing File for solving `transmcp.gms`

```
               S O L V E      S U M M A R Y

     MODEL   TRANSPORT
     TYPE    MCP
     SOLVER  PATH                FROM LINE  45

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL

 RESOURCE USAGE, LIMIT          0.057     1000.000
 ITERATION COUNT, LIMIT        31        10000
 EVALUATION ERRORS              0            0
```

```
 Work space allocated             --    0.06 Mb


---- EQU RATIONAL

                       LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york    -0.225    -0.225      +INF      50.000
seattle  .chicago     -0.153    -0.153      +INF     300.000
seattle  .topeka      -0.162    -0.126      +INF        .

...

---- VAR X           shipment quantities in cases

                       LOWER     LEVEL     UPPER    MARGINAL

seattle  .new-york       .       50.000     +INF        .
seattle  .chicago        .      300.000     +INF        .

...

**** REPORT SUMMARY :         0     NONOPT
                              0 INFEASIBLE
                              0  UNBOUNDED
                              0  REDEFINED
                              0     ERRORS
```

After a summary line indicating the model name and type and the solver name, the listing file shows a solver status and a model status. Table 1 and Table 2 display the relevant codes that are returned under different circumstances. A modeler can access these codes within the `transmcp.gms` file using `transport.solveStat` and `transport.modelStat` respectively.

Table 1: Solver Status Codes

Code	String	Meaning
1	Normal completion	Solver finished normally
2	Iteration interrupt	Solver reached the iterations limit
3	Resource interrupt	Solver reached the time limit
4	Terminated by solver	Solver reached an unspecified limit
8	User interrupt	The user interrupted the solution process

Table 2: Model Status Codes

Code	String	Meaning
1	Optimal	Solver found a solution of the problem
6	Intermediate infeasible	Solver failed to solve the problem

After this, a listing of the time and iterations used is given, along with a count on the number of evaluation errors encountered. If the number of evaluation errors is greater than zero, further information can typically be found later in the listing file, prefaced by "****". Information provided by the solver is then displayed.

Next comes the solution listing, starting with each of the equations in the model. For each equation passed to the solver, four columns are reported, namely the lower bound, level, upper bound and marginal.

GAMS moves all parts of a constraint involving variables to the left hand side, and accumulates the constants on the right hand side. The lower and upper bounds correspond to the constants that GAMS generates. For equations, these should be equal, whereas for inequalities one of them should be infinite. The level value of the equation (an evaluation of the left hand side of the constraint at the current point) should be between these bounds, otherwise the solution is infeasible and the equation is marked as follows:

```
seattle  .chicago     -0.153    -2.000     +INF    300.000 INFES
```

The marginal column in the equation contains the value of the variable that was matched with this equation.

For the variable listing, the lower, level and upper columns indicate the lower and upper bounds on the variables and the solution value. The level value returned by PATH will always be between these bounds. The marginal column contains the value of the slack on the equation that was paired with this variable. If a variable appears in one of the constraints in the model statement but is not explicitly paired to a constraint, the slack reported here contains the internally matched constraint slack. The definition of this slack is the minimum of equ.l - equ.lower and equ.l - equ.upper, where equ is the paired equation.

Finally, a summary report is given that indicates how many errors were found. Figure 4 is nicely-solved case; when the model has infeasibilities, these can be found by searching for the string "INFES" as described above.

Redefined Equations

Unfortunately, this is not the end of the story. Some equations may have the following form:

```
                LOWER      LEVEL      UPPER    MARGINAL

new-york    325.000    350.000    325.000    0.225 REDEF
```

This should be construed as a warning from GAMS, as opposed to an error. In principle, the `REDEF` should only occur if the paired variable to the constraint has a finite lower and/or upper bound and the variable is at one of those bounds. In this case, at the solution of the complementarity problem the sense of the constraint (e.g. `=E=`) may not be satisfied. This occurs when GAMS constraints are used to define the function F. The bounds on each component of the function F are derived from the bounds on the matching variable component, and these bounds may be inconsistent with those implied by the constraint type used. GAMS warns the user via the `REDEF` label when the solution found leads to a violated "constraint". To avoid `REDEF` warnings the `=N=` can be used to define all complementarity functions.

Note that a `REDEF` is not an error, just a warning. The solver has solved the complementarity problem specified. GAMS gives this report to ensure that the modeler understands that the complementarity problem derives the relationships on the equations from the variable bounds, not from the equation definition.

5.31.1.4 Pitfalls

As indicated above, the ordering of an equation is important in the specification of an MCP. Since the data of the MCP is the function F and the bounds ℓ and u, it is important for the modeler to pass the solver the function F and not $-F$.

For example, if we have the optimization problem,

$$\min_{x \in [0,2]} (x-1)^2$$

then the first order optimality conditions are

$$0 \leq x \leq 2 \quad \perp \quad 2(x-1)$$

which has a unique solution, $x = 1$. Figure 5 provides correct GAMS code for this problem.

Figure 5: First order conditions as an MCP, `first.gms`

```
variables x;
equations d_f;

x.lo = 0;
x.up = 2;

d_f.. 2*(x - 1) =e= 0;

model first / d_f.x /;
solve first using mcp;
```

However, if we accidentally write the valid equation

```
d_f.. 0 =e= 2*(x - 1);
```

the problem given to the solver is

$$0 \leq x \leq 2 \quad \perp \quad -2(x-1)$$

which has three solutions, $x = 0$, $x = 1$, and $x = 2$. This problem is in fact the stationary conditions for the nonconvex quadratic problem,

$$\max_{x \in [0,2]} (x-1)^2,$$

not the problem we intended to solve.

Continuing with the example, when x is a free variable, we might conclude that the ordering of the equation is irrelevant because we always have the equation, $2(x-1) = 0$, which does not change under multiplication by -1. In most cases, the ordering of equations (which are complementary to free variables) does not make a difference since the equation is internally "substituted out" of the model. In particular, for defining equations, such as that presented in Figure 3, the choice appears to be arbitrary.

However, in difficult (singular or near singular) cases, the substitution cannot be performed, and instead a perturbation is applied to F, in the hope of "(strongly) convexifying" the problem. If the perturbation happens to be in the wrong direction because F was specified incorrectly, the perturbation actually makes the problem less convex, and hence less likely to solve. Note that determining which of the above orderings of the equations makes most sense is typically tricky. One rule of thumb is to check whether if you replace the "=e=" by "=g=", and then increase "x", is the inequality intuitively more likely to be satisfied. If so, you probably have it the right way round, if not, reorder.

Furthermore, underlying model convexity is important. For example, if we have the linear program

$$\begin{array}{rl} \min_x & c^T x \\ \text{subject to} & Ax = b,\ x \geq 0 \end{array}$$

we can write the first order optimality conditions as either

$$\begin{array}{rcl} 0 \leq x & \perp & -A^T\mu + c \\ \mu \text{ free} & \perp & Ax - b \end{array}$$

or, equivalently,

$$\begin{array}{rcl} 0 \leq x & \perp & -A^T\mu + c \\ \mu \text{ free} & \perp & b - Ax \end{array}$$

because we have an equation. The former is a linear complementarity problem with a positive semidefinite matrix, while the latter is almost certainly indefinite. Also, if we need to perturb the problem because of numerical problems, the former system will become positive definite, while the later becomes highly nonconvex and unlikely to solve. The rule of thumb here is to consider what happens when the dual multiplier μ is made non-negative in the KKT system. In the former system, the constraint becomes $Ax \geq b$, which is consistent with a minimization and a positive multiplier. In the latter system, we get $Ax \leq b$, which is backwards.

Finally, users are strongly encouraged to match equations and free variables when the matching makes sense for their application. Structure and convexity can be destroyed if it is left to the solver to perform the matching. For example, in the above example, we could lose the positive semidefinite matrix with an arbitrary matching of the free variables.

5.31.2 PATH

Newton's method, perhaps the most famous solution technique, has been extensively used in practice to solve square systems of nonlinear equations. The basic idea is to construct a local approximation of the nonlinear equations around a given point, x^k, solve the approximation to find the Newton point, x^N, update the iterate, $x^{k+1} = x^N$, and repeat until we find a solution to the nonlinear system. This method works extremely well close to a solution, but can fail to make progress when started far from a solution. To guarantee progress is made, a line search between x^k and x^N is used to enforce sufficient decrease on an appropriately defined merit function. Typically, $\frac{1}{2}\|F(x)\|^2$ is used.

PATH uses a generalization of this method on a nonsmooth reformulation of the complementarity problem. To construct the Newton direction, we use the normal map [151] representation

$$F(\pi(x)) + x - \pi(x)$$

associated with the MCP, where $\pi(x)$ represents the projection of x onto $[\ell, u]$ in the Euclidean norm. We note that if x is a zero of the normal map, then $\pi(x)$ solves the MCP. At each iteration, a linearization of the normal map, a linear complementarity problem, is solved using a pivotal code related to Lemke's method.

Versions of PATH prior to 4.x are based entirely on this formulation using the residual of the normal map

$$\|F(\pi(x)) + x - \pi(x)\|$$

as a merit function. However, the residual of the normal map is not differentiable, meaning that if a subproblem is not solvable then a "steepest descent" step on this function cannot be taken. PATH 4.x considers an alternative nonsmooth system [63] , $\Phi(x) = 0$, where $\Phi_i(x) = \phi(x_i, F_i(x))$ and $\phi(a, b) := \sqrt{a^2 + b^2} - a - b$. The merit function, $\|\Phi(x)\|^2$, in this case is differentiable, and is used for globalization purposes. When the subproblem solver fails, a projected gradient direction for this merit function is searched. It is shown in [61] that this provides descent and a new feasible point to continue PATH, and convergence to stationary points and/or solutions of the MCP is provided under appropriate conditions.

The remainder of this chapter details the interpretation of output from PATH and ways to modify the behavior of the code. To this end, we will assume that the modeler has created a file named `transmcp.gms` which defines an MCP model `transport` as described in Section Transportation Problem and is using PATH 4.x to solve it. See Section Solution for information on changing the solver.

5.31.2.1 Log File

We will now describe the behavior of the PATH algorithm in terms of the output typically produced. An example of the log for a particular run is given in Figure 6 and Figure 7.

Figure 6: Log File from PATH for solving `transmcp.gms`

```
--- Starting compilation
--- trnsmcp.gms(46) 1 Mb
--- Starting execution
--- trnsmcp.gms(27) 1 Mb
--- Generating model transport
--- trnsmcp.gms(45) 1 Mb
---    11 rows, 11 columns, and 24 non-zeroes.
--- Executing PATH
 Work space allocated           --    0.06 Mb
 Reading the matrix.
 Reading the dictionary.
Path v4.3: GAMS Link ver037, SPARC/SOLARIS
11 row/cols, 35 non-zeros, 28.93% dense.

Path 4.3 (Sat Feb 26 09:38:08 2000)
Written by Todd Munson, Steven Dirkse, and Michael Ferris

INITIAL POINT STATISTICS
Maximum of X. . . . . -0.0000e+00 var: (x.seattle.new-york)
Maximum of F. . . . .  6.0000e+02 eqn: (supply.san-diego)
Maximum of Grad F . .  1.0000e+00 eqn: (demand.new-york)
                                  var: (x.seattle.new-york)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . .  3.0000e+00 eqn: (supply.seattle)
Minimum Row Norm. . .  2.0000e+00 eqn: (rational.seattle.new-york)
Maximum Column Norm .  3.0000e+00 var: (p_supply.seattle)
Minimum Column Norm .  2.0000e+00 var: (x.seattle.new-york)

Crash Log
major  func  diff  size  residual    step       prox   (label)
    0     0             1.0416e+03             0.0e+00 (demand.new-york)
    1     1     3     3 1.0029e+03  1.0e+00    1.0e+01 (demand.new-york)
pn_search terminated: no basis change.
```

Figure 7: Log File from PATH for solving `transmcp.gms` (continued)

```
Major Iteration Log
major minor  func  grad  residual    step  type prox    inorm  (label)
    0     0     2     2 1.0029e+03           I 9.0e+00 6.2e+02 (demand.new-york)
    1     1     3     3 8.3096e+02  1.0e+00 SO 3.6e+00 4.5e+02 (demand.new-york)

...

   15     2    17    17 1.3972e-09  1.0e+00 SO 4.8e-06 1.3e-09 (demand.chicago)

FINAL STATISTICS
Inf-Norm of Complementarity . .  1.4607e-08 eqn: (rational.seattle.chicago)
Inf-Norm of Normal Map. . . . .  1.3247e-09 eqn: (demand.chicago)
```

```
Inf-Norm of Minimum Map . . . .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Fischer Function. .  1.3247e-09 eqn: (demand.chicago)
Inf-Norm of Grad Fischer Fcn. .  1.3247e-09 eqn: (rational.seattle.chicago)

FINAL POINT STATISTICS
Maximum of X. . . . . .  3.0000e+02 var: (x.seattle.chicago)
Maximum of F. . . . . .  5.0000e+01 eqn: (supply.san-diego)
Maximum of Grad F . . . . . .  1.0000e+00 eqn: (demand.new-york)
                                          var: (x.seattle.new-york)

 ** EXIT - solution found.

Major Iterations. . . . 15
Minor Iterations. . . . 31
Restarts. . . . . 0
Crash Iterations. . . . 1
Gradient Steps. . . . . 0
Function Evaluations. . 17
Gradient Evaluations. . 17
Total Time. . . . . 0.020000
Residual. . . . . 1.397183e-09
--- Restarting execution
```

The first few lines on this log file are printed by GAMS during its compilation and generation phases. The model is then passed off to PATH at the stage where the "Executing PATH" line is written out. After some basic memory allocation and problem checking, the PATH solver checks if the modeler required an option file to be read. In the example this is not the case. If PATH is directed to read an option file (see PATH Options below), then the following output is generated after the PATH banner.

```
Reading options file PATH.OPT
 > output_linear_model yes;
Options: Read: Line 2 invalid: hi_there;
Read of options file complete.
```

If the option reader encounters an invalid option (as above), it reports this but carries on executing the algorithm. Following this, the algorithm starts working on the problem.

Diagnostic Information

Some diagnostic information is initially generated by the solver at the starting point. Included is information about the initial point and function evaluation. The log file here tells the value of the largest element of the starting point and the variable where it occurs. Similarly, the maximum function value is displayed along with the row producing it. The maximum element in the gradient is also presented with the equation and variable where it is located.

The second block provides more information about the Jacobian at the starting point. This information can be used to help scale the model. See Section Advanced Topics for complete details.

Crash Log

The first phase of the code is a crash procedure attempting to quickly determine which of the inequalities should be active. This procedure is documented fully in [47] , and an example of the Crash Log can be seen in Figure 6. The first column of the crash log is just a label indicating the current iteration number,

the second gives an indication of how many function evaluations have been performed so far. Note that precisely one Jacobian (gradient) evaluation is performed per crash iteration. The number of changes to the active set between iterations of the crash procedure is shown under the "diff" column. The crash procedure terminates if this becomes small. Each iteration of this procedure involves a factorization of a matrix whose size is shown in the next column. The residual is a measure of how far the current iterate is from satisfying the complementarity conditions (MCP); it is zero at a solution. See Merit Functions for further information. The column "step" corresponds to the steplength taken in this iteration - ideally this should be 1. If the factorization fails, then the matrix is perturbed by an identity matrix scaled by the value indicated in the "prox" column. The "label" column indicates which row in the model is furthest away from satisfying the conditions (MCP). Typically, relatively few crash iterations are performed. Section PATH Options gives mechanisms to affect the behavior of these steps.

Major Iteration Log

After the crash is completed, the main algorithm starts as indicated by the "Major Iteration Log" flag (see Figure 7). The columns that have the same labels as in the crash log have precisely the same meaning described above. However, there are some new columns that we now explain. Each major iteration attempts to solve a linear mixed complementarity problem using a pivotal method that is a generalization of Lemke's method [117] . The number of pivots performed `per` major iteration is given in the "minor" column.

The "grad" column gives the cumulative number of Jacobian evaluations used; typically one evaluation is performed per iteration. The "inorm" column gives the value of the error in satisfying the equation indicated in the "label" column.

At each iteration of the algorithm, several different step types can be taken, due to the use of nonmonotone searches [46] , [56] which are used to improve robustness. In order to help the PATH user, we have added two code letters indicating the return code from the linear solver and the step type to the log file. Table 3 explains the return codes for the linear solver and Table 4 explains the meaning of each step type. The ideal output in this column is "SO"; "SD" and "SB" also indicate reasonable progress. Codes different from these are not catastrophic, but typically indicate the solver is having difficulties due to numerical issues or nonconvexities in the model.

Table 3: Linear Solver Codes

Code	Meaning
C	A cycle was detected.
E	An error occurred in the linear solve.
I	The minor iteration limit was reached.
N	The basis became singular.
R	An unbounded ray was encountered.
S	The linear subproblem was solved.
T	Failed to remain within tolerance after factorization was performed.

Table 4: Step Type Codes

Code	Meaning
B	A Backtracking search was performed from the current iterate to the Newton point in order to obtain sufficient decrease in the merit function.
D	The step was accepted because the Distance between the current iterate and the Newton point was small.
G	A gradient step was performed.
I	Initial information concerning the problem is displayed.

Code	Meaning
M	The step was accepted because the Merit function value is smaller than the nonmonotone reference value.
O	A step that satisfies both the distance and merit function tests.
R	A Restart was carried out.
W	A Watchdog step was performed in which we returned to the last point encountered with a better merit function value than the nonmonotone reference value (M, O, or B step), regenerated the Newton point, andperformed a backtracking search.

Minor Iteration Log

If more than 500 pivots are performed, a minor log is output that gives more details of the status of these pivots. A listing from `transmcp` model follows, where we have set the `output_minor_iteration_frequency` option to 1.

```
Minor Iteration Log
minor      t          z     w     v   art ckpts  enter    leave
    1  4.2538e-01     8     2     0     0     0 t[    0] z[   11]
    2  9.0823e-01     8     2     0     0     0 w[   11] w[   10]
    3  1.0000e+00     9     2     0     0     0 z[   10] t[    0]
```

`t` is a parameter that goes from zero to 1 for normal starts in the pivotal code. When the parameter reaches 1, we are at a solution to the subproblem. The `t` column gives the current value for this parameter. The next columns report the current number of problem variables z and slacks corresponding to variables at lower bound w and at upper bound v. Artificial variables are also noted in the minor log, see [57] for further details. Checkpoints are times where the basis matrix is refactorized. The number of checkpoints is indicated in the `ckpts` column. Finally, the minor iteration log displays the entering and leaving variables during the pivot sequence.

Restart Log

The PATH code attempts to fully utilize the resources provided by the modeler to solve the problem. Versions of PATH after 3.0 have been much more aggressive in determining that a stationary point of the residual function has been encountered. When it is determined that no progress is being made, the problem is restarted from the initial point supplied in the GAMS file with a different set of options. These restarts give the flexibility to change the algorithm in the hopes that the modified algorithm leads to a solution. The ordering and nature of the restarts were determined by empirical evidence based upon tests performed on real-world problems.

The exact options set during the restart are given in the restart log, part of which is reproduced below.

```
Restart Log
proximal_perturbation 0
crash_method none
crash_perturb yes
nms_initial_reference_factor 2
proximal_perturbation 1.0000e-01
```

If a particular problem solves under a restart, a modeler can circumvent the wasted computation by setting the appropriate options as shown in the log. Note that sometimes an option is repeated in this log. In this case, it is the last option that is used.

Solution Log

A solution report is now given by the algorithm for the point returned. The first component is an evaluation of several different merit functions. Next, a display of some statistics concerning the final point is given. This report can be used detect problems with the model and solution as detailed in Section Advanced Topics .

At the end of the log file, summary information regarding the algorithm's performance is given. The string "** EXIT - solution found". is an indication that PATH solved the problem. Any other EXIT string indicates a termination at a point that may not be a solution. These strings give an indication of what `modelStat` and `solveStat` will be returned to GAMS. After this, the "Restarting execution" flag indicates that GAMS has been restarted and is processing the results passed back by PATH.

5.31.2.2 Status File

If for some reason the PATH solver exits without writing a solution, or the `sysout` flag is turned on, the status file generated by the PATH solver will be reported in the listing file. The status file is similar to the log file, but provides more detailed information. The modeler is typically not interested in this output.

5.31.2.3 User Interrupts

A user interrupt can be effected by typing Ctrl-C. We only check for interrupts every major iteration. If a more immediate response is wanted, repeatedly typing Ctrl-C will eventually kill the job. The number needed is controlled by the `interrupt_limit` option. In this latter case, when a kill is performed, no solution is written and an execution error will be generated in GAMS.

5.31.2.4 Preprocessing

The purpose of a preprocessor is to reduce the size and complexity of a model to achieve improved performance by the main algorithm. Another benefit of the analysis performed is the detection of some provably unsolvable problems. A comprehensive preprocessor has been incorporated into PATH as developed in [58] .

The preprocessor reports its finding with some additional output to the log file. This output occurs before the initial point statistics. An example of the preprocessing on the `forcebsm` model is presented below.

```
Zero:     0 Single:   112 Double:     0 Forced:     0
Preprocessed size: 72
```

The preprocessor looks for special polyhedral structure and eliminates variables using this structure. These are indicated with the above line of text. Other special structure is also detected and reported.

On exit from the algorithm, we must generate a solution for the original problem. This is done during the postsolve. Following the postsolve, the residual using the original model is reported.

```
Postsolved residual: 1.0518e-10
```

This number should be approximately the same as the final residual reported on the presolved model.

Constrained Nonlinear System

Modelers typically add bounds to their variables when attempting to solve nonlinear problems in order to restrict the domain of interest. For example, many square nonlinear systems are formulated as

$$F(z) = 0,\ \ell \leq z \leq u,$$

where typically, the bounds on z are inactive at the solution. This is *not* an MCP, but is an example of a "constrained nonlinear system" (CNS). It is important to note the distinction between MCP and CNS. The MCP uses the bounds to infer relationships on the function F. If a finite bound is active at a solution, the corresponding component of F is only constrained to be nonnegative or nonpositive in the MCP, whereas in CNS it must be zero. Thus there may be many solutions of MCP that do not satisfy $F(z) = 0$. Only if z^* is a solution of MCP with $\ell < z^* < u$ is it guaranteed that $F(z^*) = 0$.

Internally, PATH reformulates a constrained nonlinear system of equations to an equivalent complementarity problem. The reformulation adds variables, y, with the resulting problem written as:

$$\begin{array}{rcl} \ell \leq x \leq u & \perp & -y \\ y \text{ free} & \perp & F(x). \end{array}$$

This is the MCP model passed on to the PATH solver.

5.31.3 PATH Options

The default options of PATH should be sufficient for most models. If desired, PATH-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

We give a list of the available PATH options along with their defaults and meaning below. Note that only the first three characters of every word are significant.

5.31.3.1 General options

Option	Description	Default
chen_lambda	lambda parameter for Chen-Chen-Kanzow residual Range: [0, 1]	0.8
convergence_tolerance	stopping criterion When the residual error is within this tolerance, convergence is declared.	1e-6
crash_iteration_limit	maximum iterations allowed in basis crash	50
crash_merit_function	merit function used in crash method normal: Use the normal map fischer: Use the Fischer function	fischer
crash_method	pnewton or none pnewton: Use projected Newton method none	pnewton
crash_minimum_dimension	minimum problem dimension to perform crash	1
crash_nbchange_limit	limit on crash iterations without basis change The basis crashing procedure is stopped if this many successive stagnant iterations occur.	1
crash_perturb	perturb the problem using pnewton crash	1

Option	Description	Default
crash_searchtype	search type to use in the crash method `line`: Use a linesearch `arc`: Use an arcsearch	`line`
cumulative_iteration_limit	maximum minor iterations allowed	`GAMS iterlim`
factorization_library_name	name of factorization library	

factorization_method	basis package to use
	`lusol`
	`blu_lusol`
	`umfpack`
		`lusol`

gradient_searchtype	search type to use on a gradient step `line`: Use a linesearch `arc`: Use an arcsearch	`arc`
gradient_step_limit	gradient steps allowed before restarting	5
interrupt_limit	ctrl-C's required before a hard kill of the solver Range: {1, ..., ∞}	5
major_iteration_limit	maximum major iterations allowed	500
merit_function	merit function to use (normal or fischer) `normal`: Use the normal map `fischer`: Use the Fischer function	`fischer`
minor_iteration_limit	minor iterations allowed in each major iteration: default MIN(2n,iterlim)	`auto`
nms	allow line searching, watch-dogging, and nonmonotone descent	1
nms_initial_reference_factor	controls size of initial reference value	20
nms_maximum_watchdogs	maximum number of watchdog steps allowed	5
nms_memory_size	number of reference values kept	10
nms_mstep_frequency	frequency at which m-steps are performed	10
nms_searchtype	search type to use `line`: Use a linesearch `arc`: Use an arcsearch	`line`
option_file	name of option file for PATHLIB to read If specified, the PATH optimizer will read this file using its internal reader, after the usual options processing is done.	`none`
preprocess	turns preprocessing on/off	1
proximal_perturbation	initial perturbation	0
time_limit	number of seconds algorithm is allowed to run	`GAMS reslim`
lemke_rank_deficiency_iterations	number of attempts made to fix rank-deficient basis during Lemke start	10
lemke_start	frequency of lemke starts `always`: Use a Lemke start for each LCP subproblem `automatic`: Determined by algorithm `first`: Use a Lemke start for the first LCP subproblem	`automatic`

Option	Description	Default
lemke_start_type	type of lemke start `advanced`: Start Lemke method using an advanced basis `slack`: Start Lemke method using an all-slack basis	`slack`

5.31.3.2 Output options

Option	Description	Default
output_crash_iterations	output information on crash iterations	1
output_crash_iterations_frequency	frequency at which crash iteration log is printed Range: {1, ..., ∞}	1
output_errors	output error messages	1
output_final_degeneracy_statistics	print information regarding degeneracy at the solution	0
output_final_point	output final point returned from PATH	0
output_final_point_statistics	output information about the point, function, and Jacobian at the final point	1
output_final_scaling_statistics	display matrix norms on the Jacobian at the final point	0
output_final_statistics	output evaluation of available merit functions at the final point	1
output_final_summary	output summary information	1
output_initial_point	output initial point given to PATH	0
output_initial_point_statistics	output information about the point, function, and Jacobian at the initial point	1
output_initial_scaling_statistics	display matrix norms on the Jacobian at the initial point	1
output_initial_statistics	output evaluation of available merit functions at the initial point	0
output_linear_model	output linear model at each major iteration	0
output_major_iterations	output information on major iterations	1
output_major_iterations_frequency	frequency at which major iteration log is printed Range: {1, ..., ∞}	1
output_maximum_zero_listing	limits zero columns reported to listing file	1000
output_maximum_zero_log	limits zero columns reported to log file	10
output_minor_iterations	output information on minor iterations	1
output_minor_iterations_frequency	frequency at which minor iteration log is printed Range: {1, ..., ∞}	500
output_options	output all options and their values	0
output	turns all output off or on If output is first turned off, selected parts can be turned back on using specific output options.	1
output_preprocess_level	control output of preprocessing information Range: {-1, ..., ∞}	1

Option	Description	Default
output_restart_log	output options during restarts	1
output_time	output breakdown of where time is spent	0
output_warnings	output warning messages	0

GAMS controls the total number of pivots allowed via the `iterlim` option. If more pivots are needed for a particular model then either of the following lines should be added to the `transmcp.gms` file before the solve statement

```
option iterlim = 2000;
transport.iterlim = 2000;
```

Problems with a singular basis matrix can be overcome by using the `proximal_perturbation` option [26] , and linearly dependent columns can be output with the `output_factorization_singularities` option. For more information on singularities, we refer the reader to Section Advanced Topics.

As a special case, PATH can emulate Lemke's method [39] , [117] for LCP with the following options:

```
crash_method none
crash_perturb no
major_iteration_limit 1
lemke_start first
nms no
```

If instead, PATH is to imitate the Successive Linear Complementarity method (SLCP, often called the Josephy Newton method) [101] , [132] , [131] for MCP with an Armijo style linesearch on the normal map residual, then the options to use are:

```
crash_method none
crash_perturb no
lemke_start always
nms_initial_reference_factor 1
nms_memory size 1
nms_mstep_frequency 1
nms_searchtype line
merit_function normal
```

Note that `nms_memory_size 1` and `nms_initial_reference_factor` 1 turn off the nonmonotone linesearch, while `nms_mstep_frequency` 1 turns off watchdogging [35] . `nms_searchtype line` forces PATH to search the line segment between the initial point and the solution to the linear model, while `merit_function normal` tell PATH to use the normal map for calculating the residual.

5.31.4 Advanced Topics

This chapter discusses some of the difficulties encountered when dealing with complementarity problems. We start off with a very formal definition of a complementarity problem (similar to the one given previously) which is used in later sections on merit functions and ill-defined, poorly-scaled, and singular models.

5.31.4.1 Formal Definition of MCP

The mixed complementarity problem is defined by a function, $F : D \to \mathbf{R}^n$ where $D \subseteq \mathbf{R}^n$ is the domain of F, and possibly infinite lower and upper bounds, ℓ and u. Let $C := \{x \in \mathbf{R}^n \mid \ell \leq x \leq u\}$, a Cartesian product of closed (possibly infinite) intervals. The problem is given as

$$MCP : \text{find } x \in C \cap D \text{ s.t. } \langle F(x), y - x\rangle \geq 0, \ \forall y \in C.$$

This formulation is a special case of the variational inequality problem defined by F and a (nonempty, closed, convex) set C. Special choices of ℓ and u lead to the familiar cases of a system of nonlinear equations

$$F(x) = 0$$

(generated by $\ell \equiv -\infty$, $u \equiv +\infty$) and the nonlinear complementarity problem

$$0 \leq x \ \perp \ F(x) \geq 0$$

(generated using $\ell \equiv 0$, $u \equiv +\infty$).

5.31.4.2 Algorithmic Features

We now describe some of the features of the PATH algorithm and the options affecting each.

Merit Functions

A solver for complementarity problems typically employs a merit function to indicate the closeness of the current iterate to the solution set. The merit function is zero at a solution to the original problem and strictly positive otherwise. Numerically, an algorithm terminates when the merit function is approximately equal to zero, thus possibly introducing spurious "solutions".

The modeler needs to be able to determine with some reasonable degree of accuracy whether the algorithm terminated at solution or if it simply obtained a point satisfying the desired tolerances that is not close to the solution set. For complementarity problems, we can provide several indicators with different characteristics to help make such a determination. If one of the indicators is not close to zero, then there is some evidence that the algorithm has not found a solution. We note that if all of the indicators are close to zero, we are reasonably sure we have found a solution. However, the modeler has the final responsibility to evaluate the "solution" and check that it makes sense for their application.

For the NCP, a standard merit function is

$$\|(-x)_+, (-F(x))_+, [(x_i)_+(F_i(x))_+]_i\|$$

with the first two terms measuring the infeasibility of the current point and the last term indicating the complementarity error. In this expression, we use $(\cdot)_+$ to represent the Euclidean projection of x onto the nonnegative orthant, that is $(x)_+ = \max(x, 0)$. For the more general MCP, we can define a similar function:

$$\left\| x - \pi(x), \left[\left(\frac{x_i - \ell_i}{\|\ell_i\| + 1} \right)_+ (F_i(x))_+ \right]_i, \left[\left(\frac{u_i - x_i}{\|u_i\| + 1} \right)_+ (-F_i(x))_+ \right]_i \right\|$$

where $\pi(x)$ represents the Euclidean projection of x onto C. We can see that if we have an NCP, the function is exactly the one previously given and for nonlinear systems of equations, this becomes $\|F(x)\|$.

There are several reformulations of the MCP as a system of nonlinear (nonsmooth) equations for which the corresponding residual is a natural merit function. Some of these are as follows:

- Generalized Minimum Map: $x - \pi(x - F(x))$
- Normal Map: $F(\pi(y)) + y - \pi(y)$
- Fischer Function: $\Phi(x)$, where $\Phi_i(x) := \phi(x_i, F_i(x))$ with

$$\phi(a, b) := \sqrt{a^2 + b^2} - a - b.$$

  Note that $\phi(a, b) = 0$ if and only if $0 \leq a \perp b \geq 0$. A straightforward extension of Φ to the MCP format is given for example in [61] .

In the context of nonlinear complementarity problems the generalized minimum map corresponds to the classic minimum map $\min(x, F(x))$. Furthermore, for NCPs the minimum map and the Fischer function are both local error bounds and were shown to be equivalent in [185] . Figure 10 in the subsequent section plots all of these merit functions for the ill-defined example discussed therein and highlights the differences between them.

The squared norm of Φ, namely $\Psi(x) := \frac{1}{2}\sum \phi(x_i, F_i)^2$, is continuously differentiable on $\mathbf{R}^n$ provided F itself is. Therefore, the first order optimality conditions for the unconstrained minimization of $\Psi(x)$, namely $\nabla\Psi(x) = 0$ give another indication as to whether the point under consideration is a solution of MCP.

The merit functions and the information PATH provides at the solution can be useful for diagnostic purposes. By default, PATH 4.x returns the best point with respect to the merit function because this iterate likely provides better information to the modeler. As detailed in Section PATH Options, the default merit function in PATH 4.x is the Fischer function. To change this behavior the `merit_function` option can be used.

Crashing Method

The crashing technique [47] is used to quickly identify an active set from the user-supplied starting point. At this time, a proximal perturbation scheme [24] , [25] is used to overcome problems with a singular basis matrix. The proximal perturbation is introduced in the crash method, when the matrix factored is determined to be singular. The value of the perturbation is based on the current merit function value.

Even if the crash method is turned off, for example via the option `crash_method none`, perturbation can be added. This is determined by factoring the matrix that crash would have initially formed. This behavior is extremely useful for introducing a perturbation for singular models. It can be turned off by issuing the option `crash_perturb no`.

Nonmontone Searches

The first line of defense against convergence to stationary points is the use of a nonmonotone linesearch [90] , [91] , [56] . In this case we define a reference value, R^k, and we use this value to test for sufficient decrease:

$$\Psi(x^k + t_k d^k) \leq \mathbf{R}^k + t_k \nabla\Psi(x^k)^T d^k.$$

Depending upon the choice of the reference value, this allows the merit function to increase from one iteration to the next. This strategy can not only improve convergence, but can also avoid local minimizers by allowing such increases.

We now need to detail our choice of the reference value. We begin by letting $\{M_1, \ldots, M_m\}$ be a finite set of values initialized to $\kappa\Psi(x^0)$, where κ is used to determine the initial set of acceptable merit function values. The value of κ defaults to 1 in the code and can be modified with the `nms_initial_reference_factor` option; $\kappa = 1$ indicates that we are not going to allow the merit function to increase beyond its initial value.

Having defined the values of $\{M_1, \ldots, M_m\}$ (where the code by default uses $m = 10$), we can now calculate a reference value. We must be careful when we allow gradient steps in the code. Assuming that d^k is the Newton direction, we define $i_0 = \text{argmax } M_i$ and $R^k = M_{i_0}$. After the nonmonotone linesearch rule above finds t_k, we update the memory so that $M_{i_0} = \Psi(x^k + t_k d^k)$, i.e. we remove an element from the memory having the largest merit function value.

When we decide to use a gradient step, it is beneficial to let $x^k = x^{\text{best}}$ where x^{best} is the point with the absolute best merit function value encountered so far. We then recalculate $d^k = -\nabla\Psi(x^k)$ using the best point and let $R^k = \Psi(x^k)$. That is to say that we force decrease from the best iterate found whenever a gradient step is performed. After a successful step we set $M_i = \Psi(x^k + t_k d^k)$ for all $i \in [1, \ldots, m]$. This prevents future iterates from returning to the same problem area.

A watchdog strategy [35] is also available for use in the code.The method employed allows steps to be accepted when they are "close" to the current iterate. Nonmonotonic decrease is enforced every m iterations, where m is set by the `nms_mstep_frequency` option.

Linear Complementarity Problems

PATH solves a linear complementarity problem at each major iteration. Let $M \in \Re^{n\times n}$, $q \in \Re^n$, and $B = [l, u]$ be given. $(\bar{z}, \bar{w}, \bar{v})$ solves the linear mixed complementarity problem defined by M, q, and B if and only if it satisfies the following constrained system of equations:

$$Mz - w + v + q = 0 \tag{4}$$

$$w^T(z - l) = 0 \tag{5}$$

$$v^T(u - z) = 0 \tag{6}$$

$$z \in B, w \in \Re^n_+, v \in \Re^n_+ \tag{7}$$

where $x + \infty = \infty$ for all $x \in \Re$ and $0 \cdot \infty = 0$ by convention. A triple, $(\hat{z}, \hat{w}, \hat{v})$, satisfying equations (4) - (6) is called a complementary triple.

The objective of the linear model solver is to construct a path from a given complementary triple $(\hat{z}, \hat{w}, \hat{v})$ to a solution $(\bar{z}, \bar{w}, \bar{v})$. The algorithm used to solve the linear problem is identical to that given in [44] ; however, artificial variables are incorporated into the model. The augmented system is then:

$$Mz - w + v + Da + \frac{(1-t)}{s}(sr) + q = 0 \tag{8}$$

$$w^T(z - l) = 0 \tag{9}$$

$$v^T(u - z) = 0 \tag{10}$$

$$z \in B, w \in \Re^n_+, v \in \Re^n_+, a \equiv 0, t \in [0, 1] \tag{11}$$

where r is the residual, t is the path parameter, and a is a vector of artificial variables. The residual is scaled by s to improve numerical stability.

The addition of artificial variables enables us to construct an initial invertible basis consistent with the given starting point even under rank deficiency. The procedure consists of two parts: constructing an initial guess as to the basis and then recovering from rank deficiency to obtain an invertible basis. The crash technique gives a good approximation to the active set. The first phase of the algorithm uses this information to construct a basis by partitioning the variables into three sets:

1. $W = \{i \in \{1, \ldots, n\} \mid \hat{z}_i = l_i \text{ and } \hat{w}_i > 0\}$
2. $V = \{i \in \{1, \ldots, n\} \mid \hat{z}_i = u_i \text{ and } \hat{v}_i > 0\}$
3. $Z = \{1, \ldots, n\} \setminus W \cup V$

Since $(\hat{z}, \hat{w}, \hat{v})$ is a complementary triple, $Z \cap W \cap V = \emptyset$ and $Z \cup W \cup V = \{1, \ldots, n\}$. Using the above guess, we can recover an invertible basis consistent with the starting point by defining D appropriately. The technique relies upon the factorization to indicate the linearly dependent rows and columns of the basis matrix. Some of the variables may be nonbasic, but not at their bounds. For such variables, the corresponding artificial will be basic.

We use a modified version of EXPAND [83] to perform the ratio test. Variables are prioritized as follows:

1. t leaving at its upper bound.
2. Any artificial variable.
3. Any z, w, or v variable.

If a choice as to the leaving variable can be made while maintaining numerical stability and sparsity, we choose the variable with the highest priority (lowest number above).

When an artificial variable leaves the basis and a z-type variable enters, we have the choice of either increasing or decreasing that entering variable because it is nonbasic but not at a bound. The determination is made such that t increases and stability is preserved.

If the code is forced to use a ray start at each iteration (`lemke_start always`), then the code carries out Lemke's method, which is known [39] not to cycle. However,by default, we use a regular start to guarantee that the generated path emanates from the current iterate. Under appropriate conditions, this guarantees a decrease in the nonlinear residual. However, it is then possible for the pivot sequence in the linear model to cycle. To prevent this undesirable outcome, we attempt to detect the formation of a cycle with the heuristic that if a variable enters the basis more that a given number of times, we are cycling. The number of times the variable has entered is reset whenever t increases beyond its previous maximum or an artificial variable leaves the basis. If cycling is detected, we terminate the linear solver at the largest value of t and return this point.

Another heuristic is added when the linear code terminates on a ray. The returned point in this case is not the base of the ray. We move a slight distance up the ray and return this new point. If we fail to solve the linear subproblem five times in a row, a Lemke ray start will be performed in an attempt to solve the linear subproblem. Computational experience has shown this to be an effective heuristic and generally results in solving the linear model. Using a Lemke ray start is not the default mode, since typically many more pivots are required.

For times when a Lemke start is actually used in the code, an advanced ray can be used. We basically choose the "closest" extreme point of the polytope and choose a ray in the interior of the normal cone at this point. This helps to reduce the number of pivots required. However, this can fail when the basis corresponding to the cell is not invertible. We then revert to the Lemke start.

Since the EXPAND pivot rules are used, some of the variables may be nonbasic, but slightly infeasible, as the solution. Whenever the linear code finishes, the nonbasic variables are put at their bounds and the basic variables are recomputed using the current factorization. This procedure helps to find the best possible solution to the linear system.

The resulting linear solver as modified above is robust and has the desired property that we start from $(\hat{z}, \hat{w}, \hat{v})$ and construct a path to a solution.

Other Features

Some other heuristics are incorporated into the code. During the first iteration, if the linear solver fails to find a Newton point, a Lemke start is used. Furthermore, under repeated failures during the linear solve, a Lemke start will be attempted. A gradient step can also be used when we fail repeatedly.

The proximal perturbation is reduced at each major iteration. However, when numerical difficulties are encountered, it will be increased to a fraction of the current merit function value. These are determined when the linear solver returns a Reset or Singular status.

Spacer steps are taken every major iteration, in which the iterate is chosen to be the best point for the normal map. The corresponding basis passed into the Lemke code is also updated.

Scaling is done based on the diagonal of the matrix passed into the linear solver.

We finally note, that if the merit function fails to show sufficient decrease over the last 100 iterates, a restart will be performed, as this indicates we are close to a stationary point.

5.31.4.3 Difficult Models

Ill-Defined Models

A problem can be ill-defined for several different reasons. We concentrate on the following particular cases. We will call F well-defined at $\bar{x} \in C$ if $\bar{x} \in D$ and ill-defined at $\bar{x}$ otherwise. Furthermore, we define F to be well-defined near $\bar{x} \in C$ if there exists an open neighborhood of $\bar{x}$, $\mathcal{N}(\bar{x})$, such that $C \cap \mathcal{N}(\bar{x}) \subseteq D$. By saying the function is well-defined near $\bar{x}$, we are simply stating that F is defined for all $x \in C$ sufficiently close to $\bar{x}$. A function not well-defined near $\bar{x}$ is termed ill-defined near $\bar{x}$.

We will say that F has a well-defined Jacobian at $\bar{x} \in C$ if there exists an open neighborhood of $\bar{x}$, $\mathcal{N}(\bar{x})$, such that $\mathcal{N}(\bar{x}) \subseteq D$ and F is continuously differentiable on $\mathcal{N}(\bar{x})$. Otherwise the function has an ill-defined Jacobian at $\bar{x}$. We note that a well-defined Jacobian at $\bar{x}$ implies that the MCP has a well-defined function near $\bar{x}$, but the converse is not true.

PATH uses both function and Jacobian information in its attempt to solve the MCP. Therefore, both of these definitions are relevant. We discuss cases where the function and Jacobian are ill-defined in the next two subsections. We illustrate uses for the merit function information and final point statistics within the context of these problems.

Function Undefined We begin with a one-dimensional problem for which F is ill-defined at $x = 0$ as follows:

$$0 \leq x \quad \perp \quad \tfrac{1}{x} \geq 0.$$

Here x must be strictly positive because $\frac{1}{x}$ is undefined at $x = 0$. This condition implies that $F(x)$ must be equal to zero. Since $F(x)$ is strictly positive for all x strictly positive, this problem has no solution.

We are able to perform this analysis because the dimension of the problem is small. Preprocessing linear problems can be done by the solver in an attempt to detect obviously inconsistent problems, reduce problem size, and identify active components at the solution. Similar processing can be done for nonlinear models, but the analysis becomes more difficult to perform. Currently, PATH only checks the consistency of the bounds and removes fixed variables and the corresponding complementary equations from the model.

A modeler would likely not know a priori that a problem has no solution and would thus attempt to formulate and solve it. GAMS code for the model above is provided in Figure 8. We must specify an initial value for x in the code. If we were to not provide one, GAMS would use $x = 0$ as the default value, notice that F is undefined at the initial point, and terminate before giving the problem to PATH. The error message indicates that the function $\frac{1}{x}$ is ill-defined at $x = 0$, but does not imply that the corresponding MCP problem has no solution.

Figure 8: GAMS Code for Ill-Defined Function

```
positive variable x;
equations F;

F.. 1 / x =g= 0;

model simple / F.x /;

x.l = 1e-6;

solve simple using mcp;
```

After setting the starting point, GAMS generates the model, and PATH proceeds to "solve" it. A portion of the output relating statistics about the solution is given in Figure 9 PATH uses the Fischer Function indicator as its termination criteria by default, but evaluates all of the merit functions given in Section Merit Functions at the final point. The Normal Map merit function, and to a lesser extent, the complementarity error, indicate that the "solution" found does not necessarily solve the MCP.

Figure 9: PATH Output for Ill-Defined Function

```
FINAL STATISTICS
Inf-Norm of Complementarity . .  1.0000e+00 eqn: (F)
Inf-Norm of Normal Map. . . . .  1.1181e+16 eqn: (F)
Inf-Norm of Minimum Map . . . .  8.9441e-17 eqn: (F)
Inf-Norm of Fischer Function. .  8.9441e-17 eqn: (F)
Inf-Norm of Grad Fischer Fcn. .  8.9441e-17 eqn: (F)

FINAL POINT STATISTICS
Maximum of X. . . . . .  8.9441e-17 var: (X)
Maximum of F. . . . . .  1.1181e+16 eqn: (F)
Maximum of Grad F . . . . . .  1.2501e+32 eqn: (F)
                                          var: (X)
```

To indicate the difference between the merit functions, Figure 10 plots them all for this simple example. We note that as x approaches positive infinity, numerically, we are at a solution to the problem with respect to all of the merit functions except for the complementarity error, which remains equal to one. As x approaches zero, the merit functions diverge, also indicating that $x = 0$ is not a solution.

The natural residual and Fischer function tend toward 0 as $x \downarrow 0$. From these measures, we might think $x = 0$ is the solution. However, as previously remarked F is ill-defined at $x = 0$. F and ∇F become very large, indicating that the function (and Jacobian) might not be well-defined. We might be tempted to conclude that if one of the merit function indicators is not close to zero, then we have not found a solution. This conclusion is not always warranted. When one of the indicators is non-zero, we have reservations about the solution, but we cannot eliminate the possibility that we are actually close to a solution. If we slightly perturb the original problem to

$$0 \leq x \quad \perp \quad \tfrac{1}{x+\epsilon} \geq 0$$

for a fixed $\epsilon > 0$, the function is well-defined over $C = \mathbf{R}^n_+$ and has a unique solution at $x = 0$. In this case, by starting at $x > 0$ and sufficiently small, all of the merit functions, with the exception of the Normal Map, indicate that we have solved the problem as is shown by the output in Figure 11 for $\epsilon = 1 * 10^{-6}$ and $x = 1 * 10^{-20}$.

Figure 11: PATH Output for Well-Defined Function

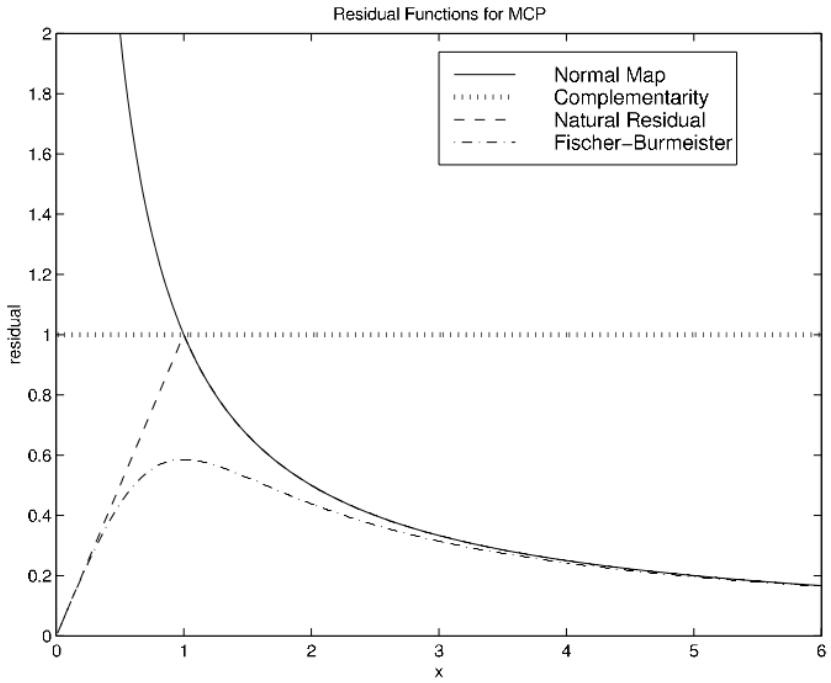


Figure 10: Merit Function Plot

```
FINAL STATISTICS
Inf-Norm of Complementarity . .  1.0000e-14 eqn: (G)
Inf-Norm of Normal Map. . . . .  1.0000e+06 eqn: (G)
Inf-Norm of Minimum Map . . . .  1.0000e-20 eqn: (G)
Inf-Norm of Fischer Function. .  1.0000e-20 eqn: (G)
Inf-Norm of Grad Fischer Fcn. .  1.0000e-20 eqn: (G)

FINAL POINT STATISTICS
Maximum of X. . . . . .  1.0000e-20 var: (X)
Maximum of F. . . . . .  1.0000e+06 eqn: (G)
Maximum of Grad F . . . . . .  1.0000e+12 eqn: (G)
                                          var: (X)
```

In this case, the Normal Map is quite large and we might think that the function and Jacobian are undefined. When only the normal map is non-zero, we may have just mis-identified the optimal basis. By setting the `merit_function normal` option, we can resolve the problem, identify the correct basis, and solve the problem with all indicators being close to zero. This example illustrates the point that all of these tests are not infallible. The modeler still needs to do some detective work to determine if they have found a solution or if the algorithm is converging to a point where the function is ill-defined.

Jacobian Undefined Since PATH uses a Newton-like method to solve the problem, it also needs the Jacobian of F to be well-defined. One model for which the function is well-defined over C, but for which the Jacobian is undefined at the solution is: $0 \leq x \perp -\sqrt{x} \geq 0$. This model has a unique solution at $x = 0$.

Using PATH and starting from the point $x = 1 * 10^{-14}$, PATH generates the output given in Figure 12 .

Figure 12: PATH Output for Ill-Defined Jacobian

```
FINAL STATISTICS
Inf-Norm of Complementarity . .  1.0000e-07 eqn: (F)
Inf-Norm of Normal Map. . . . .  1.0000e-07 eqn: (F)
Inf-Norm of Minimum Map . . . .  1.0000e-07 eqn: (F)
Inf-Norm of Fischer Function. .  2.0000e-07 eqn: (F)
Inf-Norm of Grad FB Function. .  2.0000e+00 eqn: (F)

FINAL POINT STATISTICS
Maximum of X. . . . . .  1.0000e-14 var: (X)
Maximum of F. . . . . .  1.0000e-07 eqn: (F)
Maximum of Grad F . . . . . .  5.0000e+06 eqn: (F)
                                          var: (X)
```

We can see that the gradient of the Fischer Function is nonzero and the Jacobian is beginning to become large. These conditions indicate that the Jacobian is undefined at the solution. It is therefore important for a modeler to inspect the given output to guard against such problems.

If we start from $x = 0$, PATH correctly informs us that we are at the solution. The large value reported for the Jacobian is an indication that the Jacobian is undefined.

Poorly Scaled Models

Well-defined problems can still have various numerical problems that impede the algorithm's convergence. One particular problem is a badly scaled Jacobian. In such cases, we can obtain a poor "Newton" direction because of numerical problems introduced in the linear algebra performed. This problem can also lead the code to a point from which it cannot recover.

The final model given to the solver should be scaled such that we avoid numerical difficulties in the linear algebra. The output provided by PATH can be used to iteratively refine the model so that we eventually end up with a well-scaled problem. We note that we only calculate our scaling statistics at the starting point provided. For nonlinear problems these statistics may not be indicative of the overall scaling of the model. Model specific knowledge is very important when we have a nonlinear problem because it can be used to appropriately scale the model to achieve a desired result.

We look at the `titan.gms` model in MCPLIB, that has some scaling problems. The relevant output from PATH for the original code is given in Figure 13.

Figure 13: PATH Output - Poorly Scaled Model

```
INITIAL POINT STATISTICS
Maximum of X. . . . . .  4.1279e+06 var: (w.29)
Maximum of F. . . . . .  2.2516e+00 eqn: (a1.33)
Maximum of Grad F . . . . . .  6.7753e+06 eqn: (a1.29)
                                          var: (x1.29)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . . . . .  9.4504e+06 eqn: (a2.29)
Minimum Row Norm. . . . . .  2.7680e-03 eqn: (g.10)
Maximum Column Norm . . . . . .  9.4504e+06 var: (x2.29)
Minimum Column Norm . . . . . .  1.3840e-03 var: (w.10)
```

The maximum row norm is defined as

$$\max_{1\le i\le n} \sum_{1\le j\le n} \mid (\nabla F(x))_{ij} \mid$$

and the minimum row norm is

$$\min_{1\le i\le n} \sum_{1\le j\le n} \mid (\nabla F(x))_{ij} \mid .$$

Similar definitions are used for the column norm. The norm numbers for this particular example are not extremely large, but we can nevertheless improve the scaling. We first decided to reduce the magnitude of the `a1` block of equations as indicated by PATH. Using the GAMS modeling language, we can scale particular equations and variables using the .scale attribute. To turn the scaling on for the model we use the .scaleopt model attribute. After scaling the `a1` block, we re-ran PATH and found an additional block of equations that also needed scaling, `a2`. We also scaled some of the variables, `g` and `w`. The code added to the model follows:

```
titan.scaleopt = 1;
a1.scale(i) = 1000;
a2.scale(i) = 1000;
g.scale(i) = 1/1000;
w.scale(i) = 100000;
```

By scaling these equation and variable blocks, we have improved the model scaling. The statistics for the manually scaled model are given in Figure 14.

Figure 14: PATH Output - Well-Scaled Model

```
INITIAL POINT STATISTICS
Maximum of X. . . . . .  1.0750e+03 var: (x1.49)
Maximum of F. . . . . .  3.9829e-01 eqn: (g.10)
Maximum of Grad F . . . . . .  6.7753e+03 eqn: (a1.29)
                                          var: (x1.29)

INITIAL JACOBIAN NORM STATISTICS
Maximum Row Norm. . . . . .  9.4524e+03 eqn: (a2.29)
Minimum Row Norm. . . . . .  2.7680e+00 eqn: (g.10)
Maximum Column Norm . . . . . .  9.4904e+03 var: (x2.29)
Minimum Column Norm . . . . . .  1.3840e-01 var: (w.10)
```

For this particular problem PATH cannot solve the unscaled model, while it can find a solution to the scaled model. Using the scaling features of the GAMS language and the information provided by PATH we are able to remove some of the problem's difficulty and obtain better performance from PATH.

It is possible to get even more information on initial point scaling by inspecting the GAMS listing file. The equation row listing gives the values of all the entries of the Jacobian at the starting point. The row norms generated by PATH give good pointers for starting to use the row listing.

Not all of the numerical problems are directly attributable to poorly scaled models. Problems for which the Jacobian of the active constraints is singular or nearly singular can also cause numerical difficulty as illustrated next.

Singular Models

Assuming that the problem is well-defined and properly scaled, we can still have a Jacobian for which the active constraints are singular or nearly singular (i.e. it is ill-conditioned). When problems are singular or nearly singular, we are also likely to have numerical problems. As a result the "Newton" direction obtained from the linear problem solver can be very bad. In PATH, we can use proximal perturbation or add artificial variables to attempt to remove the singularity problems from the model. However, it is most often beneficial for solver robustness to remove singularities if possible.

The easiest problems to detect are those for which the Jacobian has zero rows and columns. A simple problem for which we have zero rows and columns is:

$$-2 \leq x \leq 2 \quad \perp \quad -x^2 + 1.$$

Note that the Jacobian, $-2x$, is non-singular at all three solutions, but singular at the point $x = 0$. Output from PATH on this model starting at $x = 0$ is given in Figure 15.

Figure 15: PATH Output - Zero Rows and Columns

```
INITIAL POINT STATISTICS
Zero column of order. . . . . .  0.0000e+00 var: (X)
Zero row of order . . . . . .  0.0000e+00 eqn: (F)
Total zero columns. . . . . .  1
Total zero rows . . . . . .  1
Maximum of F. . . . . .  1.0000e+00 eqn: (F)
Maximum of Grad F . . . . . .  0.0000e+00 eqn: (F)
                                          var: (X)
```

We display in the code the variables and equations for which the row/column in the Jacobian is close to zero. These situations are problematic and for nonlinear problems likely stem from the modeler providing an inappropriate starting point or fixing some variables resulting in some equations becoming constant. We note that the solver may perform well in the presence of zero rows and/or columns, but the modeler should make sure that these are what was intended.

Singularities in the model can also be detected by the linear solver. This in itself is a hard problem and prone to error. For matrices which are poorly scaled, we can incorrectly identify "linearly dependent" rows because of numerical problems. Setting `output_factorization_singularities yes` in an options file will inform the user which equations the linear solver thinks are linearly dependent. Typically, singularity does not cause a lot of problems and the algorithm can handle the situation appropriately. However, an excessive number of singularities are cause for concern. A further indication of possible singularities at the solution is a lack of quadratic convergence to the solution.

5.31.5 Case Study: Von Thunen Land Model

We now turn our attention towards using the diagnostic information provided by PATH to improve an actual model. The Von Thunen land model is a problem recognized in the mathematical programming literature for its computational difficulty. We attempt to understand more carefully the facets of the problem that make it difficult to solve. This will enable us to outline and identify these problems and furthermore to extend the model to a more realistic and computationally more tractable form.

5.31.5.1 Classical Model

The problem is cast in the Arrow-Debreu framework as an equilibrium problem.The basic model is a closed economy consisting of three economic agents, a landowner, a worker and a porter. There is a central market, around which concentric regions of land are located. Since the produced goods have to be delivered to the market, this is an example of a spatial price equilibrium. The key variables of the model are the prices of commodities, land, labour and transport. Given these prices, it is assumed that the agents demand certain amounts of the commodities, which are supplied so as to maximize profit in each sector. Walras' law is then a consequence of the assumed competitive paradigm, namely that supply will equal demand in the equilibrium state.

We now describe the problems that the consumers and the producers face. We first look at consumption and derive a demand function for each of the consumer agents in the economy. Each of these agents has a utility function, that they wish to maximize subject to their budgetary constraints. As is typical in such problems, the utility function is assumed to be Cobb-Douglas:

$$u_a(d) = \prod_c d_c^{\alpha_{c,a}}, \quad \alpha_{c,a} \geq 0, \sum_c \alpha_{c,a} = 1,$$

where the $\alpha_{c,a}$ are given parameters dependent only on the agent. For each agent a, the variables d_c represent quantities of the desired commodities c. In the Von Thunen model, the goods are wheat, rice, corn and barley. The agent's endowments determine their budgetary constraint as follows. Given current market prices, an agent's wealth is the value of the initial endowment of goods at those prices. The agent's problem is therefore

$$\max_d u_a(d) \text{ subject to } \langle p, d\rangle \leq \langle p, e_a\rangle, d \geq 0,$$

where e_a is the endowment bundle for agent a. A closed form solution, corresponding to demand from agent a for commodity c is thus

$$d_{c,a}(p) := \frac{\alpha_{c,a}\langle p, e_a\rangle}{p_c}.$$

Note that this assumes the prices of the commodities p_c are positive.

The supply side of the economy is similar. The worker earns a wage w_L for his labour input. The land is distributed around the market in rings with a rental rate w_r associated with each ring r of land. The area of land a_r in each ring is an increasing function of r. The model assumes that labour and land are substitutable via a constant elasticities of substitution (CES) function.

Consider the production $x_{c,r}$ of commodity c in region r. In order to maximize profit (or minimize costs), the labour y_L and land use y_r solve

$$\min w_L y_L + w_r y_r \text{ subject to } \phi_c y_L^{\beta_c} y_r^{1-\beta_c} \geq x_{c,r},\ y_L, y_r \geq 0, \tag{12}$$

where ϕ_c is a given cost function scale parameter, and $\beta_c \in [0, 1]$ is the share parameter. The technology constraint is precisely the CES function allowing a suitable mix of labour and land use. Again, a closed form solution can be calculated. For example, the demand for labour in order to produce $x_{c,r}$ of commodity c in region r is given by

$$x_{c,r} \frac{\beta_c \left(\frac{w_L}{\beta_c}\right)^{\beta_c} \left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_L}.$$

Considering all such demands, this clearly assumes the prices of inputs w_L, w_r are positive. A key point to note is that input commodity (factor) demands to produce $x_{c,r}$ can be determined by first solving (12) for unit demand $x_{c,r} \equiv 1$ and then multiplying these factor demands by the actual amount desired. Let $\bar{y}_L$ and $\bar{y}_r$ denote the optimal solutions of (12) with $x_{c,r} \equiv 1$. Using this fact, the *unit* production cost $\gamma_{c,r}$ for commodity c in region r can be calculated as follows:

$$\begin{aligned}\gamma_{c,r} &= w_L\bar{y}_L + w_r\bar{y}_r \\ &= w_L\frac{\beta_c\left(\frac{w_L}{\beta_c}\right)^{\beta_c}\left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_L} + w_r\frac{(1-\beta_c)\left(\frac{w_L}{\beta_c}\right)^{\beta_c}\left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}}{\phi_c w_r} \\ &= \frac{1}{\phi_c}\left(\frac{w_L}{\beta_c}\right)^{\beta_c}\left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c}.\end{aligned}$$

Transportation is provided by a porter, earning a wage w_p. If we denote the unit cost for transportation of commodity c by t_c, then unit transportation cost to market is

$$T_{c,r}(w_p) := t_c d_r w_p,$$

where d_r is the distance of region r to the market. Spatial price equilibrium arises from the consideration:

$$0 \le x_{c,r} \perp \gamma_{c,r}(w_L, w_r) + T_{c,r}(w_p) \ge p_c.$$

This is intuitively clear; it states that commodity c will be produced in region r only if the combined cost of production and transportation equals the market price.

The above derivations assumed that the producers and consumers acted as price takers. Walras' law is now invoked to determine the prices so that markets clear. The resulting complementarity problem is:

$$\gamma_{c,r} = \frac{1}{\phi_c}\left(\frac{w_L}{\beta_c}\right)^{\beta_c}\left(\frac{w_r}{1-\beta_c}\right)^{1-\beta_c} \tag{13}$$

$$0 \le x_{c,r} \perp \gamma_{c,r} + T_{c,r}(w_p) \ge p_c \tag{14}$$

$$0 \le w_L \perp e_L \ge \sum_{r,c} x_{c,r}\frac{\beta_c\gamma_{c,r}}{w_L} \tag{15}$$

$$0 \le w_r \perp a_r \ge \sum_{c} \frac{x_{c,r}(1-\beta_c)\gamma_{c,r}}{w_r} \tag{16}$$

$$0 \le w_p \perp e_P \ge \sum_{r,c} t_c d_r x_{c,r} \tag{17}$$

$$0 \le p_c \perp \sum_r x_{c,r} \ge \frac{\alpha_{c,P}e_P w_p + \alpha_{c,L}e_L w_L + \alpha_{c,O}\sum_r w_r a_r}{p_c} \tag{18}$$

Note that in (15), (16) and (17), the amounts of labour, land and transport are bounded from above, and hence the prices on these inputs are determined as multipliers (or shadow prices) on the corresponding constraints. The final relationship (18) in the above complementarity problem corresponds to market clearance; prices are nonnegative and can only be positive if supply equals demand. (Some modelers multiply the last inequality throughout by p_c. This removes problems where p_c becomes zero, but can also introduce spurious solutions.)

The Arrow-Debreu theory guarantees that the problem is homogeneous in prices; $(x, \lambda w, \lambda p)$ is also a solution whenever (x, w, p) solves the above. Typically this singularity in the model is removed by fixing a numeraire, that is fixing a price (for example $w_L = 1$) and dropping the corresponding complementary relationship.

Unfortunately, in this formulation even after fixing a numeraire, some of the variables p and w may go to zero, resulting in an ill-defined problem. In the case of the Von Thunen land model, the rental price of land w_r decreases as the distance to market increases, and for remote rings of land, it becomes zero. A standard modeling fix is to put artificial lower bounds on these variables. Even with this fix, the problem typically remains very hard to solve. More importantly, the homogeneity property of the prices used above to fix a numeraire no longer holds, and the corresponding complementary relationship (which was dropped from the problem) may fail to be satisfied. It therefore matters which numeriare is fixed, and many modelers run into difficulty since in many cases the solution found by a solver is invalid for the originally posed model.

In order to test our diagnostic information, we implemented a version of the above model in GAMS. The model corresponds closely to the MCPLIB model pgvon105.gms except we added more regions to make the problem even more difficult. The model file has been documented more fully, and the data rounded to improve clarity.

Our first trial was to solve the model without fixing a numeraire. In this case, PATH 4.x failed to find a solution. At the starting point, the indicators described in Sectiion Ill-Defined Models are reasonable, and there are no zero rows/columns in the Jacobian. At the best point found, all indicators are still reasonable. However, the listing file indicates a large number of division by zero problems occurring in (16). We also note that a nonzero proximal perturbation is used in the first iteration of the crash method. This is an indication of singularities. We therefore added an option to output factorization singularities, and singularities appeared in the first iteration. At this point, we decided to fix a numeraire to see if this alleviated the problem.

We chose to fix the labour wage rate to 1. After increasing the iterations allowed to 100,000, PATH 4.x solved the problem. The statistics at the solution are cause for concern. In particular, the gradient of the Fischer function is 7 orders of magnitude larger than all the other residuals. Furthermore, the Jacobian is very large at the solution point. Looking further in the listing file, a large number of division by zero problems occur in (16).

To track down the problem further, we added an artificial lower bound on the variables w_r of 10^{-5}, that would not be active at the aforementioned solution. Resolving gave the same "solution", but resulted in the domain errors disappearing.

Although the problem is solved, there is concern on two fronts. Firstly, the gradient of the Fischer function should go to zero at the solution. Secondly, if a modeler happens to make the artificial lower bounds on the variables a bit larger, then they become active at the solution, and hence the constraint that has been dropped by fixing the price of labour at 1 is violated at this point. Of course, the algorithm is unable to detect this problem, since it is not part of the model that is passed to it, and the corresponding output looks satisfactory.

We are therefore led to the conclusion that the model as postulated is ill-defined. The remainder of this section outlines two possible modeling techniques to overcome the difficulties with ill-defined problems of this type.

5.31.5.2 Intervention Pricing

The principal difficulty is the fact that the rental prices on land go to zero as proximity to the market decreases, and become zero for sufficiently remote rings. Such a property is unlikely to hold in a practical setting. Typically, a landowner has a minimum rental price (for example, land in fallow increases in value). As outlined above, a fixed lower bound on the rental price violates the well-established homogeneity property. A suggestion postulated by Professor Thomas Rutherford is to allow the landowner to intervene and "purchase-back" his land whenever the rental cost gets smaller than a certain fraction of the labour wage.

The new model adds a (homogeneous in price) constraint

$$0 \leq i_r \quad \perp \quad w_r \geq 0.0001 * w_L$$

and modifies (16) and (18) as follows:

$$0 \leq w_r \perp a_r - i_r \geq \sum_c \frac{x_{c,r}(1-\beta_c)\gamma_{c,r}}{w_r}$$

$$0 \leq p_c \perp \sum_r x_{c,r} \geq \frac{\alpha_{c,P} e_P w_p + \alpha_{c,L} e_L w_L + \alpha_{c,O} \sum_r w_r(a_r - i_r)}{p_c}. \tag{19}$$

Given the intervention purchase, we can now add a lower bound on w_r to avoid division by zero errors. In our model we chose 10^{-5} since this will never be active at the solution and therefore will not affect the positive homogeneity. After this reformulation, PATH 4.x solves the problem. Furthermore, the gradient of the Fischer function, although slightly larger than the other residuals, is quite small, and can be made even smaller by reducing the convergence tolerance of PATH. Inspecting the listing file, the only difficulties mentioned are division by zero errors in the market clearance condition (19), that can be avoided a posteori by imposing an artificial (inactive) lower bound on these prices. We chose not to do this however.

5.31.5.3 Nested Production and Maintenance

Another observation that can be used to overcome the land price going to zero is the fact that land typically requires some maintenance labour input to keep it usable for crop growth. Traditionally, in economics, this is carried out by providing a nested CES function as technology input to the model. The idea is that commodity c in region r is made from labour and an intermediate good, which is "maintained land". Essentially, the following production problem replaces (12):

$$\begin{array}{rc} \min_{y_M, y_L, y_r, g} & w_L(y_M + y_L) + w_r y_r \\ \text{subject to} & y_r \geq (1 - \beta_c - \epsilon)g \\ & y_M \geq \epsilon g \\ & \phi_c y_L^{\beta_c} g^{1-\beta_c} \geq 1, \\ & y_M, y_L, y_r, g \geq 0. \end{array}$$

Note that the variable y_M represents "maintenance labour" and g represents the amount of "maintained land" produced, an intermediate good. The process of generating maintained land uses a Leontieff production function, namely

$$\min(\lambda_r y_r, \lambda_M y_M) \geq g.$$

Here $\lambda_M = \frac{1}{\epsilon}$, ϵ small, corresponds to small amounts of maintenance labour, while $\lambda_r = \frac{1}{1-\beta_c-\epsilon}$ is chosen to calibrate the model correctly. A simple calculus exercise then generates appropriate demand and cost expressions. The resulting complementarity problem comprises (14), (17), (18) and

$$\begin{aligned} \gamma_{c,r} &= \frac{w_L^{\beta_c}}{\phi_c} \left(\frac{w_L \epsilon + w_r(1-\beta_c-\epsilon)}{1-\beta_c} \right)^{1-\beta_c} \\ 0 \leq w_L \quad \perp \quad e_L &\geq \sum_{r,c} x_{c,r}\gamma_{c,r} \left(\frac{\beta_c}{w_L} + \frac{\epsilon(1-\beta_c)}{w_L\epsilon + w_r(1-\beta_c-\epsilon)} \right) \\ 0 \leq w_r \quad \perp \quad a_r &\geq \sum_c \frac{x_{c,r}\gamma_{c,r}(1-\beta_c)(1-\beta_c-\epsilon)}{w_L\epsilon + w_r(1-\beta_c-\epsilon)} \end{aligned}$$

After making the appropriate modifications to the model file, PATH 4.x solved the problem on defaults without any difficulties. All indicators showed the problem and solution found to be well-posed.

5.32 quadMINOS

For LP (and RMIP) models that cause difficulty for the standard solvers, quadMINOS provides considerably greater reliability and accuracy than MINOS by using quadruple-precision floating-point arithmetic. Although it is slower by a factor of 20 or 30 for cold-start runs, you can use standard MINOS or another LP solver first and let them do their best. Warm-starts with quadMINOS may give greatly improved solutions at moderate cost.

To take advantage of quadMINOS, the main requirement is to set a few runtime options to non-default values in order to request higher accuracy. For example, it is reasonable to set

```
Feasibility tolerance  1e-15
Optimality  tolerance  1e-15
```

because IEEE Quad floating-point has nearly 34 digits of precision. For an example of using Double and Quad MINOS together, see **[DQQ]** in the GAMS model library.

Note

GAMS/quadMINOS can only solve LPs and RMIPs at the moment because GAMS does not provide function evaluations in quad-precision.

The documentation of MINOS also refers to quadMINOS.

5.33 SBB

5.33.1 Introduction

SBB is a GAMS solver for Mixed Integer Nonlinear Programming (MINLP) models. It is based on a combination of the standard Branch and Bound (B&B) method known from Mixed Integer Linear Programming and some of the standard NLP solvers already supported by GAMS. SBB can use all GAMS NLP solvers as subsolvers but it works best with NLP solvers that can utilize a near optimal point as a starting point like Conopt, Minos, and Snopt.

SBB supports all types of discrete variables supported by GAMS, including Binary, Integer, Semicont, Semiint, Sos1, and Sos2.

5.33.2 The Branch and Bound Algorithm

The Relaxed Mixed Integer Nonlinear Programming (RMINLP) model is initially solved using the starting point provided by the modeler. SBB will stop immediately if the RMINLP model is unbounded or infeasible, or if it fails (see option `infeasseq` and `failseq` below for an exception). If all discrete variables in the RMINLP model are integer, SBB will return this solution as the optimal integer solution. Otherwise, the current solution is stored and the Branch and Bound procedure will start.

During the Branch and Bound process, the feasible region for the discrete variables is subdivided, and bounds on discrete variables are tightened to new integer values to cut off the current non-integer solutions. Each time a bound is tightened, a new, tighter NLP submodel is solved starting from the optimal solution to the previous looser submodel. The objective function values from the NLP submodel is assumed to be lower bounds on the objective in the restricted feasible space (assuming minimization), even though the local optimum found by the NLP solver may not be a global optimum. If the NLP solver returns a Locally Infeasible status for a submodel, it is usually assumed that there is no feasible solution to the submodel, even though the infeasibility only has been determined locally (see option `infeasseq` below for an exception). If the model is convex, these assumptions will be satisfied and SBB will provide correct bounds. If the model is not convex, the objective bounds may not be correct and better solutions may exist in other, unexplored parts of the search space.

5.33.3 SBB with Pseudo Costs

Over the last decades quite a number of search strategies have been successfully introduced for mixed integer linear programming (for details see e.g. J.T. Linderoth and M.W.P. Savelsbergh, A Computational Study of Search Strategies for Mixed Integer Programming, INFORMS Journal on Computing, 11(2), 1999). Pseudo costs are key elements of sophisticated search strategies. Using pseudo costs, we can estimate the degradation of the objective function if we move a fractional variable to a close integer value. Naturally, the variable selection can be based on pseudo costs (see SBB option `varsel`). Node selection can also make use of pseudo cost: If we can estimate the change of the objective for moving one fractional variable to the closed integer value, we can then aggregate this change for all fractional variables, to estimate the objective of the best integer solution reachable from a particular node (see SBB option `nodesel`).

Unfortunately, the computation of pseudo cost can be a substantial part of the overall computation. Models with a large number of fractional variables in the root node are **not** good candidates for search strategies which require pseudo costs (`varsel 3, nodesel 3,5,6`). The impact (positive or negative) of using pseudo cost depends significantly on the particular model. At this stage, general statements are difficult to make.

Selecting pseudo cost related search strategies (`varsel 3, nodesel 3,5,6`) may use computation time which sometimes does not pay off. However, we encourage the user to try these options for difficult models which require a large number of branch-and-bound nodes to solve.

5.33.4 The SBB Options

SBB works like other GAMS solvers, and many options can be set in the GAMS model. The most relevant GAMS options are IterLim, ResLim, NodLim, OptCA, OptCR, OptfFile, Cheat, and CutOff. A description of all available GAMS options can be found in the GAMS User's Guide Solver related options. GAMS options PriorOpt and TryInt are also accepted by SBB.

SBB uses the `var.prior` information to select the fractional variable with the smallest priority during the variable selection process. SBB uses the TryInt information to set the branching direction in the B&B algorithm. At the beginning, SBB looks at the levels of the discrete variables provided by the user and if Abs(Round(x.l)-x.l) $<$ m.TryInt, SBB will branch on that variable in the direction of Round(x.l). For example, x.l=0.9 and m.TryInt=0.2. We have Abs(Round(0.9)-0.9)=0.1 $<$ 0.2, so when SBB decides to branch on this variable (because it is fractional, lets say with value 0.5), the node explored next will have the additional constraint x $\geq$ 1 (the node with x $\leq$ 0 will be explored later). If everything goes well (there is the chance that we end up in a different local optima in the subsolves for non-convex problems), SBB should reproduce a preset incumbent solution in a couple of nodes.

If you specify <`modelname`>`.OptFile = 1;` before the Solve statement in your GAMS model, SBB will then look for and read an option file with the name `sbb.opt` (see The Solver Option File for general use of solver option files). Unless explicitly specified in the SBB option file, the NLP subsolvers will not read an option file. The syntax for the SBB option file is

```
optname value
```

with one option on each line.

For example,

```
rootsolver conopt.1
subsolver snopt
loginterval 10
```

The first two lines determine the NLP subsolvers for the Branch and Bound procedure. CONOPT with the option file `conopt.opt` will be used for solving the root node. SNOPT with no option file will be used for the remaining nodes. The last option determines the frequency for log line printing. Every 10th node, and each node with a new integer solution, causes a log line to be printed. The following options are implemented:

Option	Description	Default
acceptnonopt	accepts feasible solution from subsolver	0
avgresmult	average resource multiplicator	5
dfsstay	keeps DFS node selection after solution has been found	0
epint	integer feasibility tolerance	1.0e-5
failseq	solver sequence for failed nodes	
infeasseq	solver sequence for infeasible nodes	
intsollim	maximum number of integer solutions	2100000000
loginterval	progress display interval	1
loglevel	level of solver display	1
memnodes	maximum number of nodes in memory	10000
miptrace	filename of MIP trace file	
miptracenode	node interval when a trace record is written	100
miptracetime	time interval when a trace record is written	5.0
nodesel	node selection strategy	0
printbbinfo	prints additional node info	0
rootsolver	solver for the root node	GAMS NLP solver
solvelink	Solvelink for GAMS NLP solver	5
subiter	iteration limit for the subsolve	GAMS iterlim
subres	resource limit for the subsolve	GAMS reslim
subsolver	solver for the subproblems	GAMS NLP solver
usercallparmfile	Command-line parameter include file used in GAMS command-line calls triggered by BCH	
usergdxname	the name of the GDX file exported from the solver with the solution at the node	bchout.gdx
usergdxnameinc	the name of the GDX file exported from the solver with the incumbent solution	bchout_i.gdx
userheurcall	the GAMS command line to call the heuristic	
userheurfirst	calls the cut generator for the first n nodes	10
userheurfreq	determines the frequency of the cut generator model calls	10
userheurinterval	determines the interval when to apply the multiplier for the frequency of the cut generator model calls	100
userheurmult	determines the multiplier for the frequency of the cut generator model calls	2
userheurnewint	calls the heuristic if the solver found a new integer feasible solution	0
userheurobjfirst	Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement	50
varsel	variable selection strategy at each node	0

acceptnonopt *(boolean)*: accepts feasible solution from subsolver ↩

If this option is set to 1 and the subsolver terminates with solver status *Terminated by Solver* and model status *Intermediate Nonoptimal* SBB takes this as a good solution and keeps on going. In default mode such a return is treated as a subsolver failure and the failseq is consulted.

Default: `0`

avgresmult *(integer)*: average resource multiplicator ↩

Similar to subres, this option allows the user to control the time limit spend in a node. SBB keeps track of how much time is spent in the nodes, and builds an average over time. This average multiplied by the factor `avgresmult` is set as a time limit for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure: the node is ignored or the solvers in the failseq are called. The default multiplier `avgresmult` is 5. Setting `avgresmult` to 0 will disable the automatic time limit feature. A multiplier is not very useful for very small node solution times; therefore, independent of each node, SBB grants the solver at least 5 seconds to solve the node. The competing option subres overwrites the automatically generated resource limit.

Default: `5`

dfsstay *(integer)*: keeps DFS node selection after solution has been found ↩

If the node selection is a B∗/DFS mix, SBB switches frequently to DFS node selection mode. It switches back into B∗ node selection mode, if no subnodes were created (new int, pruned, infeasible, fail). It can be advantageous to search the neighborhood of the last node also in a DFS manner. Setting `dfsstay` to n instructs SBB to stay in DFS mode for another n nodes.

Default: `0`

epint *(real)*: integer feasibility tolerance ↩

The integer infeasibility tolerance.

Range: [`1e-9`, `1`]

Default: `1.0e-5`

failseq *(string)*: solver sequence for failed nodes ↩

`solver1[.n1] solver2[.n2] ...` where `solver1` is the name of a GAMS NLP solver to be used if the default solver fails, i.e., if it was not stopped by an iteration, resource, or domain limit and does not return a locally optimal or locally infeasible solution. `n1` is the value of optfile passed to the alternative NLP solver. If `.n1` is left blank it is interpreted as zero. Similarly, `solver2` is the name of a GAMS NLP solver that is used if `solver1` fails, and `n2` is the value of optfile passed to the second NLP solver. If you have a difficult model where solver failures are not unlikely, you may add more `solver.n` pairs. You can use the same solver several times with different options files. `failseq conopt conopt.2 conopt.3` means to try CONOPT with no options file. If this approach also fails, try CONOPT with options file `conopt.op2`, and if it again fails, try CONOPT with options file `conopt.op3`. If all solver and options file combinations fail the node will be labeled *ignored* and the node will not be explored further. The default is to try only one solver (the `rootsolver` or `subsolver`) and to ignore nodes with a solver failure.

infeasseq *(string)*: solver sequence for infeasible nodes ↩

`level solver1[.n1] solver2[.n2] ...` The purpose of `infeasseq` is to avoid cutting parts of the search tree that appear to be infeasible but really are feasible. If the NLP solver labels a node *Locally Infeasible* and the model is not convex a feasible solution may actually exist. If SBB is high in the search tree it can be very drastic to prune the node immediately. SBB is therefore directed to try the solver/option combinations in the list as long as the depth in the search tree is less than the integer value `level`. If the list is exhausted without finding a feasible solution, the node is assumed to be infeasible. The default is to trust that *Locally Infeasible* nodes are indeed infeasible and to remove them from further consideration.

intsollim *(integer)*: maximum number of integer solutions ↩

Maximum number of integer solutions. If this number is exceeded, SBB will terminate and return the best solution found so far.

Default: `2100000000`

loginterval *(integer)*: progress display interval ↩

The interval (number of nodes) for which log lines are written.

Default: `1`

loglevel *(integer)*: level of solver display ↩

The level of log output.

Default: `1`

value	meaning
0	only SBB log lines with one line every loginterval nodes
1	NLP solver log for the root node plus SBB loglines as 0
2	NLP solver log for all nodes plus SBB log lines as 0

memnodes *(integer)*: maximum number of nodes in memory ↩

The maximum number of nodes SBB can have in memory. If this number is exceeded, SBB will terminate and return the best solution found so far.

Default: `10000`

miptrace *(string)*: filename of MIP trace file ↩

More info is available in chapter Solve trace

miptracenode *(integer)*: node interval when a trace record is written ↩

More info is available in chapter Solve trace

Default: `100`

miptracetime *(real)*: time interval when a trace record is written ↩

More info is available in chapter Solve trace

Default: `5.0`

nodesel *(integer)*: node selection strategy ↩

Node selection scheme.

Default: `0`

value	meaning
0	automatic
1	Depth First Search (DFS)
2	Best Bound (BB)
3	Best Estimate (BE)
4	DFS/BB mix
5	DFS/BE mix
6	DFS/BB/BE mix

printbbinfo *(integer)*: prints additional node info ↩

Additional info of log output.

Default: `0`

value	meaning
0	print no additional info
1	print variable selection letter The node and variable selection for the current node are indicated by a two letter code at the end of the log line. The first letter represents the node selection: D for DFS, B for Best Bound, and E for Best Estimate. The second letter represents the variable selection: X for maximum infeasibility, N for minimum infeasibility, and P for pseudo cost.
2	print best estimate

rootsolver *(string)*: solver for the root node ↩

`solver[.n]` Solver is the name of the GAMS NLP solver that should be used in the root node, and `n` is the integer corresponding to optfile for the root node. If `.n` is missing, the optfile treated as zero i.e. the NLP solver will not look for an options file. This SBB option can be used to overwrite the default that uses the NLP solver specified with an `Option NLP = solver;` statement or the default GAMS solver for NLP.

Default: `GAMS NLP solver`

solvelink *(integer)*: Solvelink for GAMS NLP solver ↩

Default: `5`

value	meaning
1	Call GAMS NLP solver via script
2	Call GAMS NLP solver via module
5	Call GAMS NLP solver in memory

subiter *(integer)*: iteration limit for the subsolve ↩

The default for `subiter` passed on through `iterlim`. Similar to subres but sets the iteration limit for solving a node in the B&B tree.

Default: `GAMS iterlim`

subres *(real)*: resource limit for the subsolve ↩

The default for `subres` passed on through `reslim`. Sets the time limit in seconds for solving a node in the B&B tree. If the NLP solver exceeds this limit it is handled like a failure and the node is ignored, or the solvers in the failseq are called.

Default: `GAMS reslim`

subsolver *(string)*: solver for the subproblems ↩

`solver[.n]` Similar to rootsolver but applied to the subnodes.

Default: `GAMS NLP solver`

usercallparmfile *(string)*: Command-line parameter include file used in GAMS command-line calls triggered by BCH ↩

usergdxname *(string)*: the name of the GDX file exported from the solver with the solution at the node ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `bchout.gdx`

usergdxnameinc *(string)*: the name of the GDX file exported from the solver with the incumbent solution ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `bchout_i.gdx`

userheurcall *(string)*: the GAMS command line to call the heuristic ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

userheurfirst *(integer)*: calls the cut generator for the first n nodes ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `10`

userheurfreq *(integer)*: determines the frequency of the cut generator model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `10`

userheurinterval *(integer)*: determines the interval when to apply the multiplier for the frequency of the cut generator model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `100`

userheurmult *(integer)*: determines the multiplier for the frequency of the cut generator model calls ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `2`

userheurnewint *(boolean)*: calls the heuristic if the solver found a new integer feasible solution ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `0`

userheurobjfirst *(integer)*: Similar to UserHeurFirst but only calls the heuristic if the relaxed objective promises an improvement ↩

More info is available in chapter The GAMS Branch-and-Cut-and-Heuristic Facility.

Default: `50`

varsel *(integer)*: variable selection strategy at each node ↩

Variable selection scheme.

Default: `0`

value	meaning
0	automatic
1	maximum integer infeasibility
2	minimum integer infeasibility
3	pseudo costs

5.33.5 The SBB Log File

The SBB Log file (usually directed to the screen) can be controlled with the `loginterval` and `loglevel` options in SBB. It will by default first show the iteration output from the NLP solver that solves the root node. This is followed by output from SBB describing the search tree. An example of this search tree output follows:

```
Root node solved locally optimal.
   Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
      0     0   0   8457.6878   3            -       8457.6878       -
      1     1   1   8491.2869   2            -       8457.6878       -
      2     2   2   8518.1779   1            -       8457.6878       -
*     3     3   3   9338.1020   0    9338.1020       8457.6878  0.1041
      4     2   1      pruned   -    9338.1020       8491.2869  0.0997
 Solution satisfies optcr
 Statistics:
    Iterations    :                 90
    NLP Seconds   :           0.110000
    B&B nodes     :                  3
    MIP solution  :        9338.101979 found in node 3
    Best possible :        8491.286941
    Absolute gap  :         846.815039     optca :  0.000000
    Relative gap  :           0.099728     optcr :  0.100000
    Model Status  :                  8
    Solver Status :                  1

 NLP Solver Statistics
    Total Number of NLP solves  :           7
    Total Number of NLP failures:           0
    Details:         conopt
      # execs             7
      # failures          0
 Terminating.
```

The fields in the log are:

Field	Description
`Node`	The number of the current node. The root node is node 0.
`Act`	The number of active nodes defined as the number of subnodes that have not yet been solved.
`Lev`	The level in the search tree, i.e., the number of branches needed to reach this node.
`Objective`	The objective function value for the node. A numerical value indicates that the node was solved and the objective was good enough for the node to not be ignored. "pruned" indicates that the objective value was worse than the Best Integer value, "infeasible" indicates that the node was Infeasible or Locally Infeasible, and "ignored" indicates that the node could not be solved (see under failseq above).

Field	Description
IInf	The number of integer infeasibilities, i.e. the number of variables that are supposed to be binary or integer that do not satisfy the integrality requirement. Semi continuous variables and SOS variables may also contribute to IInf.
Best Int	The value of the best integer solution found so far. A dash (-) indicates that an integer solution has not yet been found. A star (∗) in column one indicates that the node is integer and that the solution is better than the best yet found.
Best Bound	The minimum value of "Objective" for the subnodes that have not been solved yet (maximum for maximization models). For convex models, Best Bound will increase monotonically. For nonconvex models, Best Bound may decrease, indicating that the Objective value for a node was not a valid lower bound for that node.
Gap	The relative gap between the Best Integer solution and the Best Bound.

The remaining part of the Log file displays various solution statistics similar to those provided by the MIP solvers. This information can also be found in the Solver Status area of the GAMS listing file.

The following Log file shows cases where the NLP solver fails to solve a subnode. The text "ignored" in the Objective field shows the failure, and the values in parenthesis following the Gap field are the Solve and Model status returned by the NLP solver:

```
Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
   0     0   0    6046.0186  12            -       6046.0186       -
   1     1   1   infeasible   -            -       6046.0186       -
   2     0   1    6042.0995  10            -       6042.0995       -
   3     1   2      ignored   -            -       6042.0995       - (4,6)
   4     0   2    5804.5856   8            -       5804.5856       -
   5     1   3      ignored   -            -       5804.5856       - (4,7)
```

The next Log file shows the effect of the infeasseq and failseq options on the model above. CONOPT with options file *conopt.opt* (the default solver and options file pair for this model) considers the first subnode to be locally infeasible. CONOPT, MINOS, and SNOPT, all with no options file, are therefore tried in sequence. In this case, they all declare the node infeasible and it is considered to be infeasible.

In node 3, CONOPT with option file fails but CONOPT without option file finds a Locally Optimal solution, and this solution is then used for further search. The SBB option file for the following run would be:

```
rootsolver conopt.1
subsolver conopt.1
failseq conopt
infeasseq 100 conopt minos snopt
```

The log looks as follows:

```
Root node solved locally optimal.
Node  Act. Lev.  Objective  IInf  Best Int.      Best Bound   Gap  (2 secs)
   0     0   0    6046.0186  12            -       6046.0186       -
conopt.1 reports locally infeasible
Executing conopt
conopt reports locally infeasible
Executing minos
minos reports locally infeasible
```

```
Executing snopt
   1     1    1  infeasible   -            -       6046.0186       -
   2     0    1   6042.0995  10            -       6042.0995       -
conopt.1 failed. 4 TERMINATED BY SOLVER, 7 FEASIBLE SOLUTION
Executing conopt
   3     1    2   4790.2373   8            -       6042.0995       -
   4     2    3   4481.4156   6            -       6042.0995       -
conopt.1 reports locally infeasible
Executing conopt
conopt reports locally infeasible
Executing minos
minos failed. 4 TERMINATED BY SOLVER, 6 INTERMEDIATE INFEASIBLE
Executing snopt
   5     3    4  infeasible   -            -       6042.0995       -
   6     2    4   4480.3778   4            -       6042.0995       -
```

The Log file shows a solver statistic at the end, summarizing how many times an NLP was executed and how often it failed:

```
NLP Solver Statistics
   Total Number of NLP solves  :   45
   Total Number of NLP failures:   13
   Details:     conopt     minos   snopt
     # execs        34         3       8
     # failures      4         3       6
```

The solutions found by the NLP solver to the subproblems in the Branch and Bound may not be the global optima. Therefore, the objective can improve even though we restrict the problem by tightening some bounds. These *jumps* of the objective in the *wrong* direction which might also have an impact on the best bound/possible are reported in a separate statistic:

```
Non convex model!
   # jumps in best bound       :           2
   Maximum jump in best bound  :   20.626587 in node 13
   # jumps to better objective :           2
   Maximum jump in objective   :   20.626587 in node 13
```

5.33.6 Comparison of SBB and other MINLP Solvers

GAMS offers a variety of MINLP solvers including local and global MINLP solver. They implement different algorithms and it is usually unclear which solver performs best. Here we give a brief comparison between SBB and the well known solver DICOPT.

DICOPT is based on the outer approximation method. Initially, the RMINLP model is solved just as in SBB. The model is then linearized around this point and a linear MIP model is solved. The discrete variables are then fixed at the optimal values from the MIP model, and the resulting NLP model is solved. If the NLP model is feasible, we have an integer feasible solution.

The model is linearized again and a new MIP model with both the old and new linearized constraints is solved. The discrete variables are again fixed at the optimal values, and a new NLP model is solved.

The process stops when the MIP model becomes infeasible, when the NLP solution becomes worse, or, in some cases, when bounds derived from the MIP model indicate that it is safe to stop.

DICOPT is based on the assumption that MIP models can be solved efficiently while NLP models can be expensive and difficult to solve. The MIP models try to approximate the NLP model over a large area and solve it using much cheaper linear technology. Ideally, only a few NLPs must be solved.

DICOPT can experience difficulties solving models, if many or all the NLP submodels are infeasible. DICOPT can also have problems if the linearizations used for the MIP model create ill-conditioned models. The MIP models may become very difficult to solve, and the results from the MIP models may be poor as initial values for the NLP models. The linearized constraint used by DICOPT may also exclude certain areas of the feasible space from consideration.

SBB uses different assumptions and works very differently. Most of the work in SBB involves solving NLP models. Since the NLP submodels differ only in one or a few bounds, the assumption is that the NLP models can be solved quickly using a good restart procedure. Since the NLP models differ very little and good initial values are available, the solution process will be fairly reliable compared to the solution process in DICOPT, where initial values of good quality seldom are available. Because search space is reduced based on very different grounds than in DICOPT, other solutions may therefore be explored.

Overall, DICOPT should perform better on models that have a significant and difficult combinatorial part, while SBB may perform better on models that have fewer discrete variables but more difficult nonlinearities (and possibly also on models that are fairly non convex).

5.34 SCIP

SCIP (**S**olving **C**onstraint **I**nteger **P**rograms) is a framework for Constraint Integer Programming oriented towards the needs of Mathematical Programming experts who want to have total control of the solution process and access detailed information down to the guts of the solver. SCIP can also be used as a pure MIP or MINLP solver or as a framework for branch-cut-and-price. Within GAMS, the MIP and MINLP solving facilities of SCIP are available.

For more detailed information, we refer to [4] [1] [2] [21] [20] [74] [127] [88] [89] [75] [22] [192] [191] [204] and the SCIP web site.

GAMS/SCIP uses CPLEX, if licensed, and otherwise SOPLEX [206] as LP solver, the COIN-OR Interior Point Optimizer IPOPT [197] as nonlinear solver, and CppAD to compute derivatives of nonlinear functions.

SCIP supports continuous, binary, integer, semi-continuous, semi-integer variables, indicator constraints, special ordered sets, and branching priorities for discrete variables.

5.34.1 Usage

The following statement can be used inside your GAMS program to specify using SCIP

```
Option MIP = SCIP;     { or QCP or NLP or MIQCP or MINLP or ... }
```

The above statement should appear before the Solve statement. If SCIP was specified as the default solver during GAMS installation, the above statement is not necessary.

5.34.1.1 Specification of SCIP Options

GAMS/SCIP supports the GAMS parameters reslim, iterlim, nodlim, optca, optcr, and workspace. Further, the option threads can be used to control the number of threads used in the MILP presolver, the linear algebra routines (e.g., in IPOPT), and for solving LPs if CPLEX' barrier solver is used. Setting threads to 0 (the default) will enable the automatic choice of the number of threads in the MILP presolver and LP solver, but does not enable multithreading for linear algebra routines at the moment.

Options can be specified by a SCIP options file. A SCIP options file consists of one option or comment per line. A pound sign (#) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs). Further, string values have to be enclosed in quotation marks.

A small example for a scip.opt file is:

```
propagating/probing/maxprerounds = 0
separating/maxrounds             = 0
separating/maxroundsroot         = 0
```

It causes GAMS/SCIP to disable probing during presolve and to turn off all cut generators.

5.34.1.2 Specification of Indicators

Indicators are a modeling tool to specify that certain equations in a model must only be satisfied if certain binary variables take a specified value. Indicators are not supported by the GAMS language, but can be passed to SCIP via a separate file, see Indicator Constraints for more details on its syntax. The name of that file is specified via the option gams/indicatorfile in a SCIP option file. Currently, indicators can only be used for linear equations.

5.34.2 Special Features

5.34.2.1 SCIP interactive shell

The interactive shell in SCIP is a powerful tool that allows the user to display various information (e.g., branching statistics, presolved model), load emphasis settings, interrupt a solve to change parameters or trigger a restart, write the model in various file formats, start SCIPs solution counter, and many more things.

When setting the option gams/interactive to a nonempty string, the GAMS/SCIP interface opens the interactive shell of SCIP after having load the GAMS problem and parameters and passes the value of the gams/interactive parameter to the SCIP interactive shell.

By default, SCIP behaves as if `gams/interactive` has been set to `"optimize write gamssol quit"`, that is, SCIP is requested to solve the problem, then to pass the solution back to GAMS, and to quit.

An example use of the SCIP interactive shell feature via GAMS is to add the following line to your SCIP options file:

```
gams/interactive = "write prob orig.lp presolve write transprob presol.mps opt write gamssol quit"
```

This instructs SCIP to write the original problem to the file `orig.lp` in LP format, to presolve the instance, to write the presolved problem to the file `presolved.mps` in MPS format, to solve the instance, to write the solution out to GAMS, and to finish.

By omitting the `quit` command, SCIPs interactive shell remains open and awaits user input. The command `help` prints a list of available commands. Note, that on Windows, GAMS need to be called with the option interactivesolver enabled to allow user input for the solver process.

A tutorial on using the SCIP shell is available at https://www.scipopt.org/doc/html/SHELL.php.

5.34.2.2 Emphasis Settings

SCIP includes various emphasis settings, which are predefined values for a set of SCIP parameters. Such predefined settings are available for setting the effort that SCIP should spend for, e.g., presolving, separation, or heuristics.

The emphasis settings are not available as single parameters, but can be set via SCIPs interactive shell. E.g., writing `set heuristics emphasis` in the shell displays the available emphasis settings for heuristics (`aggressive`, `fast`, `off`) and expects the user to input which setting to use. Further, general emphasis settings are available in the `set emphasis` menu, some of them giving predefined settings similar to the CPLEX option mipemphasis.

Via the gams/interactive option, c.f. Section SCIP interactive shell, emphasis settings can be activated via a SCIP options file. For example, adding the option

```
gams/interactive = "set emphasis feasibility set loadgams optimize write gamssol quit"
```

instructs SCIP to load the emphasis setting `feasibility` prior to optimizing the model and passing the solution back to GAMS. Note, that setting one of the emphasis settings `feasibility`, `hardlp`, and `optimality` resets all previously set parameters to their default values, which includes the ones that are set by the GAMS/SCIP interface or which were loaded from a SCIP options file. Therefore, the command `set loadgams` has been used above to restore these parameter settings.

The following emphasis settings are available in SCIP:

shell command	purpose
`set emphasis easycip`	use for easy problems
`set emphasis feasibility`	emphasize finding a feasible solution
`set emphasis hardlp`	use for problems with a hard LP
`set emphasis optimality`	emphasize proving optimality
`set emphasis numerics`	increase numerical stability
`set emphasis cpsolver`	setup a constraint programming like search
`set heuristic emphasis aggressive`	use primal heuristics aggressively
`set heuristic emphasis fast`	use only fast primal heuristics
`set heuristic emphasis off`	disable all primal heuristics
`set presolving emphasis aggressive`	do aggressive presolving
`set presolving emphasis fast`	use only fast presolving steps
`set presolving emphasis off`	disable presolving
`set separating emphasis aggressive`	use cutting plane separators aggressively
`set separating emphasis fast`	use only fast cutting plane separators
`set separating emphasis off`	disable all cutting plane separators

5.34.2.3 Starting point

Using the completesol heuristic, SCIP can try to find a feasible solution based on values given by the user for all or some of the variables. The values need to be specified as variable levels in the GAMS model. The heuristic solves a copy of the problem where variables for which values have been provided are restricted to be close to that value. When an integral value is specified for a binary or integer variable, the variable is fixed to that value. See also Section 2.3.2 in [127] for a more detailed description of the heuristic.

For which variables the level values are passed from GAMS to SCIP is controlled by the parameter gams/mipstart. The parameter values have the following meaning:

- 0: do not pass any variable values to SCIP, the heuristic will not run
- 1: pass values for all binary and integer variables to SCIP and let SCIP try to find a feasible solution in its neighborhood by using the completesol heuristic
- 2 (default): pass values for all variables to SCIP and let SCIP check feasibility of the given solution, the heuristic will not run
- 3: pass values for all variables to SCIP and let SCIP try to find a feasible solution in the neighborhood by using the completesol heuristic
- 4: pass values for all binary and integer variables to SCIP which fractionality is at most the value of GAMS option tryint (thus, with default tryint=0, only for variables with integral values, the value is passed to SCIP) and let SCIP try to find a feasible solution in the neighborhood by using the completesol heuristic

Note, that the completesol heuristic will not run if there are too many variables with unknown values. This behavior can be adjusted by setting parameter heuristics/completesol/maxunknownrate.

5.34.2.4 Decomposition Information

SCIP potentially can make use of a user-provided `problem decomposition`, e.g., in primal heuristics PADM or GINS. Note, that SCIP's Benders Decomposition is not available with GAMS at the moment. A GAMS user can inform SCIP about such a decomposition by using the .stage variable suffix.

In an attempt to following the same convention on interpreting the `.stage` variable suffix as the GAMS/CPLEX link, variables which `.stage` suffix is set to an integral value greater than 0 are assigned to a block with this number. Variables which `.stage` suffix is set to a fractional value or a value smaller than 1 are assigned to no block.

Note, that the default value for the `.stage` suffix is 1. Therefore, GAMS/SCIP only passes decomposition information to SCIP if at least one variable has its `.stage` suffix set to a value different than 1. Moreover, in order to receive the `.stage` values, `scaleOpt` and `priorOpt` must be left at their default of 0.

5.34.2.5 Solution Pool

When SCIP solves a problem, it may find several solutions, whereof only the best one is available to the GAMS user via the variable level values in the GAMS model. If the option gams/dumpsolutions is specified, then all alternative solutions found by SCIP are written into GDX files and an index file with the name given by the this option is written. If the option gams/dumpsolutionsmerged is specified, then all alternative solutions found by SCIP are written into a single GDX file, which name is given by the this option.

The GAMS testlib model **dumpsol** shows an example use for this option.

5.34.2.6 Solving process tracing

The option gams/solvetrace/file can be used to specify the name of a file where information about the progress of the branch-and-bound tree search in SCIP is stored. The file is created and updated during the solution process, so it may also be used to monitor the progress of SCIP while it still solves the model.

New entries are written periodically, depending on how many nodes have been processed or how much time has been elapsed since the last entry was written. Each entry contains information on the current primal and dual bound.

5.34.2.7 Notes on solving MINLPs

SCIP includes capabilities to handle nonlinear functions that are specified via algebraic expressions. Therefore, neither external/extrinsic functions nor all GAMS mathematical functions are supported.

Nonconvex MINLPs are solved via a spatial branch-and-bound algorithm using linear relaxations. The tightness of this relaxation depends heavily on the variable bounds, thus tight bounds for the nonlinear variables are crucial for SCIP.

Convex MINLPs are much easier to solve for SCIP, provided it recognizes the convexity of the model. The convexity check implemented in SCIP may not give a conclusive answer in all cases. However, setting the option constraints/nonlinear/assumeconvex to `TRUE` can be used to tell SCIP that it should assume all nonlinear constraints to be of convex type. This may help to improve solving times for convex MINLPs considerably in some cases. However, it can also deteriorate performance as it prevents the use of certain extended formulations.

Another feature that can be especially useful for convex MINLPs is to enable the generation of cuts in feasible solutions, since these supporting hyperplanes can be strong. At the moment, this is only enabled for purely continuous nonlinear programs (QCPs and NLPs), but can be adjusted by parameter constraints/nonlinear/linearizeheursol. In this connection, it may also be beneficial to set separating/poolfreq to 1.

5.34.3 Components

In the following, we list components that are available in SCIP together with some common properties. Many of these properties can be modified with corresponding parameters.

5.34.3.1 Branching Rules

branching rule	priority	maxdepth	maxbounddist	description
relpscost	10000	-1	100	reliability branching on pseudo cost values
pscost	2000	-1	100	branching on pseudo cost values
inference	1000	-1	100	inference history branching
mostinf	100	-1	100	most infeasible branching
leastinf	50	-1	100	least infeasible branching
distribution	0	-1	100	branching rule based on variable influence on cumulative normal distribution of row activities
fullstrong	0	-1	100	full strong branching
cloud	0	-1	100	branching rule that considers several alternative LP optima
lookahead	0	-1	100	full strong branching over multiple levels
multaggr	0	-1	100	fullstrong branching on fractional and multi-aggregated variables
allfullstrong	-1000	-1	100	all variables full strong branching
vanillafullstrong	-2000	-1	100	vanilla full strong branching
random	-100000	-1	100	random variable branching
nodereopt	-9000000	-1	100	branching rule for node reoptimization

See https://www.scipopt.org/doc/html/BRANCH.php for a detailed description of the branching rule properties.

5.34.3.2 Conflict Handler

conflict handler	priority	description
logicor	800000	conflict handler creating logic or constraints
setppc	700000	conflict handler creating set covering constraints
indicatorconflict	200000	replace slack variables and generate logicor constraints
linear	-1000000	conflict handler creating linear constraints
bounddisjunction	-3000000	conflict handler creating bound disjunction constraints

5.34.3.3 Constraint Handler

constraint handler	checkprio	enfoprio	sepaprio	sepafreq	propfreq	eagerfreq	presolvetimings	description
integral	0	0	0	-1	-1	-1	always	integrality constraint
SOS1	-10	100	1000	10	1	100	medium	SOS1 constraint handler
SOS2	-10	100	10	0	1	100	fast	SOS2 constraint handler
varbound	-500000	-500000	900000	0	1	100	fast medium	variable bounds lhs <= x + c*y <= rhs, x non-binary, y non-continuous
knapsack	-600000	-600000	600000	0	1	100	always	knapsack constraint of the form a^T x <= b, x binary and a >= 0
setppc	-700000	-700000	700000	0	1	100	always	set partitioning / packing / covering constraints

constraint handler	checkprio	enfoprio	sepaprio	sepafreq	propfreq	eagerfreq	presolvetimings	description
or	-850000	-850000	850000	0	1	100	medium	constraint handler for or constraints: r = or(x1, ..., xn)
and	-850100	-850100	850100	1	1	100	fast exhaustive	constraint handler for AND-constraints: r = and(x1, ..., xn)
xor	-850200	-850200	850200	0	1	100	always	constraint handler for xor constraints: r = xor(x1, ..., xn)
linear	-1000000	-1000000	100000	0	1	100	fast exhaustive	linear constraints of the form lhs <= a^T x <= rhs
orbisack	-1005200	-1005200	40100	5	5	-1	exhaustive	symmetry breaking constraint handler for orbisacks
orbitope	-1005200	-1005200	40100	-1	1	-1	medium	symmetry breaking constraint handler relying on (partitioning/packing) orbitopes

constraint handler	check-prio	enfo-prio	sepa-prio	sepa-freq	propfreq	eager-freq	presol-vetim-ings	de-scrip-tion
symresack	-1005200	-1005200	40100	5	5	-1	exhaustive	symmetry breaking constraint handler relying on sym-resacks
logicor	-2000000	-2000000	10000	0	1	100	always	logic or con-straints
bounddisjunction	-3000000	-3000000	0	-1	1	100	fast	bound disjunc-tion con-straints
nonlinear	-4000010	-60	10	1	1	100	always	handler for non-linear con-straints specified by al-gebraic expres-sions
indicator	-6000000	-100	10	10	1	100	fast	indicator con-straint handler
components	-9999999	0	0	-1	1	-1		independent compo-nents con-straint handler

See https://www.scipopt.org/doc/html/CONS.php for a detailed description of the constraint handler properties.

5.34.3.4 Display Columns

display column	header	position	width	priority	status	description
solfound		0	1	80000	auto	letter that indi-cates the heuris-tic which found the solution
time	time	50	5	4000	auto	total solution time
nnodes	node	100	7	100000	auto	number of pro-cessed nodes

display column	header	position	width	priority	status	description
nodesleft	left	200	7	90000	auto	number of unprocessed nodes
nrank1nodes	rank1	500	7	40000	off	current number of rank1 nodes left
nnodesbelowinc	nbInc	550	6	40000	off	current number of nodes with an estimate better than the current incumbent
lpiterations	LP iter	1000	7	30000	auto	number of simplex iterations
lpavgiterations	LP it/n	1400	7	25000	auto	average number of LP iterations since the last output line
lpcond	LP cond	1450	7	0	auto	estimate on condition number of LP solution
memused	umem	1500	5	0	auto	total number of bytes used in block memory
memtotal	mem/heur	1500	8	20000	auto	total number of bytes in block memory or the creator name when a new incumbent solution was found
depth	depth	2000	5	500	auto	depth of current node
maxdepth	mdpt	2100	5	5000	auto	maximal depth of all processed nodes
plungedepth	pdpt	2200	5	10	auto	current plunging depth
nfrac	frac	2500	5	700	auto	number of fractional variables in the current solution
nexternbranchcands	extbr	2600	5	650	auto	number of extern branching variables in the current node
vars	vars	3000	5	3000	auto	number of variables in the problem
conss	cons	3100	5	3100	auto	number of globally valid constraints in the problem
curconss	ccons	3200	5	600	auto	number of enabled constraints in current node

display column	header	position	width	priority	status	description
curcols	cols	3300	5	800	auto	number of LP columns in current node
currows	rows	3400	5	900	auto	number of LP rows in current node
cuts	cuts	3500	5	2100	auto	total number of cuts applied to the LPs
separounds	sepa	3600	4	100	auto	number of separation rounds performed at the current node
poolsize	pool	3700	5	50	auto	number of LP rows in the cut pool
conflicts	confs	4000	5	2000	auto	total number of conflicts found in conflict analysis
strongbranchs	strbr	5000	5	1000	auto	total number of strong branching calls
pseudoobj	pseudoobj	6000	14	300	auto	current pseudo objective value
lpobj	lpobj	6500	14	300	auto	current LP objective value
curdualbound	curdualbound	7000	14	400	auto	dual bound of current node
estimate	estimate	7500	14	200	auto	estimated value of feasible solution in current node
avgdualbound	avgdualbound	8000	14	40	auto	average dual bound of all unprocessed nodes
dualbound	dualbound	9000	14	70000	auto	current global dual bound
primalbound	primalbound	10000	14	80000	auto	current primal bound
cutoffbound	cutoffbound	10100	14	10	auto	current cutoff bound
gap	gap	20000	8	60000	auto	current (relative) gap using |primal-dual|/MIN(|dual|,|primal|)
primalgap	primgap	21000	8	20000	off	current (relative) gap using |primal-dual|/|primal|
nsols	nsols	30000	5	0	auto	current number of solutions found

display column	header	position	width	priority	status	description
completed	compl.	30100	8	110000	auto	completion of search in percent (based on tree size estimation)
nobjleaves	objleav	31000	7	0	auto	current number of encountered objective limit leaves
ninfeasleaves	infleav	32000	7	0	auto	number of encountered infeasible leaves
sols	sols	100000	7	110000	off	number of detected feasible solutions
feasST	feasST	110000	6	110000	off	number of detected non trivial feasible subtrees

See https://www.scipopt.org/doc/html/DISP.php for a detailed description of the display column properties.

5.34.3.5 Nonlinear Handler

nonlinear handler	enabled	detect priority	enforce priority	description
default	1	0	0	default handler for expressions
convex	1	50	50	handler that identifies and estimates convex expressions
concave	1	40	40	handler that identifies and estimates concave expressions
bilinear	1	-10	-10	bilinear handler for expressions
perspective	1	-20	125	perspective handler for expressions
quadratic	1	1	100	handler for quadratic expressions
quotient	1	20	20	nonlinear handler for quotient expressions
soc	1	100	100	nonlinear handler for second-order cone structures

5.34.3.6 Node Selectors

node selector	standard priority	memsave priority	description
estimate	200000	100	best estimate search

node selector	standard priority	memsave priority	description
bfs	100000	0	best first search
hybridestim	50000	50	hybrid best estimate / best bound search
restartdfs	10000	50000	depth first search with periodical selection of the best node
uct	10	0	node selector which balances exploration and exploitation
dfs	0	100000	depth first search
breadthfirst	-10000	-1000000	breadth first search

See https://www.scipopt.org/doc/html/NODESEL.php for a detailed description of the node selector properties.

5.34.3.7 Presolvers

presolver	priority	timing	maxrounds	description
milp	9999999	medium	-1	MILP specific presolving methods
trivial	9000000	fast	-1	round fractional bounds on integers, fix variables with equal bounds
boundshift	7900000	fast	0	converts variables with domain [a,b] to variables with domain [0,b-a]
inttobinary	7000000	fast	-1	converts integer variables with domain [a,a+1] to binaries
convertinttobin	6000000	fast	0	converts integer variables to binaries
gateextraction	1000000	exhaustive	-1	presolver extracting gate(and)-constraints
qpkktref	-1	medium	0	adds KKT conditions to (mixed-binary) quadratic programs
dualcomp	-50	exhaustive	-1	compensate single up-/downlocks by singleton continuous variables
stuffing	-100	exhaustive	0	fix redundant singleton continuous variables
domcol	-1000	exhaustive	-1	dominated column presolver
tworowbnd	-2000	exhaustive	0	do bound tigthening by using two rows
dualinfer	-3000	exhaustive	0	exploit dual information for fixings and side changes
implics	-10000	medium	-1	implication graph aggregator
dualagg	-12000	exhaustive	0	aggregate variables by dual arguments
sparsify	-24000	exhaustive	-1	eliminate non-zero coefficients
dualsparsify	-240000	exhaustive	-1	eliminate non-zero coefficients
redvub	-9000000	exhaustive	0	detect redundant variable bound constraints

See https://www.scipopt.org/doc/html/PRESOL.php for a detailed description of the presolver properties.

5.34.3.8 Primal Heuristics

primal heuristic	type	priority	freq	freqoffset	description
actconsdiving	d	-1003700	-1	5	LP diving heuristic that chooses fixings w.r.t. the active constraints
adaptivediving	d	-70000	5	3	diving heuristic that selects adaptively between the existing, public divesets
bound	p	-1107000	-1	0	heuristic which fixes all integer variables to a bound and solves the remaining LP
clique	p	5000	0	0	LNS heuristic using a clique partition to restrict the search neighborhood
coefdiving	d	-1001000	-1	1	LP diving heuristic that chooses fixings w.r.t. the matrix coefficients
completesol	L	0	0	0	primal heuristic trying to complete given partial solutions
conflictdiving	d	-1000100	10	0	LP diving heuristic that chooses fixings w.r.t. conflict locks
crossover	L	-1104000	30	0	LNS heuristic that fixes all variables that are identic in a couple of solutions
dins	L	-1105000	-1	0	distance induced neighborhood search by Ghosh
distributiondiving	d	-1003300	10	3	Diving heuristic that chooses fixings w.r.t. changes in the solution density
dps	L	75000	-1	0	primal heuristic for decomposable MIPs
dualval	L	-10	-1	0	primal heuristic using dual values
farkasdiving	d	-900000	10	0	LP diving heuristic that tries to construct a Farkas-proof
feaspump	o	-1000000	20	0	objective feasibility pump 2.0
fixandinfer	p	-500000	-1	0	iteratively fixes variables and propagates inferences
fracdiving	d	-1003000	10	3	LP diving heuristic that chooses fixings w.r.t. the fractionalities
gins	L	-1103000	20	8	gins works on k-neighborhood in a variable-constraint graph
guideddiving	d	-1007000	10	7	LP diving heuristic that chooses fixings in direction of incumbent solutions
zeroobj	L	100	-1	0	heuristic trying to solve the problem without objective
indicator	L	-20200	1	0	indicator heuristic to create feasible solutions from values for indicator variables
intdiving	d	-1003500	-1	9	LP diving heuristic that fixes binary variables with large LP value to one
intshifting	r	-10000	10	0	LP rounding heuristic with infeasibility recovering and final LP solving
linesearchdiving	d	-1006000	10	6	LP diving heuristic that chooses fixings following the line from root solution to current solution
localbranching	L	-1102000	-1	0	local branching heuristic by Fischetti and Lodi

primal heuristic	type	priority	freq	freqoffset	description
locks	p	3000	0	0	heuristic that fixes variables based on their rounding locks
lpface	L	-1104010	15	0	LNS heuristic that searches the optimal LP face inside a sub-MIP
alns	L	-1100500	20	0	Large neighborhood search heuristic that orchestrates the popular neighborhoods Local Branching, RINS, RENS, DINS etc.
nlpdiving	d	-1003010	10	3	NLP diving heuristic that chooses fixings w.r.t. the fractionalities
mutation	L	-1103010	-1	8	mutation heuristic randomly fixing variables
multistart	L	-2100000	0	0	multistart heuristic for convex and nonconvex MINLPs
mpec	d	-2050000	50	0	regularization heuristic for convex and nonconvex MINLPs
objpscostdiving	o	-1004000	20	4	LP diving heuristic that changes variable's objective values instead of bounds, using pseudo costs as guide
octane	r	-1008000	-1	0	octane primal heuristic for pure {0;1}-problems based on Balas et al.
ofins	L	60000	0	0	primal heuristic for reoptimization, objective function induced neighborhood search
oneopt	i	-20000	1	0	1-opt heuristic which tries to improve setting of single integer variables
padm	L	70000	0	0	penalty alternating direction method primal heuristic
proximity	L	-2000000	-1	0	heuristic trying to improve the incumbent by an auxiliary proximity objective function
pscostdiving	d	-1002000	10	2	LP diving heuristic that chooses fixings w.r.t. the pseudo cost values
randrounding	r	-200	20	0	fast LP rounding heuristic
rens	L	-1100000	0	0	LNS exploring fractional neighborhood of relaxation's optimum
reoptsols	p	40000	0	0	primal heuristic updating solutions found in a previous optimization round
repair	L	-20	-1	0	tries to repair a primal infeasible solution
rins	L	-1101000	25	0	relaxation induced neighborhood search by Danna, Rothberg, and Le Pape
rootsoldiving	o	-1005000	20	5	LP diving heuristic that changes variable's objective values using root LP solution as guide
rounding	r	-1000	1	0	LP rounding heuristic with infeasibility recovering
shiftandpropagate	p	1000	0	0	Pre-root heuristic to expand an auxiliary branch-and-bound tree and apply propagation techniques

primal heuristic	type	priority	freq	freqoffset	description
shifting	r	-5000	10	0	LP rounding heuristic with infeasibility recovering also using continuous variables
simplerounding	r	-30	1	0	simple and fast LP rounding heuristic
subnlp	L	-2000010	1	0	primal heuristic that performs a local search in an NLP after fixing integer variables and presolving
trivial	t	10000	0	0	start heuristic which tries some trivial solutions
trivialnegation	p	39990	0	0	negate solution entries if an objective coefficient changes the sign, enters or leaves the objective.
trustregion	L	-1102010	-1	0	LNS heuristic for Benders' decomposition based on trust region methods
trysol	t	-3000010	1	0	try solution heuristic
twoopt	i	-20100	-1	0	primal heuristic to improve incumbent solution by flipping pairs of variables
undercover	L	-1110000	0	0	solves a sub-CIP determined by a set covering approach
vbounds	p	2500	0	0	LNS heuristic uses the variable lower and upper bounds to determine the search neighborhood
veclendiving	d	-1003100	10	4	LP diving heuristic that rounds variables with long column vectors
zirounding	r	-500	1	0	LP rounding heuristic as suggested by C. Wallace taking row slacks and bounds into account

See https://www.scipopt.org/doc/html/HEUR.php for a detailed description of the primal heuristic properties.

5.34.3.9 Propagators

propagator	propprio	freq	presolveprio	presolvetiming	description
rootredcost	10000000	1	0	always	reduced cost strengthening using root node reduced costs and the cutoff bound
dualfix	8000000	0	8000000	fast	roundable variables dual fixing
genvbounds	3000000	1	-2000000	fast	generalized variable bounds propagator
vbounds	3000000	1	-90000	medium exhaustive	propagates variable upper and lower bounds
pseudoobj	3000000	1	6000000	fast	pseudo objective function propagator
redcost	1000000	1	0	always	reduced cost strengthening propagator

propagator	propprio	freq	presolveprio	presolvetiming	description
probing	-100000d	-1	-100000	exhaustive	probing propagator on binary variables
symmetry	-1000000	1	-10000000	exhaustive	propagator for handling symmetry
obbt	-1000000d	0	0	always	optimization-based bound tightening propagator
nlobbt	-1100000d	-1	0	always	propagator template

See https://www.scipopt.org/doc/html/PROP.php for a detailed description of the propagator properties.

5.34.3.10 Separators

separator	priority	freq	bounddist	description
closecuts	1000000	-1	1	closecuts meta separator
rlt	10	0	1	reformulation-linearization-technique separator
disjunctive	10d	0	0	disjunctive cut separator
gauge	0	-1	1	gauge separator
interminor	0	-1	1	intersection cuts separator to ensure that 2x2 minors of X (= xx') have determinant 0
convexproj	0d	-1	1	separate at projection of point onto convex region
minor	0	10	1	separator to ensure that 2x2 principal minors of X - xx' are positive semi-definite
impliedbounds	-50	10	1	implied bounds separator
mixing	-50	10	1	mixing inequality separator
intobj	-100	-1	0	integer objective value separator
gomory	-1000	10	1	separator for Gomory mixed-integer and strong CG cuts from LP tableau rows
cgmip	-1000	-1	0	Chvatal-Gomory cuts via MIPs separator
aggregation	-3000	10	1	aggregation heuristic for complemented mixed integer rounding cuts and flowcover cuts
clique	-5000	0	0	clique separator of stable set relaxation
zerohalf	-6000	10	1	{0,1/2}-cuts separator
mcf	-10000	0	0	multi-commodity-flow network cut separator
eccuts	-13000	-1	1	separator for edge-concave functions
oddcycle	-15000	-1	1	odd cycle separator
strongcg	-100000	10	0	separator for strong CG cuts
gomorymi	-100000	10	0	separator for Gomory mixed-integer cuts
knapsackcover	-100000	10	0	separator for knapsack cover cuts
flowcover	-100000	10	0	separator for flowcover cuts
cmir	-100000	10	0	separator for cmir cuts
rapidlearning	-1200000	5	1	rapid learning heuristic and separator

See https://www.scipopt.org/doc/html/SEPA.php for a detailed description of the separator properties.

5.34.4 List of SCIP Options

SCIP supports a large set of options. In the following, we give a detailed list of all SCIP options.

5.34.4.1 gams

Option	Description	Default
gams/dumpsolutions	name of solutions index gdx file for writing all alternate solutions Range: string	
gams/dumpsolutionsmerged	name of gdx file for writing all alternate solutions into a single file Range: string	
gams/indicatorfile	name of GAMS options file that contains definitions on indicators Range: string	
gams/infbound	value to use for variable bounds that are missing or exceed numerics/infinity Range: [0, ∞]	∞
gams/interactive	command to be issued to the SCIP shell instead of issuing a solve command Range: string	
gams/mipstart	how to handle initial variable levels, see also section Starting point Range: {0, ..., 4}	2

5.34.4.2 gams/solvetrace

Option	Description	Default
gams/solvetrace/file	name of file where to write branch-and-bound trace information too Range: string	
gams/solvetrace/nodefreq	frequency in number of nodes when to write branch-and-bound trace information, 0 to disable Range: {0, ..., ∞}	100
gams/solvetrace/timefreq	frequency in seconds when to write branch-and-bound trace information, 0.0 to disable Range: [0, ∞]	5

5.34.4.3 branching

Option	Description	Default
branching/clamp	minimal relative distance of branching point to bounds when branching on a continuous variable Range: [0, 0.5]	0.2
branching/delaypscostupdate	should updating pseudo costs for continuous variables be delayed to the time after separation? Range: boolean	1

Option	Description	Default
branching/divingpscost	should pseudo costs be updated also in diving and probing mode? Range: boolean	1
branching/lpgainnormalize	strategy for normalization of LP gain when updating pseudocosts of continuous variables (divide by movement of 'l'p value, reduction in 'd'omain width, or reduction in domain width of 's'ibling) Range: d, l, s	s
branching/midpull	fraction by which to move branching point of a continuous variable towards the middle of the domain; a value of 1.0 leads to branching always in the middle of the domain Range: [0, 1]	0.75
branching/midpullreldomtrig	multiply midpull by relative domain width if the latter is below this value Range: [0, 1]	0.5
branching/preferbinary	should branching on binary variables be preferred? Range: boolean	0
Options for expert users		
branching/checksol	should LP solutions during strong branching with propagation be checked for feasibility? Range: boolean	1
branching/firstsbchild	child node to be regarded first during strong branching (only with propagation): 'u'p child, 'd'own child, 'h'istory-based, or 'a'utomatic Range: a, d, u, h	a
branching/forceallchildren	should all strong branching children be regarded even if one is detected to be infeasible? (only with propagation) Range: boolean	0
branching/roundsbsol	should LP solutions during strong branching with propagation be rounded? (only when checksbsol=TRUE) Range: boolean	1
branching/scorefac	branching score factor to weigh downward and upward gain prediction in sum score function Range: [0, 1]	0.167
branching/scorefunc	branching score function ('s'um, 'p'roduct, 'q'uotient) Range: s, p, q	p
branching/sumadjustscore	score adjustment near zero by adding epsilon (TRUE) or using maximum (FALSE) Range: boolean	0

5.34.4.4 branching/allfullstrong

Option	Description	Default
branching/allfullstrong/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/allfullstrong/maxdepth	maximal depth level, up to which branching rule <allfullstrong> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1

Option	Description	Default
branching/allfullstrong/priority	priority of branching rule <allfullstrong> Range: {-536870912, ..., 536870911}	-1000

5.34.4.5 branching/cloud

Option	Description	Default
branching/cloud/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/cloud/maxdepth	maximal depth level, up to which branching rule <cloud> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/cloud/maxdepthunion	maximum depth for the union Range: {0, ..., 65000}	65000
branching/cloud/maxpoints	maximum number of points for the cloud (-1 means no limit) Range: {-1, ..., ∞}	-1
branching/cloud/minsuccessrate	minimum success rate for the cloud Range: [0, 1]	0
branching/cloud/minsuccessunion	minimum success rate for the union Range: [0, 1]	0
branching/cloud/onlyF2	should only F2 be used? Range: boolean	0
branching/cloud/priority	priority of branching rule <cloud> Range: {-536870912, ..., 536870911}	0
branching/cloud/usecloud	should a cloud of points be used? Range: boolean	1
branching/cloud/useunion	should the union of candidates be used? Range: boolean	0

5.34.4.6 branching/distribution

Option	Description	Default
branching/distribution/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/distribution/maxdepth	maximal depth level, up to which branching rule <distribution> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/distribution/priority	priority of branching rule <distribution> Range: {-536870912, ..., 536870911}	0
Options for expert users		
branching/distribution/onlyactiverows	should only rows which are active at the current node be considered? Range: boolean	0

Option	Description	Default
branching/distribution/scoreparam	the score;largest 'd'ifference, 'l'owest cumulative probability,'h'ighest c.p., 'v'otes lowest c.p., votes highest c.p.('w') Range: d, h, l, v, w	v
branching/distribution/weightedscore	should the branching score weigh up- and down-scores of a variable Range: boolean	0

5.34.4.7 branching/fullstrong

Option	Description	Default
branching/fullstrong/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/fullstrong/maxdepth	maximal depth level, up to which branching rule <fullstrong> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/fullstrong/priority	priority of branching rule <fullstrong> Range: {-536870912, ..., 536870911}	0
Options for expert users		
branching/fullstrong/forcestrongbranch	should strong branching be applied even if there is just a single candidate? Range: boolean	0
branching/fullstrong/maxproprounds	maximum number of propagation rounds to be performed during strong branching before solving the LP (-1: no limit, -2: parameter settings) Range: {-3, ..., ∞}	-2
branching/fullstrong/probingbounds	should valid bounds be identified in a probing-like fashion during strong branching (only with propagation)? Range: boolean	1
branching/fullstrong/reevalage	number of intermediate LPs solved to trigger reevaluation of strong branching value for a variable that was already evaluated at the current node Range: {0, ..., ∞}	10

5.34.4.8 branching/inference

Option	Description	Default
branching/inference/conflictprio	priority value for using conflict weights in lex. order Range: {0, ..., ∞}	1
branching/inference/cutoffprio	priority value for using cutoff weights in lex. order Range: {0, ..., ∞}	1

Option	Description	Default
branching/inference/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/inference/maxdepth	maximal depth level, up to which branching rule <inference> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/inference/priority	priority of branching rule <inference> Range: {-536870912, ..., 536870911}	1000
branching/inference/useweightedsum	should a weighted sum of inference, conflict and cutoff weights be used? Range: boolean	1
Options for expert users		
branching/inference/conflictweight	weight in score calculations for conflict score Range: [0, ∞]	1000
branching/inference/cutoffweight	weight in score calculations for cutoff score Range: [0, ∞]	1
branching/inference/fractionals	should branching on LP solution be restricted to the fractional variables? Range: boolean	1
branching/inference/inferenceweight	weight in score calculations for inference score Range: real	1
branching/inference/reliablescore	weight in score calculations for conflict score Range: [0, ∞]	0.001

5.34.4.9 branching/leastinf

Option	Description	Default
branching/leastinf/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/leastinf/maxdepth	maximal depth level, up to which branching rule <leastinf> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/leastinf/priority	priority of branching rule <leastinf> Range: {-536870912, ..., 536870911}	50

5.34.4.10 branching/lookahead

Option	Description	Default
branching/lookahead/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1

Option	Description	Default
branching/lookahead/maxdepth	maximal depth level, up to which branching rule <lookahead> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/lookahead/priority	priority of branching rule <lookahead> Range: {-536870912, ..., 536870911}	0
Options for expert users		
branching/lookahead/abbreviated	toggles the abbreviated LAB. Range: boolean	1
branching/lookahead/abbrevpseudo	if abbreviated: Use pseudo costs to estimate the score of a candidate. Range: boolean	0
branching/lookahead/addbinconsrow	should binary constraints be added as rows to the base LP? (0: no, 1: separate, 2: as initial rows) Range: {0, ..., 2}	0
branching/lookahead/addclique	add binary constraints with two variables found at the root node also as a clique Range: boolean	0
branching/lookahead/addnonviocons	should binary constraints, that are not violated by the base LP, be collected and added? Range: boolean	0
branching/lookahead/applychildbounds	should bounds known for child nodes be applied? Range: boolean	0
branching/lookahead/deeperscoringfunction	scoring function to be used at deeper levels Range: d, f, s, w, l, c, r, x	x
branching/lookahead/enforcemaxdomreds	should the maximum number of domain reductions maxnviolateddomreds be enforced? Range: boolean	0
branching/lookahead/filterbymaxgain	should lookahead branching only be applied if the max gain in level 1 is not uniquely that of the best candidate? Range: boolean	0
branching/lookahead/level2avgscore	should the average score be used for uninitialized scores in level 2? Range: boolean	0
branching/lookahead/level2zeroscore	should uninitialized scores in level 2 be set to 0? Range: boolean	0
branching/lookahead/maxncands	if abbreviated: The max number of candidates to consider at the node. Range: {1, ..., ∞}	4
branching/lookahead/maxndeepercands	if abbreviated: The max number of candidates to consider per deeper node. Range: {0, ..., ∞}	2
branching/lookahead/maxnviolatedbincons	how many binary constraints that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted] Range: {0, ..., ∞}	0

Option	Description	Default
branching/lookahead/maxnviolatedcons	how many constraints that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted] Range: {0, ..., ∞}	1
branching/lookahead/maxnviolateddomreds	how many domain reductions that are violated by the base lp solution should be gathered until the rule is stopped and they are added? [0 for unrestricted] Range: {0, ..., ∞}	1
branching/lookahead/maxproprounds	maximum number of propagation rounds to perform at each temporary node (-1: unlimited, 0: SCIP default) Range: {-1, ..., ∞}	0
branching/lookahead/mergedomainreductions	should domain reductions of feasible siblings should be merged? Range: boolean	0
branching/lookahead/minweight	if scoringfunction is 's', this value is used to weight the min of the gains of two child problems in the convex combination Range: [0, ∞]	0.8
branching/lookahead/onlyvioldomreds	should only domain reductions that violate the LP solution be applied? Range: boolean	0
branching/lookahead/prefersimplebounds	should domain reductions only be applied if there are simple bound changes? Range: boolean	0
branching/lookahead/propagate	should domain propagation be executed before each temporary node is solved? Range: boolean	1
branching/lookahead/recursiondepth	the max depth of LAB. Range: {1, ..., ∞}	2
branching/lookahead/reevalage	max number of LPs solved after which a previous prob branching results are recalculated Range: {0, ..., ∞}	10
branching/lookahead/reevalagefsb	max number of LPs solved after which a previous FSB scoring results are recalculated Range: {0, ..., ∞}	10
branching/lookahead/reusebasis	if abbreviated: Should the information gathered to obtain the best candidates be reused? Range: boolean	1
branching/lookahead/scoringfunction	scoring function to be used at the base level Range: d, f, s, w, p, l, c, r, a	a
branching/lookahead/scoringscoringfunction	scoring function to be used during FSB scoring Range: d, f, s, w, l, c, r	d
branching/lookahead/storeunviolatedsol	if only non violating constraints are added, should the branching decision be stored till the next call? Range: boolean	1

Option	Description	Default
branching/lookahead/updatebranchingresults	should branching results (and scores) be updated w.r.t. proven dual bounds? Range: boolean	0
branching/lookahead/usedomainreduction	should domain reductions be collected and applied? Range: boolean	1
branching/lookahead/useimpliedbincons	should binary constraints be collected and applied? Range: boolean	0
branching/lookahead/uselevel2data	should branching data generated at depth level 2 be stored for re-using it? Range: boolean	1
branching/lookahead/worsefactor	if the FSB score is of a candidate is worse than the best by this factor, skip this candidate (-1: disable) Range: [-1, ∞]	-1

5.34.4.11 branching/mostinf

Option	Description	Default
branching/mostinf/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/mostinf/maxdepth	maximal depth level, up to which branching rule <mostinf> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/mostinf/priority	priority of branching rule <mostinf> Range: {-536870912, ..., 536870911}	100

5.34.4.12 branching/multaggr

Option	Description	Default
branching/multaggr/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/multaggr/maxdepth	maximal depth level, up to which branching rule <multaggr> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/multaggr/priority	priority of branching rule <multaggr> Range: {-536870912, ..., 536870911}	0
Options for expert users		
branching/multaggr/maxproprounds	maximum number of propagation rounds to be performed during multaggr branching before solving the LP (-1: no limit, -2: parameter settings) Range: {-2, ..., ∞}	0

Option	Description	Default
branching/multaggr/probingbounds	should valid bounds be identified in a probing-like fashion during multaggr branching (only with propagation)? Range: boolean	1
branching/multaggr/reevalage	number of intermediate LPs solved to trigger reevaluation of strong branching value for a variable that was already evaluated at the current node Range: {0, ..., ∞}	0

5.34.4.13 branching/nodereopt

Option	Description	Default
branching/nodereopt/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/nodereopt/maxdepth	maximal depth level, up to which branching rule <nodereopt> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/nodereopt/priority	priority of branching rule <nodereopt> Range: {-536870912, ..., 536870911}	-9000000

5.34.4.14 branching/pscost

Option	Description	Default
branching/pscost/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/pscost/maxdepth	maximal depth level, up to which branching rule <pscost> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/pscost/narymaxdepth	maximal depth where to do n-ary branching, -1 to turn off Range: {-1, ..., ∞}	-1
branching/pscost/naryminwidth	minimal domain width in children when doing n-ary branching, relative to global bounds Range: [0, 1]	0.001
branching/pscost/narywidthfactor	factor of domain width in n-ary branching when creating nodes with increasing distance from branching value Range: [1, ∞]	2
branching/pscost/nchildren	number of children to create in n-ary branching Range: {2, ..., ∞}	2
branching/pscost/priority	priority of branching rule <pscost> Range: {-536870912, ..., 536870911}	2000

Option	Description	Default
branching/pscost/strategy	strategy for utilizing pseudo-costs of external branching candidates (multiply as in pseudo costs 'u'pdate rule, or by 'd'omain reduction, or by domain reduction of 's'ibling, or by 'v'ariable score) Range: d, s, u, v	u
Options for expert users		
branching/pscost/maxscoreweight	weight for maximum of scores of a branching candidate when building weighted sum of min/max/sum of scores Range: real	1.3
branching/pscost/minscoreweight	weight for minimum of scores of a branching candidate when building weighted sum of min/max/sum of scores Range: real	0.8
branching/pscost/sumscoreweight	weight for sum of scores of a branching candidate when building weighted sum of min/max/sum of scores Range: real	0.1

5.34.4.15 branching/random

Option	Description	Default
branching/random/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/random/maxdepth	maximal depth level, up to which branching rule <random> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/random/priority	priority of branching rule <random> Range: {-536870912, ..., 536870911}	-100000
branching/random/seed	initial random seed value Range: {0, ..., ∞}	41

5.34.4.16 branching/relpscost

Option	Description	Default
branching/relpscost/initcand	maximal number of candidates initialized with strong branching per node Range: {0, ..., ∞}	100
branching/relpscost/inititer	iteration limit for strong branching initializations of pseudo cost entries (0: auto) Range: {0, ..., ∞}	0
branching/relpscost/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1

Option	Description	Default
branching/relpscost/maxdepth	maximal depth level, up to which branching rule <relpscost> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/relpscost/priority	priority of branching rule <relpscost> Range: {-536870912, ..., 536870911}	10000
branching/relpscost/sbiterofs	additional number of allowed strong branching LP iterations Range: {0, ..., ∞}	100000
branching/relpscost/sbiterquot	maximal fraction of strong branching LP iterations compared to node relaxation LP iterations Range: [0, ∞]	0.5
Options for expert users		
branching/relpscost/confidencelevel	the confidence level for statistical methods, between 0 (Min) and 4 (Max). Range: {0, ..., 4}	2
branching/relpscost/conflictlengthweight	weight in score calculations for conflict length score Range: real	0
branching/relpscost/conflictweight	weight in score calculations for conflict score Range: real	0.01
branching/relpscost/cutoffweight	weight in score calculations for cutoff score Range: real	0.0001
branching/relpscost/degeneracyaware	should degeneracy be taken into account to update weights and skip strong branching? (0: off, 1: after root, 2: always) Range: {0, ..., 2}	1
branching/relpscost/dynamicweights	should the weights of the branching rule be adjusted dynamically during solving based on objective and infeasible leaf counters? Range: boolean	1
branching/relpscost/filtercandssym	Use symmetry to filter branching candidates? Range: boolean	0
branching/relpscost/higherrortol	high relative error tolerance for reliability Range: [0, ∞]	1
branching/relpscost/inferenceweight	weight in score calculations for inference score Range: real	0.0001
branching/relpscost/lowerrortol	low relative error tolerance for reliability Range: [0, ∞]	0.05
branching/relpscost/maxbdchgs	maximal number of bound tightenings before the node is reevaluated (-1: unlimited) Range: {-1, ..., ∞}	5
branching/relpscost/maxlookahead	maximal number of further variables evaluated without better score Range: {1, ..., ∞}	9
branching/relpscost/maxproprounds	maximum number of propagation rounds to be performed during strong branching before solving the LP (-1: no limit, -2: parameter settings) Range: {-2, ..., ∞}	-2

Option	Description	Default
branching/relpscost/maxreliable	maximal value for minimum pseudo cost size to regard pseudo cost value as reliable Range: [0, ∞]	5
branching/relpscost/minreliable	minimal value for minimum pseudo cost size to regard pseudo cost value as reliable Range: [0, ∞]	1
branching/relpscost/nlscoreweight	weight in score calculations for nlcount score Range: real	0.1
branching/relpscost/probingbounds	should valid bounds be identified in a probing-like fashion during strong branching (only with propagation)? Range: boolean	1
branching/relpscost/pscostweight	weight in score calculations for pseudo cost score Range: real	1
branching/relpscost/randinitorder	should candidates be initialized in random-ized order? Range: boolean	0
branching/relpscost/skipbadinitcands	should branching rule skip candidates that have a low probability to be better than the best strong-branching or pseudo-candidate? Range: boolean	1
branching/relpscost/startrandseed	start seed for random number generation Range: {0, ..., ∞}	5
branching/relpscost/storesemiinitcosts	should strong branching result be considered for pseudo costs if the other direction was infeasible? Range: boolean	0
branching/relpscost/transsympscost	Transfer pscost information to symmetric variables? Range: boolean	0
branching/relpscost/usedynamicconfidence	should the confidence level be adjusted dy-namically? Range: boolean	0
branching/relpscost/usehyptestforreliability	should the strong branching decision be based on a hypothesis test? Range: boolean	0
branching/relpscost/userelerrorreliability	should reliability be based on relative errors? Range: boolean	0
branching/relpscost/usesblocalinfo	should the scoring function use only local cutoff and inference information obtained for strong branching candidates? Range: boolean	0
branching/relpscost/usesmallweightsitlim	should smaller weights be used for pseudo cost updates after hitting the LP iteration limit? Range: boolean	0

5.34.4.17 branching/treemodel

Option	Description	Default
branching/treemodel/enable	should candidate branching variables be scored using the Treemodel branching rules? Range: boolean	0
branching/treemodel/height	estimated tree height at which we switch from using the low rule to the high rule Range: {0, ..., ∞}	10
branching/treemodel/highrule	scoring function to use at nodes predicted to be high in the tree ('d'efault, 's'vts, 'r'atio, 't'ree sample) Range: d, s, r, t	r
branching/treemodel/lowrule	scoring function to use at nodes predicted to be low in the tree ('d'efault, 's'vts, 'r'atio, 't'ree sample) Range: d, s, r, t	r
Options for expert users		
branching/treemodel/fallbackinf	which method should be used as a fallback if the tree size estimates are infinite? ('d'efault, 'r'atio) Range: d, r	r
branching/treemodel/fallbacknoprim	which method should be used as a fallback if there is no primal bound available? ('d'efault, 'r'atio) Range: d, r	r
branching/treemodel/filterhigh	should dominated candidates be filtered before using the high scoring function? ('a'uto, 't'rue, 'f'alse) Range: a, t, f	a
branching/treemodel/filterlow	should dominated candidates be filtered before using the low scoring function? ('a'uto, 't'rue, 'f'alse) Range: a, t, f	a
branching/treemodel/maxfpiter	maximum number of fixed-point iterations when computing the ratio Range: {1, ..., ∞}	24
branching/treemodel/maxsvtsheight	maximum height to compute the SVTS score exactly before approximating Range: {0, ..., ∞}	100
branching/treemodel/smallpscost	threshold at which pseudocosts are considered small, making hybrid scores more likely to be the deciding factor in branching Range: [0, ∞]	0.1

5.34.4.18 branching/vanillafullstrong

Option	Description	Default
branching/vanillafullstrong/idempotent	should strong branching side-effects be prevented (e.g., domain changes, stat updates etc.)? Range: boolean	0
branching/vanillafullstrong/integralcands	should integral variables in the current LP solution be considered as branching candidates? Range: boolean	0

Option	Description	Default
branching/vanillafullstrong/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying branching rule (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
branching/vanillafullstrong/maxdepth	maximal depth level, up to which branching rule <vanillafullstrong> should be used (-1 for no limit) Range: {-1, ..., 65534}	-1
branching/vanillafullstrong/priority	priority of branching rule <vanillafullstrong> Range: {-536870912, ..., 536870911}	-2000
Options for expert users		
branching/vanillafullstrong/collectscores	should strong branching scores be collected? Range: boolean	0
branching/vanillafullstrong/donotbranch	should candidates only be scored, but no branching be performed? Range: boolean	0
branching/vanillafullstrong/scoreall	should strong branching scores be computed for all candidates, or can we early stop when a variable has infinite score? Range: boolean	0

5.34.4.19 conflict

Option	Description	Default
conflict/conflictgraphweight	the weight the VSIDS score is weight by updating the VSIDS for a variable if it is part of a conflict graph Range: [0, 1]	1
conflict/conflictweight	the weight the VSIDS score is weight by updating the VSIDS for a variable if it is part of a conflict Range: [0, 1]	0
conflict/enable	should conflict analysis be enabled? Range: boolean	1
conflict/preferbinary	should binary conflicts be preferred? Range: boolean	0
conflict/restartfac	factor to increase restartnum with after each restart Range: [0, ∞]	1.5
conflict/restartnum	number of successful conflict analysis calls that trigger a restart (0: disable conflict restarts) Range: {0, ..., ∞}	0
conflict/sepaaltproofs	apply cut generating functions to construct alternative proofs Range: boolean	0
conflict/useboundlp	should bound exceeding LP conflict analysis be used? ('o'ff, 'c'onflict graph, 'd'ual ray, 'b'oth conflict graph and dual ray) Range: o, c, d, b	b

Option	Description	Default
conflict/useinflp	should infeasible LP conflict analysis be used? ('o'ff, 'c'onflict graph, 'd'ual ray, 'b'oth conflict graph and dual ray) Range: o, c, d, b	b
conflict/useprop	should propagation conflict analysis be used? Range: boolean	1
conflict/usepseudo	should pseudo solution conflict analysis be used? Range: boolean	1
conflict/usesb	should infeasible/bound exceeding strong branching conflict analysis be used? Range: boolean	1
Options for expert users		
conflict/allowlocal	should conflict constraints be generated that are only valid locally? Range: boolean	1
conflict/cleanboundexceedings	should conflicts based on an old cutoff bound be removed from the conflict pool after improving the primal bound? Range: boolean	1
conflict/downlockscorefac	score factor for down locks in bound relaxation heuristic Range: real	0
conflict/dynamic	should the conflict constraints be subject to aging? Range: boolean	1
conflict/fuiplevels	number of depth levels up to which first UIP's are used in conflict analysis (-1: use All-FirstUIP rule) Range: {-1, ..., ∞}	-1
conflict/fullshortenconflict	try to shorten the whole conflict set or terminate early (depending on the 'maxvarsdetectimpliedbounds' parameter) Range: boolean	1
conflict/ignorerelaxedbd	should relaxed bounds be ignored? Range: boolean	0
conflict/interconss	maximal number of intermediate conflict constraints generated in conflict graph (-1: use every intermediate constraint) Range: {-1, ..., ∞}	-1
conflict/keepreprop	should constraints be kept for repropagation even if they are too long? Range: boolean	1
conflict/lpiterations	maximal number of LP iterations in each LP resolving loop (-1: no limit) Range: {-1, ..., ∞}	10
conflict/maxconss	maximal number of conflict constraints accepted at an infeasible node (-1: use all generated conflict constraints) Range: {-1, ..., ∞}	10
conflict/maxlploops	maximal number of LP resolving loops during conflict analysis (-1: no limit) Range: {-1, ..., ∞}	2

Option	Description	Default
conflict/maxstoresize	maximal size of conflict store (-1: auto, 0: disable storage) Range: {-1, ..., ∞}	10000
conflict/maxvarsdetectimpliedbounds	maximal number of variables to try to detect global bound implications and shorten the whole conflict set (0: disabled) Range: {0, ..., ∞}	250
conflict/maxvarsfac	maximal fraction of variables involved in a conflict constraint Range: [0, ∞]	0.15
conflict/minimprove	minimal improvement of primal bound to remove conflicts based on a previous incumbent Range: [0, 1]	0.05
conflict/minmaxvars	minimal absolute maximum of variables involved in a conflict constraint Range: {0, ..., ∞}	0
conflict/prefinfproof	prefer infeasibility proof to boundexceeding proof Range: boolean	1
conflict/proofscorefac	score factor for impact on acticity in bound relaxation heuristic Range: real	1
conflict/reconvlevels	number of depth levels up to which UIP reconvergence constraints are generated (-1: generate reconvergence constraints in all depth levels) Range: {-1, ..., ∞}	-1
conflict/removable	should the conflict's relaxations be subject to LP aging and cleanup? Range: boolean	1
conflict/repropagate	should earlier nodes be repropagated in order to replace branching decisions by deductions? Range: boolean	1
conflict/scorefac	factor to decrease importance of variables' earlier conflict scores Range: [1e-06, 1]	0.98
conflict/separate	should the conflict constraints be separated? Range: boolean	1
conflict/settlelocal	should conflict constraints be attached only to the local subtree where they can be useful? Range: boolean	0
conflict/uplockscorefac	score factor for up locks in bound relaxation heuristic Range: real	0
conflict/uselocalrows	use local rows to construct infeasibility proofs Range: boolean	1
conflict/weightrepropdepth	weight of the repropagation depth of a conflict used in score calculation Range: [0, 1]	0.1
conflict/weightsize	weight of the size of a conflict used in score calculation Range: [0, 1]	0.001
conflict/weightvaliddepth	weight of the valid depth of a conflict used in score calculation Range: [0, 1]	1

Option	Description	Default

5.34.4.20 conflict/bounddisjunction

Option	Description	Default
conflict/bounddisjunction/continuousfrac	maximal percantage of continuous variables within a conflict Range: [0, 1]	0.4
Options for expert users		
conflict/bounddisjunction/priority	priority of conflict handler <bounddisjunction> Range: {-2147483648, ..., ∞}	-3000000

5.34.4.21 conflict/graph

Option	Description	Default
Options for expert users		
conflict/graph/depthscorefac	score factor for depth level in bound relaxation heuristic Range: real	1

5.34.4.22 conflict/indicatorconflict

Option	Description	Default
Options for expert users		
conflict/indicatorconflict/priority	priority of conflict handler <indicatorconflict> Range: {-2147483648, ..., ∞}	200000

5.34.4.23 conflict/linear

Option	Description	Default
Options for expert users		
conflict/linear/priority	priority of conflict handler <linear> Range: {-2147483648, ..., ∞}	-1000000

5.34.4.24 conflict/logicor

Option	Description	Default
Options for expert users		
conflict/logicor/priority	priority of conflict handler <logicor> Range: {-2147483648, ..., ∞}	800000

5.34.4.25 conflict/setppc

Option	Description	Default
Options for expert users		
conflict/setppc/priority	priority of conflict handler <setppc> Range: {-2147483648, ..., ∞}	700000

5.34.4.26 constraints

Option	Description	Default
Options for expert users		
constraints/agelimit	maximum age an unnecessary constraint can reach before it is deleted (0: dynamic, -1: keep all constraints) Range: {-1, ..., ∞}	0
constraints/disableenfops	should enforcement of pseudo solution be disabled? Range: boolean	0
constraints/obsoleteage	age of a constraint after which it is marked obsolete (0: dynamic, -1 do not mark constraints obsolete) Range: {-1, ..., ∞}	-1

5.34.4.27 constraints/SOS1

Option	Description	Default
constraints/SOS1/branchnonzeros	Branch on SOS constraint with most number of nonzeros? Range: boolean	0
constraints/SOS1/branchsos	Use SOS1 branching in enforcing (otherwise leave decision to branching rules)? This value can only be set to false if all SOS1 variables are binary Range: boolean	1
constraints/SOS1/branchweight	Branch on SOS cons. with highest nonzero-variable weight for branching (needs branch-nonzeros = false)? Range: boolean	0
constraints/SOS1/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/SOS1/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
Options for expert users		
constraints/SOS1/addbdsfeas	minimal feasibility value for bound inequalities in order to be added to the branching node Range: real	1
constraints/SOS1/addcomps	if TRUE then add complementarity constraints to the branching nodes (can be used in combination with neighborhood or bipartite branching) Range: boolean	0
constraints/SOS1/addcompsdepth	only add complementarity constraints to branching nodes for predefined depth (-1: no limit) Range: {-1, ..., ∞}	30

Option	Description	Default
constraints/SOS1/addcompsfeas	minimal feasibility value for complementarity constraints in order to be added to the branching node Range: real	-0.6
constraints/SOS1/addextendedbds	should added complementarity constraints be extended to SOS1 constraints to get tighter bound inequalities Range: boolean	1
constraints/SOS1/autocutsfromsos1	if TRUE then automatically switch to separating initial SOS1 constraints if the SOS1 constraints do not overlap Range: boolean	1
constraints/SOS1/autosos1branch	if TRUE then automatically switch to SOS1 branching if the SOS1 constraints do not overlap Range: boolean	1
constraints/SOS1/boundcutsdepth	node depth of separating bound cuts (-1: no limit) Range: {-1, ..., ∞}	40
constraints/SOS1/boundcutsfreq	frequency for separating bound cuts; zero means to separate only in the root node Range: {-1, ..., 65534}	10
constraints/SOS1/boundcutsfromgraph	if TRUE separate bound inequalities from the conflict graph Range: boolean	1
constraints/SOS1/boundcutsfromsos1	if TRUE separate bound inequalities from initial SOS1 constraints Range: boolean	0
constraints/SOS1/branchingrule	which branching rule should be applied ? ('n': neighborhood, 'b': bipartite, 's': SOS1/clique) (note: in some cases an automatic switching to SOS1 branching is possible) Range: n, b, s	n
constraints/SOS1/conflictprop	whether to use conflict graph propagation Range: boolean	1
constraints/SOS1/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/SOS1/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/SOS1/depthimplanalysis	number of recursive calls of implication graph analysis (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/SOS1/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/SOS1/fixnonzero	if neighborhood branching is used, then fix the branching variable (if positive in sign) to the value of the feasibility tolerance Range: boolean	0

Option	Description	Default
constraints/SOS1/implcutsdepth	node depth of separating implied bound cuts (-1: no limit) Range: {-1, ..., ∞}	40
constraints/SOS1/implcutsfreq	frequency for separating implied bound cuts; zero means to separate only in the root node Range: {-1, ..., 65534}	0
constraints/SOS1/implprop	whether to use implication graph propagation Range: boolean	1
constraints/SOS1/maxaddcomps	maximal number of complementarity constraints added per branching node (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/SOS1/maxboundcuts	maximal number of bound cuts separated per branching node Range: {0, ..., ∞}	50
constraints/SOS1/maxboundcutsroot	maximal number of bound cuts separated per iteration in the root node Range: {0, ..., ∞}	150
constraints/SOS1/maxextensions	maximal number of extensions that will be computed for each SOS1 constraint (-1: no limit) Range: {-1, ..., ∞}	1
constraints/SOS1/maximplcuts	maximal number of implied bound cuts separated per branching node Range: {0, ..., ∞}	50
constraints/SOS1/maximplcutsroot	maximal number of implied bound cuts separated per iteration in the root node Range: {0, ..., ∞}	150
constraints/SOS1/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/SOS1/maxsosadjacency	do not create an adjacency matrix if number of SOS1 variables is larger than predefined value (-1: no limit) Range: {-1, ..., ∞}	10000
constraints/SOS1/maxtightenbds	maximal number of bound tightening rounds per presolving round (-1: no limit) Range: {-1, ..., ∞}	5
constraints/SOS1/nstrongiter	maximal number LP iterations to perform for each strong branching round (-2: auto, -1: no limit) Range: {-2, ..., ∞}	10000
constraints/SOS1/nstrongrounds	maximal number of strong branching rounds to perform for each node (-1: auto); only available for neighborhood and bipartite branching Range: {-1, ..., ∞}	0
constraints/SOS1/perfimplanalysis	if TRUE then perform implication graph analysis (might add additional SOS1 constraints) Range: boolean	0
constraints/SOS1/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	8

Option	Description	Default
constraints/SOS1/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/SOS1/sosconsprop	whether to use SOS1 constraint propagation Range: boolean	0
constraints/SOS1/strthenboundcuts	if TRUE then bound cuts are strengthened in case bound variables are available Range: boolean	1

5.34.4.28 constraints/SOS2

Option	Description	Default
constraints/SOS2/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/SOS2/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
Options for expert users		
constraints/SOS2/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/SOS2/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/SOS2/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/SOS2/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/SOS2/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	4
constraints/SOS2/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.29 constraints/and

Option	Description	Default
constraints/and/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1

Option	Description	Default
constraints/and/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
Options for expert users		
constraints/and/aggrlinearization	should an aggregated linearization be used? Range: boolean	0
constraints/and/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/and/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/and/dualpresolving	should dual presolving be performed? Range: boolean	1
constraints/and/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/and/enforcecuts	should cuts be separated during LP enforcing? Range: boolean	1
constraints/and/linearize	should the AND-constraint get linearized and removed (in presolving)? Range: boolean	0
constraints/and/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/and/presolpairwise	should pairwise constraint comparison be performed in presolving? Range: boolean	1
constraints/and/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	20
constraints/and/presolusehashing	should hash table be used for detecting redundant constraints in advance Range: boolean	1
constraints/and/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/and/upgraderesultant	should all binary resultant variables be upgraded to implicit binary variables? Range: boolean	1

5.34.4.30 constraints/bounddisjunction

Option	Description	Default
constraints/bounddisjunction/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1

Option	Description	Default
constraints/bounddisjunction/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
Options for expert users		
constraints/bounddisjunction/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/bounddisjunction/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/bounddisjunction/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/bounddisjunction/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/bounddisjunction/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	4
constraints/bounddisjunction/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.31 constraints/components

Option	Description	Default
constraints/components/intfactor	the weight of an integer variable compared to binary variables Range: [0, ∞]	1
constraints/components/maxdepth	maximum depth of a node to run components detection (-1: disable component detection during solving) Range: {-1, ..., ∞}	-1
constraints/components/nodelimit	maximum number of nodes to be solved in subproblems during presolving Range: {-1, ..., ∞}	10000
constraints/components/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/components/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
Options for expert users		
constraints/components/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	1

Option	Description	Default
constraints/components/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/components/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	-1
constraints/components/feastolfactor	factor to increase the feasibility tolerance of the main SCIP in all sub-SCIPs, default value 1.0 Range: [0, 1e+06]	1
constraints/components/maxintvars	maximum number of integer (or binary) variables to solve a subproblem during presolving (-1: unlimited) Range: {-1, ..., ∞}	500
constraints/components/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/components/minrelsize	minimum relative size (in terms of variables) to solve a component individually during branch-and-bound Range: [0, 1]	0.1
constraints/components/minsize	minimum absolute size (in terms of variables) to solve a component individually during branch-and-bound Range: {0, ..., ∞}	50
constraints/components/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	32
constraints/components/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.32 constraints/indicator

Option	Description	Default
constraints/indicator/maxsepacuts	maximal number of cuts separated per separation round Range: {0, ..., ∞}	100
constraints/indicator/maxsepacutsroot	maximal number of cuts separated per separation round in the root node Range: {0, ..., ∞}	2000
constraints/indicator/maxsepanonviolated	maximal number of separated non violated IISs, before separation is stopped Range: {0, ..., ∞}	3
constraints/indicator/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/indicator/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10

Option	Description	Default
Options for expert users		
constraints/indicator/addcoupling	Add coupling constraints or rows if big-M is small enough? Range: boolean	1
constraints/indicator/addcouplingcons	Add initial variable upper bound constraints, if 'addcoupling' is true? Range: boolean	0
constraints/indicator/addopposite	Add opposite inequality in nodes in which the binary variable has been fixed to 0? Range: boolean	0
constraints/indicator/branchindicators	Branch on indicator constraints in enforcing? Range: boolean	0
constraints/indicator/conflictsupgrade	Try to upgrade bounddisjunction conflicts by replacing slack variables? Range: boolean	0
constraints/indicator/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/indicator/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/indicator/dualreductions	Should dual reduction steps be performed? Range: boolean	1
constraints/indicator/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/indicator/enforcecuts	In enforcing try to generate cuts (only if sepaalternativelp is true)? Range: boolean	0
constraints/indicator/forcerestart	Force restart if absolute gap is 1 or enough binary variables have been fixed? Range: boolean	0
constraints/indicator/generatebilinear	Do not generate indicator constraint, but a bilinear constraint instead? Range: boolean	0
constraints/indicator/genlogicor	Generate logicor constraints instead of cuts? Range: boolean	0
constraints/indicator/maxconditionaltlp	maximum estimated condition of the solution basis matrix of the alternative LP to be trustworthy (0.0 to disable check) Range: [0, ∞]	0
constraints/indicator/maxcouplingvalue	maximum coefficient for binary variable in coupling constraint Range: [0, 1e+09]	10000
constraints/indicator/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1

Option	Description	Default
constraints/indicator/nolinconscont	Decompose problem (do not generate linear constraint if all variables are continuous)? Range: boolean	0
constraints/indicator/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	4
constraints/indicator/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/indicator/removeindicators	Remove indicator constraint if corresponding variable bound constraint has been added? Range: boolean	0
constraints/indicator/restartfrac	fraction of binary variables that need to be fixed before restart occurs (in forcerestart) Range: [0, 1]	0.9
constraints/indicator/scaleslackvar	Scale slack variable coefficient at construction time? Range: boolean	0
constraints/indicator/sepaalternativelp	Separate using the alternative LP? Range: boolean	0
constraints/indicator/sepacouplingcuts	Should the coupling inequalities be separated dynamically? Range: boolean	1
constraints/indicator/sepacouplinglocal	Allow to use local bounds in order to separate coupling inequalities? Range: boolean	0
constraints/indicator/sepacouplingvalue	maximum coefficient for binary variable in separated coupling constraint Range: [0, 1e+09]	10000
constraints/indicator/sepaperspective	Separate cuts based on perspective formulation? Range: boolean	0
constraints/indicator/sepapersplocal	Allow to use local bounds in order to separate perspective cuts? Range: boolean	1
constraints/indicator/trysolfromcover	Try to construct a feasible solution from a cover? Range: boolean	0
constraints/indicator/trysolutions	Try to make solutions feasible by setting indicator variables? Range: boolean	1
constraints/indicator/updatebounds	Update bounds of original variables for separation? Range: boolean	0
constraints/indicator/upgradelinear	Try to upgrade linear constraints to indicator constraints? Range: boolean	0

Option	Description	Default
constraints/indicator/useobjectivecut	Use objective cut with current best solution to alternative LP? Range: boolean	0
constraints/indicator/useotherconss	Collect other constraints to alternative LP? Range: boolean	0

5.34.4.33 constraints/integral

Option	Description	Default
constraints/integral/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
constraints/integral/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
Options for expert users		
constraints/integral/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/integral/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/integral/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	-1
constraints/integral/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	0
constraints/integral/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	28
constraints/integral/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.34 constraints/knapsack

Option	Description	Default
constraints/knapsack/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	5
constraints/knapsack/maxroundsroot	maximal number of separation rounds per node in the root node (-1: unlimited) Range: {-1, ..., ∞}	-1
constraints/knapsack/maxsepacuts	maximal number of cuts separated per separation round Range: {0, ..., ∞}	50

Option	Description	Default
constraints/knapsack/maxsepacutsroot	maximal number of cuts separated per separation round in the root node Range: {0, ..., ∞}	200
constraints/knapsack/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/knapsack/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
Options for expert users		
constraints/knapsack/cliqueextractfactor	lower clique size limit for greedy clique extraction algorithm (relative to largest clique) Range: [0, 1]	0.5
constraints/knapsack/clqpartupdatefac	factor on the growth of global cliques to decide when to update a previous (negated) clique partition (used only if updatecliquepartitions is set to TRUE) Range: [1, 10]	1.5
constraints/knapsack/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/knapsack/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/knapsack/detectcutoffbound	should presolving try to detect constraints parallel to the objective function defining an upper bound and prevent these constraints from entering the LP? Range: boolean	1
constraints/knapsack/detectlowerbound	should presolving try to detect constraints parallel to the objective function defining a lower bound and prevent these constraints from entering the LP? Range: boolean	1
constraints/knapsack/disaggregation	should disaggregation of knapsack constraints be allowed in preprocessing? Range: boolean	1
constraints/knapsack/dualpresolving	should dual presolving steps be performed? Range: boolean	1
constraints/knapsack/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/knapsack/maxcardbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for separating knapsack cuts Range: [0, 1]	0
constraints/knapsack/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1

Option	Description	Default
constraints/knapsack/negatedclique	should negated clique information be used in solving process Range: boolean	1
constraints/knapsack/presolpairwise	should pairwise constraint comparison be performed in presolving? Range: boolean	1
constraints/knapsack/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	28
constraints/knapsack/presolusehashing	should hash table be used for detecting redundant constraints in advance Range: boolean	1
constraints/knapsack/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/knapsack/sepacardfreq	multiplier on separation frequency, how often knapsack cuts are separated (-1: never, 0: only at root) Range: {-1, ..., 65534}	1
constraints/knapsack/simplifyinequalities	should presolving try to simplify knapsacks Range: boolean	1
constraints/knapsack/updatecliquepartitions	should clique partition information be updated when old partition seems outdated? Range: boolean	0
constraints/knapsack/usegubs	should GUB information be used for separation? Range: boolean	0

5.34.4.35 constraints/linear

Option	Description	Default
constraints/linear/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	5
constraints/linear/maxroundsroot	maximal number of separation rounds per node in the root node (-1: unlimited) Range: {-1, ..., ∞}	-1
constraints/linear/maxsepacuts	maximal number of cuts separated per separation round Range: {0, ..., ∞}	50
constraints/linear/maxsepacutsroot	maximal number of cuts separated per separation round in the root node Range: {0, ..., ∞}	200
constraints/linear/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/linear/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0

Option	Description	Default
constraints/linear/separateall	should all constraints be subject to cardinality cut generation instead of only the ones with non-zero dual value? Range: boolean	0
Options for expert users		
constraints/linear/aggregatevariables	should presolving search for aggregations in equations Range: boolean	1
constraints/linear/checkrelmaxabs	should the violation for a constraint with side 0.0 be checked relative to 1.0 (FALSE) or to the maximum absolute value in the activity (TRUE)? Range: boolean	0
constraints/linear/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/linear/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/linear/detectcutoffbound	should presolving try to detect constraints parallel to the objective function defining an upper bound and prevent these constraints from entering the LP? Range: boolean	1
constraints/linear/detectlowerbound	should presolving try to detect constraints parallel to the objective function defining a lower bound and prevent these constraints from entering the LP? Range: boolean	1
constraints/linear/detectpartialobjective	should presolving try to detect subsets of constraints parallel to the objective function? Range: boolean	1
constraints/linear/dualpresolving	should dual presolving steps be performed? Range: boolean	1
constraints/linear/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/linear/extractcliques	should Cliques be extracted? Range: boolean	1
constraints/linear/maxaggrnormscale	maximal allowed relative gain in maximum norm for constraint aggregation (0.0: disable constraint aggregation) Range: [0, ∞]	0
constraints/linear/maxcardbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for separating knapsack cardinality cuts Range: [0, 1]	0
constraints/linear/maxdualmultaggrquot	maximum coefficient dynamism (ie. maxabsval / minabsval) for dual multiaggregation Range: [1, ∞]	∞

Option	Description	Default
constraints/linear/maxeasyactivitydelta	maximum activity delta to run easy propagation on linear constraint (faster, but numerically less stable) Range: [0, ∞]	1e+06
constraints/linear/maxmultaggrquot	maximum coefficient dynamism (ie. maxabsval / minabsval) for primal multiaggregation Range: [1, ∞]	1000
constraints/linear/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/linear/mingainpernmincomparisons	minimal gain per minimal pairwise presolve comparisons to repeat pairwise comparison round Range: [0, 1]	1e-06
constraints/linear/multaggrremove	should multi-aggregations only be performed if the constraint can be removed afterwards? Range: boolean	0
constraints/linear/nmincomparisons	number for minimal pairwise presolve comparisons Range: {1, ..., ∞}	200000
constraints/linear/presolpairwise	should pairwise constraint comparison be performed in presolving? Range: boolean	1
constraints/linear/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	20
constraints/linear/presolusehashing	should hash table be used for detecting redundant constraints in advance Range: boolean	1
constraints/linear/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/linear/rangedrowartcons	should presolving and propagation extract sub-constraints from ranged rows and equations? Range: boolean	1
constraints/linear/rangedrowfreq	frequency for applying ranged row propagation Range: {1, ..., 65534}	1
constraints/linear/rangedrowmaxdepth	maximum depth to apply ranged row propagation Range: {0, ..., ∞}	∞
constraints/linear/rangedrowpropagation	should presolving and propagation try to improve bounds, detect infeasibility, and extract sub-constraints from ranged rows and equations? Range: boolean	1
constraints/linear/simplifyinequalities	should presolving try to simplify inequalities Range: boolean	1

Option	Description	Default
constraints/linear/singletonstuffing	should stuffing of singleton continuous variables be performed? Range: boolean	1
constraints/linear/singlevarstuffing	should single variable stuffing be performed, which tries to fulfill constraints using the cheapest variable? Range: boolean	0
constraints/linear/sortvars	apply binaries sorting in decr. order of coeff abs value? Range: boolean	1
constraints/linear/tightenboundsfreq	multiplier on propagation frequency, how often the bounds are tightened (-1: never, 0: only at root) Range: {-1, ..., 65534}	1

5.34.4.36 constraints/linear/upgrade

Option	Description	Default
constraints/linear/upgrade/indicator	enable linear upgrading for constraint handler <indicator> Range: boolean	1
constraints/linear/upgrade/knapsack	enable linear upgrading for constraint handler <knapsack> Range: boolean	1
constraints/linear/upgrade/logicor	enable linear upgrading for constraint handler <logicor> Range: boolean	1
constraints/linear/upgrade/setppc	enable linear upgrading for constraint handler <setppc> Range: boolean	1
constraints/linear/upgrade/varbound	enable linear upgrading for constraint handler <varbound> Range: boolean	1
constraints/linear/upgrade/xor	enable linear upgrading for constraint handler <xor> Range: boolean	1

5.34.4.37 constraints/logicor

Option	Description	Default
constraints/logicor/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/logicor/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
Options for expert users		
constraints/logicor/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0

Option	Description	Default
constraints/logicor/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/logicor/dualpresolving	should dual presolving steps be performed? Range: boolean	1
constraints/logicor/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/logicor/implications	should implications/cliques be used in presolving Range: boolean	1
constraints/logicor/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/logicor/negatedclique	should negated clique information be used in presolving Range: boolean	1
constraints/logicor/presolpairwise	should pairwise constraint comparison be performed in presolving? Range: boolean	1
constraints/logicor/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	28
constraints/logicor/presolusehashing	should hash table be used for detecting redundant constraints in advance Range: boolean	1
constraints/logicor/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/logicor/strengthen	should pairwise constraint comparison try to strengthen constraints by removing superflous nonzeros? Range: boolean	1

5.34.4.38 constraints/nonlinear

Option	Description	Default
constraints/nonlinear/assumeconvex	whether to assume that any constraint is convex Range: boolean	0
constraints/nonlinear/bilinmaxnauxexprs	maximal number of auxiliary expressions per bilinear term Range: {0, ..., ∞}	10
constraints/nonlinear/linearizeheursol	whether tight linearizations of nonlinear constraints should be added to cutpool when some heuristics finds a new solution ('o'ff, on new 'i'ncumbents, on 'e'very solution) Range: o, i, e	i if QCP or NLP, o otherwise

Option	Description	Default
constraints/nonlinear/maxproprounds	limit on number of propagation rounds for a set of constraints within one round of SCIP propagation Range: {0, ..., ∞}	10
constraints/nonlinear/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/nonlinear/reformbinprods	whether to reformulate products of binary variables during presolving Range: boolean	1
constraints/nonlinear/reformbinprodsand	whether to use the AND constraint handler for reformulating binary products Range: boolean	1
constraints/nonlinear/reformbinprodsfac	minimum number of terms to reformulate bilinear binary products by factorizing variables (≤ 1: disabled) Range: {1, ..., ∞}	50
constraints/nonlinear/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
Options for expert users		
constraints/nonlinear/checkvarlocks	whether variables contained in a single constraint should be forced to be at their lower or upper bounds ('d'isable, change 't'ype, add 'b'ound disjunction) Range: b, d, t	t
constraints/nonlinear/conssiderelaxamount	by how much to relax constraint sides during bound tightening Range: [0, 1]	1e-09
constraints/nonlinear/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/nonlinear/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/nonlinear/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/nonlinear/enfoauxviolfactor	an expression will be enforced if the "auxiliary" violation is at least this factor times the "original" violation Range: [0, 1]	0.01

Option	Description	Default
constraints/nonlinear/forbidmultaggrnlvar	whether to forbid multiaggregation of nonlinear variables Range: boolean	1
constraints/nonlinear/forcestrongcut	whether to force "strong" cuts in enforcement Range: boolean	0
constraints/nonlinear/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/nonlinear/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	28
constraints/nonlinear/propauxvars	whether to check bounds of all auxiliary variable to seed reverse propagation Range: boolean	1
constraints/nonlinear/propinenforce	whether to (re)run propagation in enforcement Range: boolean	0
constraints/nonlinear/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/nonlinear/rownotremovable	whether to make rows to be non-removable in the node where they are added (can prevent some cycling): 'o'ff, in 'e'nforcement only, 'a'lways Range: o, e, a	o
constraints/nonlinear/strongcuteffica	consider efficacy requirement when deciding whether a cut is "strong" Range: boolean	0
constraints/nonlinear/strongcutmaxcoef	"strong" cuts will be scaled to have their maximal coef in [1/strongcutmaxcoef,strongcutmaxcoef] Range: [1, ∞]	1000
constraints/nonlinear/tightenlpfeastol	whether to tighten LP feasibility tolerance during enforcement, if it seems useful Range: boolean	1
constraints/nonlinear/varboundrelax	strategy on how to relax variable bounds during bound tightening: relax (n)ot, relax by (a)bsolute value, relax always by a(b)solute value, relax by (r)relative value Range: n, a, b, r	r

Option	Description	Default
constraints/nonlinear/varboundrelaxamount	by how much to relax variable bounds during bound tightening if strategy 'a', 'b', or 'r' Range: [0, 1]	1e-09
constraints/nonlinear/violscale	method how to scale violations to make them comparable (not used for feasibility check): (n)one, (a)ctivity and side, norm of (g)radient Range: n, a, g	n
constraints/nonlinear/vpadjfacetthresh	adjust computed facet of envelope of vertex-polyhedral function up to a violation of this value times LP feasibility tolerance Range: [0, ∞]	10
constraints/nonlinear/vpdualsimplex	whether to use dual simplex instead of primal simplex for LP that computes facet of vertex-polyhedral function Range: boolean	1
constraints/nonlinear/vpmaxperturb	maximal relative perturbation of reference point when computing facet of envelope of vertex-polyhedral function (dim>2) Range: [0, 1]	0.001
constraints/nonlinear/weakcutminviolfactor	retry enfo of constraint with weak cuts if violation is least this factor of maximal violated constraints Range: [0, 2]	0.5
constraints/nonlinear/weakcutthreshold	threshold for when to regard a cut from an estimator as weak (lower values allow more weak cuts) Range: [0, 1]	0.2

5.34.4.39 constraints/nonlinear/branching

Option	Description	Default
constraints/nonlinear/branching/aux	from which depth on in the tree to allow branching on auxiliary variables (variables added for extended formulation) Range: {0, ..., ∞}	∞
constraints/nonlinear/branching/domainweight	weight by how much to consider the domain width in branching score Range: [0, ∞]	0
constraints/nonlinear/branching/dualweight	weight by how much to consider the dual values of rows that contain a variable for its branching score Range: [0, ∞]	0
constraints/nonlinear/branching/external	whether to use external branching candidates and branching rules for branching Range: boolean	0

Option	Description	Default
constraints/nonlinear/branching/highscorefactor	consider a variable branching score high if its branching score ≥ this factor ∗ maximal branching score among all variables Range: [0, 1]	0.9
constraints/nonlinear/branching/highviolfactor	consider a constraint highly violated if its violation is ≥ this factor ∗ maximal violation among all constraints Range: [0, 1]	0
constraints/nonlinear/branching/pscostreliable	minimum pseudo-cost update count required to consider pseudo-costs reliable Range: [0, ∞]	2
constraints/nonlinear/branching/pscostweight	weight by how much to consider the pseudo cost of a variable for its branching score Range: [0, ∞]	1
constraints/nonlinear/branching/scoreagg	how to aggregate several branching scores given for the same expression: 'a'verage, 'm'aximum, 's'um Range: a, m, s	s
constraints/nonlinear/branching/vartypeweight	weight by how much to consider variable type (continuous: 0, binary: 1, integer: 0.1, impl-integer: 0.01) in branching score Range: [0, ∞]	0.5
constraints/nonlinear/branching/violsplit	method used to split violation in expression onto variables: 'u'niform, 'm'idness of solution, 'd'omain width, 'l'ogarithmic domain width Range: u, m, d, l	m
constraints/nonlinear/branching/violweight	weight by how much to consider the violation assigned to a variable for its branching score Range: [0, ∞]	1

5.34.4.40 constraints/nonlinear/upgrade

Option	Description	Default
constraints/nonlinear/upgrade/linear	enable nonlinear upgrading for constraint handler <linear> Range: boolean	1
constraints/nonlinear/upgrade/setppc	enable nonlinear upgrading for constraint handler <setppc> Range: boolean	1

5.34.4.41 constraints/or

Option	Description	Default
constraints/or/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/or/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0

Option	Description	Default
Options for expert users		
constraints/or/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/or/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/or/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/or/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/or/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	8
constraints/or/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.42 constraints/orbisack

Option	Description	Default
constraints/orbisack/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	5
constraints/orbisack/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	5
Options for expert users		
constraints/orbisack/checkpporbisack	Upgrade orbisack constraints to packing/partioning orbisacks? Range: boolean	1
constraints/orbisack/coeffbound	Maximum size of coefficients for orbisack inequalities Range: [0, ∞]	1e+06
constraints/orbisack/coverseparation	Separate cover inequalities for orbisacks? Range: boolean	1
constraints/orbisack/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/orbisack/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/orbisack/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	-1

Option	Description	Default
constraints/orbisack/forceconscopy	Whether orbisack constraints should be forced to be copied to sub SCIPs. Range: boolean	0
constraints/orbisack/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/orbisack/orbiSeparation	Separate orbisack inequalities? Range: boolean	0
constraints/orbisack/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16
constraints/orbisack/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.43 constraints/orbitope

Option	Description	Default
constraints/orbitope/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/orbitope/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
Options for expert users		
constraints/orbitope/checkpporbitope	Strengthen orbitope constraints to packing/partioning orbitopes? Range: boolean	1
constraints/orbitope/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/orbitope/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/orbitope/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	-1
constraints/orbitope/forceconscopy	Whether orbitope constraints should be forced to be copied to sub SCIPs. Range: boolean	0
constraints/orbitope/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/orbitope/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	8

Option	Description	Default
constraints/orbitope/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/orbitope/sepafullorbitope	Whether we separate inequalities for full orbitopes? Range: boolean	0

5.34.4.44 constraints/setppc

Option	Description	Default
constraints/setppc/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/setppc/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
Options for expert users		
constraints/setppc/addvariablesascliques	should we try to generate extra cliques out of all binary variables to maybe fasten redundant constraint detection Range: boolean	0
constraints/setppc/cliquelifting	should we try to lift variables into other clique constraints, fix variables, aggregate them, and also shrink the amount of variables in clique constraints Range: boolean	0
constraints/setppc/clqueshrinking	should we try to shrink the number of variables in a clique constraints, by replacing more than one variable by only one Range: boolean	1
constraints/setppc/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/setppc/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/setppc/dualpresolving	should dual presolving steps be performed? Range: boolean	1
constraints/setppc/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/setppc/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/setppc/npseudobranches	number of children created in pseudo branching (0: disable pseudo branching) Range: {0, ..., ∞}	2

Option	Description	Default
constraints/setppc/presolpairwise	should pairwise constraint comparison be performed in presolving? Range: boolean	1
constraints/setppc/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	28
constraints/setppc/presolusehashing	should hash table be used for detecting redundant constraints in advance Range: boolean	1
constraints/setppc/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.45 constraints/symresack

Option	Description	Default
constraints/symresack/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	5
constraints/symresack/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	5
Options for expert users		
constraints/symresack/checkmonotonicity	Check whether permutation is monotone when upgrading to packing/partioning symresacks? Range: boolean	1
constraints/symresack/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/symresack/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/symresack/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	-1
constraints/symresack/forceconscopy	Whether symresack constraints should be forced to be copied to sub SCIPs. Range: boolean	0
constraints/symresack/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/symresack/ppsymresack	Upgrade symresack constraints to packing/partioning symresacks? Range: boolean	1

Option	Description	Default
constraints/symresack/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16
constraints/symresack/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.46 constraints/varbound

Option	Description	Default
constraints/varbound/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/varbound/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
constraints/varbound/usebdwidening	should bound widening be used in conflict analysis? Range: boolean	1
Options for expert users		
constraints/varbound/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/varbound/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/varbound/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/varbound/maxlpcoef	maximum coefficient in varbound constraint to be added as a row into LP Range: [0, ∞]	1e+09
constraints/varbound/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/varbound/presolpairwise	should pairwise constraint comparison be performed in presolving? Range: boolean	1
constraints/varbound/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	12
constraints/varbound/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1

5.34.4.47 constraints/xor

Option	Description	Default
constraints/xor/propfreq	frequency for propagating domains (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
constraints/xor/sepafreq	frequency for separating cuts (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
Options for expert users		
constraints/xor/addextendedform	should the extended formulation be added in presolving? Range: boolean	0
constraints/xor/addflowextended	should the extended flow formulation be added (non-symmetric formulation otherwise)? Range: boolean	0
constraints/xor/delayprop	should propagation method be delayed, if other propagators found reductions? Range: boolean	0
constraints/xor/delaysepa	should separation method be delayed, if other separators found cuts? Range: boolean	0
constraints/xor/eagerfreq	frequency for using all instead of only the useful constraints in separation, propagation and enforcement (-1: never, 0: only in first evaluation) Range: {-1, ..., 65534}	100
constraints/xor/gausspropfreq	frequency for applying the Gauss propagator Range: {-1, ..., 65534}	5
constraints/xor/maxprerounds	maximal number of presolving rounds the constraint handler participates in (-1: no limit) Range: {-1, ..., ∞}	-1
constraints/xor/presolpairwise	should pairwise constraint comparison be performed in presolving? Range: boolean	1
constraints/xor/presoltiming	timing mask of the constraint handler's presolving method (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	28
constraints/xor/presolusehashing	should hash table be used for detecting redundant constraints in advance? Range: boolean	1
constraints/xor/proptiming	timing when constraint propagation should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS) Range: {1, ..., 15}	1
constraints/xor/separateparity	should parity inequalities be separated? Range: boolean	0

5.34.4.48 cutselection/hybrid

Option	Description	Default
cutselection/hybrid/dircutoffdistweight	weight of directed cutoff distance in cut score calculation Range: [0, ∞]	0
cutselection/hybrid/efficacyweight	weight of efficacy in cut score calculation Range: [0, ∞]	1
cutselection/hybrid/intsupportweight	weight of integral support in cut score calculation Range: [0, ∞]	0.1
cutselection/hybrid/minortho	minimal orthogonality for a cut to enter the LP Range: [0, 1]	0.9
cutselection/hybrid/minorthoroot	minimal orthogonality for a cut to enter the LP in the root node Range: [0, 1]	0.9
cutselection/hybrid/objparalweight	weight of objective parallelism in cut score calculation Range: [0, ∞]	0.1
cutselection/hybrid/priority	priority of cut selection rule <hybrid> Range: {-536870912, ..., 1073741823}	8000

5.34.4.49 decomposition

Option	Description	Default
decomposition/disablemeasures	disable expensive measures Range: boolean	0
decomposition/maxgraphedge	maximum number of edges in block graph computation (-1: no limit, 0: disable block graph computation) Range: {-1, ..., ∞}	10000

5.34.4.50 display

Option	Description	Default
display/allviols	display all violations for a given start solution / the best solution after the solving process? Range: boolean	0
display/freq	frequency for displaying node information lines Range: {-1, ..., ∞}	100
display/headerfreq	frequency for displaying header lines (every n'th node information line) Range: {-1, ..., ∞}	15
display/lpinfo	should the LP solver display status messages? Range: boolean	0
display/relevantstats	should the relevant statistics be displayed at the end of solving? Range: boolean	1
display/statistics	whether to print statistics on a solve and a provided decomposition Range: boolean	0

Option	Description	Default
display/verblevel	verbosity level of output Range: {0, ..., 5}	4
display/width	maximal number of characters in a node information line Range: {0, ..., ∞}	143 (80 for Windows without IDE)

5.34.4.51 display/avgdualbound

Option	Description	Default
display/avgdualbound/active	display activation status of display column <avgdualbound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.52 display/completed

Option	Description	Default
display/completed/active	display activation status of display column <completed> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.53 display/concdualbound

Option	Description	Default
display/concdualbound/active	display activation status of display column <concdualbound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.54 display/concgap

Option	Description	Default
display/concgap/active	display activation status of display column <concgap> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.55 display/concmemused

Option	Description	Default
display/concmemused/active	display activation status of display column <concmemused> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.56 display/concprimalbound

Option	Description	Default
display/concprimalbound/active	display activation status of display column <concprimalbound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.57 display/concsolfound

Option	Description	Default
display/concsolfound/active	display activation status of display column <concsolfound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.58 display/conflicts

Option	Description	Default
display/conflicts/active	display activation status of display column <conflicts> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.59 display/conss

Option	Description	Default
display/conss/active	display activation status of display column <conss> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.60 display/curcols

Option	Description	Default
display/curcols/active	display activation status of display column <curcols> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.61 display/curconss

Option	Description	Default
display/curconss/active	display activation status of display column <curconss> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.62 display/curdualbound

Option	Description	Default
display/curdualbound/active	display activation status of display column <curdualbound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.63 display/currows

Option	Description	Default
display/currows/active	display activation status of display column <currows> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.64 display/cutoffbound

Option	Description	Default
display/cutoffbound/active	display activation status of display column <cutoffbound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.65 display/cuts

Option	Description	Default
display/cuts/active	display activation status of display column <cuts> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.66 display/depth

Option	Description	Default
display/depth/active	display activation status of display column <depth> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.67 display/dualbound

Option	Description	Default
display/dualbound/active	display activation status of display column <dualbound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.68 display/estimate

Option	Description	Default
display/estimate/active	display activation status of display column <estimate> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.69 display/feasST

Option	Description	Default
display/feasST/active	display activation status of display column <feasST> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	0

5.34.4.70 display/gap

Option	Description	Default
display/gap/active	display activation status of display column <gap> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.71 display/lpavgiterations

Option	Description	Default
display/lpavgiterations/active	display activation status of display column <lpavgiterations> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1 (0 for Windows without IDE)

5.34.4.72 display/lpcond

Option	Description	Default
display/lpcond/active	display activation status of display column <lpcond> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.73 display/lpiterations

Option	Description	Default
display/lpiterations/active	display activation status of display column <lpiterations> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.74 display/lpobj

Option	Description	Default
display/lpobj/active	display activation status of display column <lpobj> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.75 display/maxdepth

Option	Description	Default
display/maxdepth/active	display activation status of display column <maxdepth> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1 (0 for Windows without IDE)

5.34.4.76 display/memtotal

Option	Description	Default
display/memtotal/active	display activation status of display column <memtotal> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.77 display/memused

Option	Description	Default
display/memused/active	display activation status of display column <memused> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.78 display/nexternbranchcands

Option	Description	Default
display/nexternbranchcands/active	display activation status of display column <nexternbranchcands> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.79 display/nfrac

Option	Description	Default
display/nfrac/active	display activation status of display column <nfrac> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.80 display/ninfeasleaves

Option	Description	Default
display/ninfeasleaves/active	display activation status of display column <ninfeasleaves> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.81 display/nnodes

Option	Description	Default
display/nnodes/active	display activation status of display column <nnodes> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.82 display/nnodesbelowinc

Option	Description	Default
display/nnodesbelowinc/active	display activation status of display column <nnodesbelowinc> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	0

5.34.4.83 display/nobjleaves

Option	Description	Default
display/nobjleaves/active	display activation status of display column <nobjleaves> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.84 display/nodesleft

Option	Description	Default
display/nodesleft/active	display activation status of display column <nodesleft> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.85 display/nrank1nodes

Option	Description	Default
display/nrank1nodes/active	display activation status of display column <nrank1nodes> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	0

5.34.4.86 display/nsols

Option	Description	Default
display/nsols/active	display activation status of display column <nsols> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.87 display/plungedepth

Option	Description	Default
display/plungedepth/active	display activation status of display column <plungedepth> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.88 display/poolsize

Option	Description	Default
display/poolsize/active	display activation status of display column <poolsize> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.89 display/primalbound

Option	Description	Default
display/primalbound/active	display activation status of display column <primalbound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.90 display/primalgap

Option	Description	Default
display/primalgap/active	display activation status of display column <primalgap> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	0

5.34.4.91 display/pseudoobj

Option	Description	Default
display/pseudoobj/active	display activation status of display column <pseudoobj> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.92 display/separounds

Option	Description	Default
display/separounds/active	display activation status of display column <separounds> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.93 display/solfound

Option	Description	Default
display/solfound/active	display activation status of display column <solfound> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.94 display/sols

Option	Description	Default
display/sols/active	display activation status of display column <sols> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	0

5.34.4.95 display/strongbranchs

Option	Description	Default
display/strongbranchs/active	display activation status of display column <strongbranchs> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.96 display/time

Option	Description	Default
display/time/active	display activation status of display column <time> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1 (2 for Windows without IDE)

5.34.4.97 display/vars

Option	Description	Default
display/vars/active	display activation status of display column <vars> (0: off, 1: auto, 2:on) Range: {0, ..., 2}	1

5.34.4.98 estimation

Option	Description	Default
estimation/coefmonossg	coefficient of 1 - SSG in monotone approximation of search completion Range: [0, 1]	0.6333
estimation/coefmonoweight	coefficient of tree weight in monotone approximation of search completion Range: [0, 1]	0.3667
estimation/completiontype	approximation of search tree completion: (a)uto, (g)ap, tree (w)eight, (m)onotone regression, (r)egression forest, (s)sg Range: a, g, m, r, s, w	a
estimation/method	tree size estimation method: (c)ompletion, (e)nsemble, time series forecasts on either (g)ap, (l)eaf frequency, (o)pen nodes, tree (w)eight, (s)sg, or (t)ree profile or w(b)e Range: b, c, e, g, l, o, s, t, w	w
estimation/regforestfilename	user regression forest in RFCSV format Range: string	-
Options for expert users		
estimation/reportfreq	report frequency on estimation: -1: never, 0:always, k $\geq$ 1: k times evenly during search Range: {-1, ..., 1073741823}	-1
estimation/showstats	should statistics be shown at the end? Range: boolean	0
estimation/useleafts	use leaf nodes as basic observations for time series, or all nodes? Range: boolean	1

5.34.4.99 estimation/restarts

Option	Description	Default
estimation/restarts/countonlyleaves	should only leaves count for the minnodes parameter? Range: boolean	0
estimation/restarts/hitcounterlim	limit on the number of successive samples to really trigger a restart Range: {1, ..., ∞}	50
estimation/restarts/minnodes	minimum number of nodes before restart Range: {-1, ..., ∞}	1000
estimation/restarts/restartactpricers	whether to apply a restart when active pricers are used Range: boolean	0
estimation/restarts/restartfactor	factor by which the estimated number of nodes should exceed the current number of nodes Range: [1, ∞]	50
estimation/restarts/restartlimit	restart limit Range: {-1, ..., ∞}	1
estimation/restarts/restartnonlinear	whether to apply a restart when nonlinear constraints are present Range: boolean	0
estimation/restarts/restartpolicy	restart policy: (a)lways, (c)ompletion, (e)stimation, (n)ever Range: a, c, e, n	e

5.34.4.100 estimation/ssg

Option	Description	Default
estimation/ssg/nmaxsubtrees	the maximum number of individual SSG subtrees; -1: no limit Range: {-1, ..., 1073741823}	-1
estimation/ssg/nminnodeslastsplit	minimum number of nodes to process between two consecutive SSG splits Range: {0, ..., ∞}	0

5.34.4.101 estimation/treeprofile

Option	Description	Default
estimation/treeprofile/enabled	should the event handler collect data? Range: boolean	0
estimation/treeprofile/minnodesperdepth	minimum average number of nodes at each depth before producing estimations Range: [1, ∞]	20

5.34.4.102 expr/log

Option	Description	Default
expr/log/minzerodistance	minimal distance from zero to enforce for child in bound tightening Range: [0, 1]	1e-09

5.34.4.103 expr/pow

Option	Description	Default
expr/pow/minzerodistance	minimal distance from zero to enforce for child in bound tightening Range: [0, 1]	1e-09

5.34.4.104 heuristics

Option	Description	Default
Options for expert users		
heuristics/useuctsubscip	should setting of common subscip parameters include the activation of the UCT node selector? Range: boolean	0

5.34.4.105 heuristics/actconsdiving

Option	Description	Default
heuristics/actconsdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/actconsdiving/freq	frequency for calling primal heuristic <actconsdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/actconsdiving/freqofs	frequency offset for calling primal heuristic <actconsdiving> Range: {0, ..., 65534}	5
heuristics/actconsdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/actconsdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/actconsdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/actconsdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05
heuristics/actconsdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	1
Options for expert users		
heuristics/actconsdiving/maxdepth	maximal depth level to call primal heuristic <actconsdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/actconsdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/actconsdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	1
heuristics/actconsdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/actconsdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	1
heuristics/actconsdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/actconsdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/actconsdiving/priority	priority of heuristic <actconsdiving> Range: {-536870912, ..., 536870911}	-1003700

5.34.4.106 heuristics/adaptivediving

Option	Description	Default
heuristics/adaptivediving/bestsolweight	weight of incumbent solutions compared to other solutions in computation of LP iteration limit Range: [0, ∞]	10
heuristics/adaptivediving/epsilon	parameter that increases probability of exploration among divesets (only active if seltype is 'e') Range: [0, ∞]	1
heuristics/adaptivediving/freq	frequency for calling primal heuristic <adaptivediving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	5
heuristics/adaptivediving/freqofs	frequency offset for calling primal heuristic <adaptivediving> Range: {0, ..., 65534}	3
heuristics/adaptivediving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1500
heuristics/adaptivediving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.1
heuristics/adaptivediving/scoretype	score parameter for selection: minimize either average 'n'odes, LP 'i'terations,backtrack/'c'onflict ratio, 'd'epth, 1 / 's'olutions, or 1 / solutions'u'ccess Range: c, d, i, n, s, u	c
heuristics/adaptivediving/selconfidencecoeff	coefficient c to decrease initial confidence (calls + 1.0) / (calls + c) in scores Range: [1, 2.14748e+09]	10
heuristics/adaptivediving/seltype	selection strategy: (e)psilon-greedy, (w)eighted distribution, (n)ext diving Range: e, n, w	w
Options for expert users		
heuristics/adaptivediving/maxdepth	maximal depth level to call primal heuristic <adaptivediving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/adaptivediving/priority	priority of heuristic <adaptivediving> Range: {-536870912, ..., 536870911}	-70000
heuristics/adaptivediving/useadaptivecontext	should the heuristic use its own statistics, or shared statistics? Range: boolean	0

5.34.4.107 heuristics/alns

Option	Description	Default
heuristics/alns/freq	frequency for calling primal heuristic <alns> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	20

Option	Description	Default
heuristics/alns/freqofs	frequency offset for calling primal heuristic <alns> Range: {0, ..., 65534}	0
heuristics/alns/nodesofs	offset added to the nodes budget Range: {0, ..., ∞}	500
heuristics/alns/nodesquot	fraction of nodes compared to the main SCIP for budget computation Range: [0, 1]	0.1
heuristics/alns/nodesquotmin	lower bound fraction of nodes compared to the main SCIP for budget computation Range: [0, 1]	0
heuristics/alns/nsolslim	limit on the number of improving solutions in a sub-SCIP call Range: {-1, ..., ∞}	3
heuristics/alns/seed	initial random seed for bandit algorithms and random decisions by neighborhoods Range: {0, ..., ∞}	113
Options for expert users		
heuristics/alns/adjustfixingrate	should the heuristic adjust the target fixing rate based on the success? Range: boolean	1
heuristics/alns/adjustminimprove	should the factor by which the minimum improvement is bound be dynamically updated? Range: boolean	0
heuristics/alns/adjusttargetnodes	should the target nodes be dynamically adjusted? Range: boolean	1
heuristics/alns/alpha	parameter to increase the confidence width in UCB Range: [0, 100]	0.0016
heuristics/alns/banditalgo	the bandit algorithm: (u)pper confidence bounds, (e)xp.3, epsilon (g)reedy Range: u, e, g	u
heuristics/alns/beta	reward offset between 0 and 1 at every observation for Exp.3 Range: [0, 1]	0
heuristics/alns/copycuts	should cutting planes be copied to the sub-SCIP? Range: boolean	0
heuristics/alns/domorefixings	should the ALNS heuristic do more fixings by itself based on variable prioritization until the target fixing rate is reached? Range: boolean	1
heuristics/alns/eps	increase exploration in epsilon-greedy bandit algorithm Range: [0, 1]	0.468584
heuristics/alns/fixtol	tolerance by which the fixing rate may be missed without generic fixing Range: [0, 1]	0.1
heuristics/alns/gamma	weight between uniform (gamma ∼ 1) and weight driven (gamma ∼ 0) probability distribution for exp3 Range: [0, 1]	0.0704146
heuristics/alns/initduringroot	should the heuristic be executed multiple times during the root node? Range: boolean	0

Option	Description	Default
heuristics/alns/maxcallssamesol	number of allowed executions of the heuristic on the same incumbent solution (-1: no limit, 0: number of active neighborhoods) Range: {-1, ..., 100}	-1
heuristics/alns/maxdepth	maximal depth level to call primal heuristic <alns> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/alns/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/alns/minimprovehigh	upper bound for the minimal improvement over the incumbent Range: [0, 1]	0.01
heuristics/alns/minimprovelow	lower threshold for the minimal improvement over the incumbent Range: [0, 1]	0.01
heuristics/alns/minnodes	minimum number of nodes required to start a sub-SCIP Range: {0, ..., ∞}	50
heuristics/alns/priority	priority of heuristic <alns> Range: {-536870912, ..., 536870911}	-1100500
heuristics/alns/resetweights	should the bandit algorithms be reset when a new problem is read? Range: boolean	1
heuristics/alns/rewardbaseline	the reward baseline to separate successful and failed calls Range: [0, 0.99]	0.5
heuristics/alns/rewardcontrol	reward control to increase the weight of the simple solution indicator and decrease the weight of the closed gap reward Range: [0, 1]	0.8
heuristics/alns/rewardfilename	file name to store all rewards and the selection of the bandit Range: string	-
heuristics/alns/scalebyeffort	should the reward be scaled by the effort? Range: boolean	1
heuristics/alns/shownbstats	show statistics on neighborhoods? Range: boolean	0
heuristics/alns/startminimprove	initial factor by which ALNS should at least improve the incumbent Range: [0, 1]	0.01
heuristics/alns/subsciprandseeds	should random seeds of sub-SCIPs be altered to increase diversification? Range: boolean	0
heuristics/alns/targetnodefactor	factor by which target node number is eventually increased Range: [1, 100000]	1.05
heuristics/alns/unfixtol	tolerance by which the fixing rate may be exceeded without generic unfixing Range: [0, 1]	0.1
heuristics/alns/usedistances	distances from fixed variables be used for variable prioritization Range: boolean	1

Option	Description	Default
heuristics/alns/uselocalredcost	should local reduced costs be used for generic (un)fixing? Range: boolean	0
heuristics/alns/usepscost	should pseudo cost scores be used for variable priorization? Range: boolean	1
heuristics/alns/useredcost	should reduced cost scores be used for variable prioritization? Range: boolean	1
heuristics/alns/usesubscipheurs	should the heuristic activate other sub-SCIP heuristics during its search? Range: boolean	0
heuristics/alns/waitingnodes	number of nodes since last incumbent solution that the heuristic should wait Range: {0, ..., ∞}	25

5.34.4.108 heuristics/alns/crossover

Option	Description	Default
Options for expert users		
heuristics/alns/crossover/active	is this neighborhood active? Range: boolean	1
heuristics/alns/crossover/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/crossover/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/crossover/nsols	the number of solutions that crossover should combine Range: {2, ..., 10}	2
heuristics/alns/crossover/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.109 heuristics/alns/dins

Option	Description	Default
Options for expert users		
heuristics/alns/dins/active	is this neighborhood active? Range: boolean	1
heuristics/alns/dins/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/dins/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/dins/npoolsols	number of pool solutions where binary solution values must agree Range: {1, ..., 100}	5
heuristics/alns/dins/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.110 heuristics/alns/localbranching

Option	Description	Default
Options for expert users		
heuristics/alns/localbranching/active	is this neighborhood active? Range: boolean	1
heuristics/alns/localbranching/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/localbranching/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/localbranching/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.111 heuristics/alns/mutation

Option	Description	Default
Options for expert users		
heuristics/alns/mutation/active	is this neighborhood active? Range: boolean	1
heuristics/alns/mutation/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/mutation/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/mutation/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.112 heuristics/alns/proximity

Option	Description	Default
Options for expert users		
heuristics/alns/proximity/active	is this neighborhood active? Range: boolean	1
heuristics/alns/proximity/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/proximity/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/proximity/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.113 heuristics/alns/rens

Option	Description	Default
Options for expert users		
heuristics/alns/rens/active	is this neighborhood active? Range: boolean	1

Option	Description	Default
heuristics/alns/rens/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/rens/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/rens/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.114 heuristics/alns/rins

Option	Description	Default
Options for expert users		
heuristics/alns/rins/active	is this neighborhood active? Range: boolean	1
heuristics/alns/rins/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/rins/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/rins/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.115 heuristics/alns/trustregion

Option	Description	Default
heuristics/alns/trustregion/violpenalty	the penalty for each change in the binary variables from the candidate solution Range: [0, ∞]	100
Options for expert users		
heuristics/alns/trustregion/active	is this neighborhood active? Range: boolean	0
heuristics/alns/trustregion/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/trustregion/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3
heuristics/alns/trustregion/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.116 heuristics/alns/zeroobjective

Option	Description	Default
Options for expert users		
heuristics/alns/zeroobjective/active	is this neighborhood active? Range: boolean	1
heuristics/alns/zeroobjective/maxfixingrate	maximum fixing rate for this neighborhood Range: [0, 1]	0.9
heuristics/alns/zeroobjective/minfixingrate	minimum fixing rate for this neighborhood Range: [0, 1]	0.3

Option	Description	Default
heuristics/alns/zeroobjective/priority	positive call priority to initialize bandit algorithms Range: [0.01, 1]	1

5.34.4.117 heuristics/bound

Option	Description	Default
heuristics/bound/bound	to which bound should integer variables be fixed? ('l'ower, 'u'pper, or 'b'oth) Range: l, u, b	l
heuristics/bound/freq	frequency for calling primal heuristic <bound> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/bound/freqofs	frequency offset for calling primal heuristic <bound> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/bound/maxdepth	maximal depth level to call primal heuristic <bound> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/bound/maxproprounds	maximum number of propagation rounds during probing (-1 infinity, -2 parameter settings) Range: {-1, ..., 536870911}	0
heuristics/bound/onlywithoutsol	Should heuristic only be executed if no primal solution was found, yet? Range: boolean	1
heuristics/bound/priority	priority of heuristic <bound> Range: {-536870912, ..., 536870911}	-1107000

5.34.4.118 heuristics/clique

Option	Description	Default
heuristics/clique/freq	frequency for calling primal heuristic <clique> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/clique/freqofs	frequency offset for calling primal heuristic <clique> Range: {0, ..., 65534}	0
heuristics/clique/minintfixingrate	minimum percentage of integer variables that have to be fixable Range: [0, 1]	0.65
heuristics/clique/minmipfixingrate	minimum percentage of fixed variables in the sub-MIP Range: [0, 1]	0.65
heuristics/clique/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/clique/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
Options for expert users		

Option	Description	Default
heuristics/clique/copycuts	should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/clique/maxbacktracks	maximum number of backtracks during the fixing process Range: {-1, ..., 536870911}	10
heuristics/clique/maxdepth	maximal depth level to call primal heuristic <clique> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/clique/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/clique/maxproprounds	maximum number of propagation rounds during probing (-1 infinity) Range: {-1, ..., 536870911}	2
heuristics/clique/minimprove	factor by which clique heuristic should at least improve the incumbent Range: [0, 1]	0.01
heuristics/clique/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	500
heuristics/clique/priority	priority of heuristic <clique> Range: {-536870912, ..., 536870911}	5000
heuristics/clique/uselockfixings	should more variables be fixed based on variable locks if the fixing rate was not reached? Range: boolean	0

5.34.4.119 heuristics/coefdiving

Option	Description	Default
heuristics/coefdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/coefdiving/freq	frequency for calling primal heuristic <coefdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/coefdiving/freqofs	frequency offset for calling primal heuristic <coefdiving> Range: {0, ..., 65534}	1
heuristics/coefdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/coefdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/coefdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/coefdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05

Option	Description	Default
heuristics/coefdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	0
Options for expert users		
heuristics/coefdiving/maxdepth	maximal depth level to call primal heuristic <coefdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/coefdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/coefdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/coefdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/coefdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/coefdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/coefdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/coefdiving/priority	priority of heuristic <coefdiving> Range: {-536870912, ..., 536870911}	-1001000

5.34.4.120 heuristics/completesol

Option	Description	Default
heuristics/completesol/beforepresol	should the heuristic run before presolving? Range: boolean	1
heuristics/completesol/freq	frequency for calling primal heuristic <completesol> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/completesol/freqofs	frequency offset for calling primal heuristic <completesol> Range: {0, ..., 65534}	0
heuristics/completesol/ignorecont	should number of continuous variables be ignored? Range: boolean	0
heuristics/completesol/maxcontvars	maximal number of continuous variables after presolving Range: {-1, ..., ∞}	-1
heuristics/completesol/maxlpiter	maximal number of LP iterations (-1: no limit) Range: {-1, ..., ∞}	-1

Option	Description	Default
heuristics/completesol/maxproprounds	maximal number of iterations in propagation (-1: no limit) Range: {-1, ..., ∞}	10
heuristics/completesol/maxunknownrate	maximal rate of unknown solution values Range: [0, 1]	0.85
heuristics/completesol/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/completesol/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/completesol/solutions	heuristic stops, if the given number of improving solutions were found (-1: no limit) Range: {-1, ..., ∞}	5
Options for expert users		
heuristics/completesol/addallsols	should all subproblem solutions be added to the original SCIP? Range: boolean	0
heuristics/completesol/boundwidening	bound widening factor applied to continuous variables (0: fix variables to given solution values, 1: relax to global bounds) Range: [0, 1]	0.1
heuristics/completesol/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/completesol/maxdepth	maximal depth level to call primal heuristic <completesol> (-1: no limit) Range: {-1, ..., 65534}	0
heuristics/completesol/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/completesol/minimprove	factor by which the incumbent should be improved at least Range: [0, 1]	0.01
heuristics/completesol/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	50
heuristics/completesol/objweight	weight of the original objective function (1: only original objective) Range: [0.001, 1]	1
heuristics/completesol/priority	priority of heuristic <completesol> Range: {-536870912, ..., 536870911}	0

5.34.4.121 heuristics/conflictdiving

Option	Description	Default
heuristics/conflictdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1

Option	Description	Default
heuristics/conflictdiving/freq	frequency for calling primal heuristic <conflictdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/conflictdiving/freqofs	frequency offset for calling primal heuristic <conflictdiving> Range: {0, ..., 65534}	0
heuristics/conflictdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/conflictdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/conflictdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/conflictdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.15
heuristics/conflictdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	0
Options for expert users		
heuristics/conflictdiving/likecoef	perform rounding like coefficient diving Range: boolean	0
heuristics/conflictdiving/lockweight	weight used in a convex combination of conflict and variable locks Range: [0, 1]	0.75
heuristics/conflictdiving/maxdepth	maximal depth level to call primal heuristic <conflictdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/conflictdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/conflictdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/conflictdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/conflictdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/conflictdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/conflictdiving/maxviol	try to maximize the violation Range: boolean	1
heuristics/conflictdiving/minconflictlocks	minimal number of conflict locks per variable Range: {0, ..., ∞}	5

Option	Description	Default
heuristics/conflictdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/conflictdiving/priority	priority of heuristic <conflictdiving> Range: {-536870912, ..., 536870911}	-1000100

5.34.4.122 heuristics/crossover

Option	Description	Default
heuristics/crossover/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: {-1, ..., ∞}	-1
heuristics/crossover/freq	frequency for calling primal heuristic <crossover> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	30
heuristics/crossover/freqofs	frequency offset for calling primal heuristic <crossover> Range: {0, ..., 65534}	0
heuristics/crossover/minfixingrate	minimum percentage of integer variables that have to be fixed Range: [0, 1]	0.666
heuristics/crossover/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/crossover/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/crossover/nusedsols	number of solutions to be taken into account Range: {2, ..., ∞}	3
Options for expert users		
heuristics/crossover/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/crossover/dontwaitatroot	should the nwaitingnodes parameter be ignored at the root node? Range: boolean	0
heuristics/crossover/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/crossover/maxdepth	maximal depth level to call primal heuristic <crossover> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/crossover/maxnodes	maximum number of nodes to regard in the sub-problem Range: {0, ..., ∞}	5000
heuristics/crossover/minimprove	factor by which Crossover should at least improve the incumbent Range: [0, 1]	0.01
heuristics/crossover/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	50

Option	Description	Default
heuristics/crossover/nwaitingnodes	number of nodes without incumbent change that heuristic should wait Range: {0, ..., ∞}	200
heuristics/crossover/permute	should the subproblem be permuted to increase diversification? Range: boolean	0
heuristics/crossover/priority	priority of heuristic <crossover> Range: {-536870912, ..., 536870911}	-1104000
heuristics/crossover/randomization	should the choice which sols to take be randomized? Range: boolean	1
heuristics/crossover/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	0
heuristics/crossover/useuct	should uct node selection be used at the beginning of the search? Range: boolean	0

5.34.4.123 heuristics/dins

Option	Description	Default
heuristics/dins/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: {-1, ..., ∞}	3
heuristics/dins/freq	frequency for calling primal heuristic <dins> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/dins/freqofs	frequency offset for calling primal heuristic <dins> Range: {0, ..., 65534}	0
heuristics/dins/minfixingrate	minimum percentage of integer variables that have to be fixable Range: [0, 1]	0.3
heuristics/dins/minnodes	minimum number of nodes required to start the sub-problem Range: {0, ..., ∞}	50
heuristics/dins/neighborhoodsize	radius (using Manhattan metric) of the incumbent's neighborhood to be searched Range: {1, ..., ∞}	18
heuristics/dins/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	5000
heuristics/dins/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.05
heuristics/dins/solnum	number of pool-solutions to be checked for flag array update (for hard fixing of binary variables) Range: {1, ..., ∞}	5
Options for expert users		
heuristics/dins/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1

Option	Description	Default
heuristics/dins/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	1.5
heuristics/dins/maxdepth	maximal depth level to call primal heuristic <dins> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/dins/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/dins/minimprove	factor by which dins should at least improve the incumbent Range: [0, 1]	0.01
heuristics/dins/nwaitingnodes	number of nodes without incumbent change that heuristic should wait Range: {0, ..., ∞}	200
heuristics/dins/priority	priority of heuristic <dins> Range: {-536870912, ..., 536870911}	-1105000
heuristics/dins/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	0
heuristics/dins/useuct	should uct node selection be used at the beginning of the search? Range: boolean	0

5.34.4.124 heuristics/distributiondiving

Option	Description	Default
heuristics/distributiondiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/distributiondiving/freq	frequency for calling primal heuristic <distributiondiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/distributiondiving/freqofs	frequency offset for calling primal heuristic <distributiondiving> Range: {0, ..., 65534}	3
heuristics/distributiondiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/distributiondiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/distributiondiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/distributiondiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05

Option	Description	Default
heuristics/distributiondiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	1
Options for expert users		
heuristics/distributiondiving/maxdepth	maximal depth level to call primal heuristic <distributiondiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/distributiondiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/distributiondiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/distributiondiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/distributiondiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/distributiondiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/distributiondiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/distributiondiving/priority	priority of heuristic <distributiondiving> Range: {-536870912, ..., 536870911}	-1003300
heuristics/distributiondiving/scoreparam	the score;largest 'd'ifference, 'l'owest cumulative probability,'h'ighest c.p., 'v'otes lowest c.p., votes highest c.p.('w'), 'r'evolving Range: l, v, d, h, w, r	r

5.34.4.125 heuristics/dps

Option	Description	Default
heuristics/dps/freq	frequency for calling primal heuristic <dps> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/dps/freqofs	frequency offset for calling primal heuristic <dps> Range: {0, ..., 65534}	0
heuristics/dps/maxiterations	maximal number of iterations Range: {1, ..., ∞}	50
heuristics/dps/maxlinkscore	maximal linking score of used decomposition (equivalent to percentage of linking constraints) Range: [0, 1]	1
heuristics/dps/penalty	multiplier for absolute increase of penalty parameters (0: no increase) Range: [0, ∞]	100

Option	Description	Default
heuristics/dps/reoptimize	should the problem get reoptimized with the original objective function? Range: boolean	0
heuristics/dps/reuse	should solutions get reused in subproblems? Range: boolean	0
Options for expert users		
heuristics/dps/maxdepth	maximal depth level to call primal heuristic <dps> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/dps/priority	priority of heuristic <dps> Range: {-536870912, ..., 536870911}	75000

5.34.4.126 heuristics/dualval

Option	Description	Default
heuristics/dualval/dynamicdepth	says if and how the recursion depth is computed at runtime Range: {0, ..., 1}	0
heuristics/dualval/freq	frequency for calling primal heuristic <dualval> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/dualval/freqofs	frequency offset for calling primal heuristic <dualval> Range: {0, ..., 65534}	0
heuristics/dualval/heurverblevel	verblevel of the heuristic, default is 0 to display nothing Range: {0, ..., 4}	0
heuristics/dualval/lambdaobj	scaling factor for the objective function Range: [0, 1]	0
heuristics/dualval/lambdaslack	value added to objective of slack variables, must not be zero Range: [0.1, ∞]	1
heuristics/dualval/maxcalls	maximal number of recursive calls of the heuristic (if dynamicdepth is off) Range: {0, ..., ∞}	25
heuristics/dualval/maxequalranks	maximal number of variables that may have maximal rank, quit if there are more, turn off by setting -1 Range: {-1, ..., ∞}	50
heuristics/dualval/mingap	minimal gap for which we still run the heuristic, if gap is less we return without doing anything Range: [0, 100]	5
heuristics/dualval/nlpverblevel	verblevel of the nlp solver, can be 0 or 1 Range: {0, ..., 1}	0
heuristics/dualval/onlyleaves	disable the heuristic if it was not called at a leaf of the B&B tree Range: boolean	0
heuristics/dualval/rankvalue	number of ranks that should be displayed when the heuristic is called Range: {0, ..., ∞}	10

Option	Description	Default
heuristics/dualval/relaxcontvars	relax the continous variables Range: boolean	0
heuristics/dualval/relaxindicators	relax the indicator variables by introducing continuous copies Range: boolean	0
Options for expert users		
heuristics/dualval/forceimprovements	exit if objective doesn't improve Range: boolean	0
heuristics/dualval/maxdepth	maximal depth level to call primal heuristic <dualval> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/dualval/onlycheaper	add constraint to ensure that discrete vars are improving Range: boolean	1
heuristics/dualval/priority	priority of heuristic <dualval> Range: {-536870912, ..., 536870911}	-10

5.34.4.127 heuristics/farkasdiving

Option	Description	Default
heuristics/farkasdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/farkasdiving/freq	frequency for calling primal heuristic <farkasdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/farkasdiving/freqofs	frequency offset for calling primal heuristic <farkasdiving> Range: {0, ..., 65534}	0
heuristics/farkasdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/farkasdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	1
heuristics/farkasdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/farkasdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05
heuristics/farkasdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	0
Options for expert users		
heuristics/farkasdiving/checkcands	should diving candidates be checked before running? Range: boolean	0

Option	Description	Default
heuristics/farkasdiving/maxdepth	maximal depth level to call primal heuristic <farkasdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/farkasdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/farkasdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/farkasdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/farkasdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/farkasdiving/maxobjocc	maximal occurance factor of an objective coefficient Range: [0, 1]	1
heuristics/farkasdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/farkasdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/farkasdiving/objdynamism	minimal objective dynamism (log) to run Range: [0, ∞]	0.0001
heuristics/farkasdiving/priority	priority of heuristic <farkasdiving> Range: {-536870912, ..., 536870911}	-900000
heuristics/farkasdiving/rootsuccess	should the heuristic only run within the tree if at least one solution was found at the root node? Range: boolean	1
heuristics/farkasdiving/scalescore	should the score be scaled? Range: boolean	1
heuristics/farkasdiving/scaletype	scale score by [f]ractionality or [i]mpact on farkasproof Range: f, i	i

5.34.4.128 heuristics/feaspump

Option	Description	Default
heuristics/feaspump/alpha	initial weight of the objective function in the convex combination Range: [0, 1]	1
heuristics/feaspump/alphadiff	threshold difference for the convex parameter to perform perturbation Range: [0, 1]	1
heuristics/feaspump/beforecuts	should the feasibility pump be called at root node before cut separation? Range: boolean	1

Option	Description	Default
heuristics/feaspump/freq	frequency for calling primal heuristic <feaspump> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	20
heuristics/feaspump/freqofs	frequency offset for calling primal heuristic <feaspump> Range: {0, ..., 65534}	0
heuristics/feaspump/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/feaspump/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.01
heuristics/feaspump/neighborhoodsize	radius (using Manhattan metric) of the neighborhood to be searched in stage 3 Range: {1, ..., ∞}	18
heuristics/feaspump/objfactor	factor by which the regard of the objective is decreased in each round, 1.0 for dynamic Range: [0, 1]	0.1
heuristics/feaspump/pertsolfound	should a random perturbation be performed if a feasible solution was found? Range: boolean	1
heuristics/feaspump/stage3	should we solve a local branching sub-MIP if no solution could be found? Range: boolean	0
heuristics/feaspump/usefp20	should an iterative round-and-propagate scheme be used to find the integral points? Range: boolean	0
Options for expert users		
heuristics/feaspump/copycuts	should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/feaspump/cyclelength	maximum length of cycles to be checked explicitly in each round Range: {1, ..., 100}	3
heuristics/feaspump/maxdepth	maximal depth level to call primal heuristic <feaspump> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/feaspump/maxloops	maximal number of pumping loops (-1: no limit) Range: {-1, ..., ∞}	10000
heuristics/feaspump/maxsols	total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: {-1, ..., ∞}	10
heuristics/feaspump/maxstallloops	maximal number of pumping rounds without fractionality improvement (-1: no limit) Range: {-1, ..., ∞}	10
heuristics/feaspump/minflips	minimum number of random variables to flip, if a 1-cycle is encountered Range: {1, ..., ∞}	10
heuristics/feaspump/perturbfreq	number of iterations until a random perturbation is forced Range: {1, ..., ∞}	100

Option	Description	Default
heuristics/feaspump/priority	priority of heuristic <feaspump> Range: {-536870912, ..., 536870911}	-1000000

5.34.4.129 heuristics/fixandinfer

Option	Description	Default
heuristics/fixandinfer/freq	frequency for calling primal heuristic <fixandinfer> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/fixandinfer/freqofs	frequency offset for calling primal heuristic <fixandinfer> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/fixandinfer/maxdepth	maximal depth level to call primal heuristic <fixandinfer> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/fixandinfer/minfixings	minimal number of fixings to apply before dive may be aborted Range: {0, ..., ∞}	100
heuristics/fixandinfer/priority	priority of heuristic <fixandinfer> Range: {-536870912, ..., 536870911}	-500000
heuristics/fixandinfer/proprounds	maximal number of propagation rounds in probing subproblems (-1: no limit, 0: auto) Range: {-1, ..., ∞}	0

5.34.4.130 heuristics/fracdiving

Option	Description	Default
heuristics/fracdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/fracdiving/freq	frequency for calling primal heuristic <fracdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/fracdiving/freqofs	frequency offset for calling primal heuristic <fracdiving> Range: {0, ..., 65534}	3
heuristics/fracdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/fracdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/fracdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/fracdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05

Option	Description	Default
heuristics/fracdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	0
Options for expert users		
heuristics/fracdiving/maxdepth	maximal depth level to call primal heuristic <fracdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/fracdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/fracdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/fracdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/fracdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/fracdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/fracdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/fracdiving/priority	priority of heuristic <fracdiving> Range: {-536870912, ..., 536870911}	-1003000

5.34.4.131 heuristics/gins

Option	Description	Default
heuristics/gins/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: {-1, ..., ∞}	3
heuristics/gins/freq	frequency for calling primal heuristic <gins> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	20
heuristics/gins/freqofs	frequency offset for calling primal heuristic <gins> Range: {0, ..., 65534}	8
heuristics/gins/maxdistance	maximum distance to selected variable to enter the subproblem, or -1 to select the distance that best approximates the minimum fixing rate from below Range: {-1, ..., ∞}	3
heuristics/gins/minfixingrate	percentage of integer variables that have to be fixed Range: [1e-06, 0.999999]	0.66
heuristics/gins/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500

Option	Description	Default
heuristics/gins/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.15
Options for expert users		
heuristics/gins/consecutiveblocks	should blocks be treated consecutively (sorted by ascending label?) Range: boolean	1
heuristics/gins/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/gins/fixcontvars	should continuous variables outside the neighborhoods be fixed? Range: boolean	0
heuristics/gins/maxdepth	maximal depth level to call primal heuristic <gins> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/gins/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/gins/minimprove	factor by which gins should at least improve the incumbent Range: [0, 1]	0.01
heuristics/gins/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	50
heuristics/gins/nwaitingnodes	number of nodes without incumbent change that heuristic should wait Range: {0, ..., ∞}	100
heuristics/gins/overlap	overlap of blocks between runs - 0.0: no overlap, 1.0: shift by only 1 block Range: [0, 1]	0
heuristics/gins/potential	the reference point to compute the neighborhood potential: (r)oot, (l)ocal lp, or (p)seudo solution Range: l, p, r	r
heuristics/gins/priority	priority of heuristic <gins> Range: {-536870912, ..., 536870911}	-1103000
heuristics/gins/relaxdenseconss	should dense constraints (at least as dense as 1 - minfixingrate) be ignored by connectivity graph? Range: boolean	0
heuristics/gins/rollhorizonlimfac	limiting percentage for variables already used in sub-SCIPs to terminate rolling horizon approach Range: [0, 1]	0.4
heuristics/gins/usedecomp	should user decompositions be considered, if available? Range: boolean	1
heuristics/gins/usedecomprollhorizon	should user decompositions be considered for initial selection in rolling horizon, if available? Range: boolean	0
heuristics/gins/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	0

Option	Description	Default
heuristics/gins/userollinghorizon	should the heuristic solve a sequence of sub-MIP's around the first selected variable Range: boolean	1
heuristics/gins/useselfallback	should random initial variable selection be used if decomposition was not successful? Range: boolean	1

5.34.4.132 heuristics/guideddiving

Option	Description	Default
heuristics/guideddiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/guideddiving/freq	frequency for calling primal heuristic <guideddiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/guideddiving/freqofs	frequency offset for calling primal heuristic <guideddiving> Range: {0, ..., 65534}	7
heuristics/guideddiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/guideddiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/guideddiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/guideddiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05
heuristics/guideddiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	0
Options for expert users		
heuristics/guideddiving/maxdepth	maximal depth level to call primal heuristic <guideddiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/guideddiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/guideddiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	1
heuristics/guideddiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8

Option	Description	Default
heuristics/guideddiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	1
heuristics/guideddiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/guideddiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/guideddiving/priority	priority of heuristic <guideddiving> Range: {-536870912, ..., 536870911}	-1007000

5.34.4.133 heuristics/indicator

Option	Description	Default
heuristics/indicator/freq	frequency for calling primal heuristic <indicator> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	1
heuristics/indicator/freqofs	frequency offset for calling primal heuristic <indicator> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/indicator/improvesols	Try to improve other solutions by one-opt? Range: boolean	0
heuristics/indicator/maxdepth	maximal depth level to call primal heuristic <indicator> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/indicator/oneopt	whether the one-opt heuristic should be started Range: boolean	0
heuristics/indicator/priority	priority of heuristic <indicator> Range: {-536870912, ..., 536870911}	-20200

5.34.4.134 heuristics/intdiving

Option	Description	Default
heuristics/intdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/intdiving/freq	frequency for calling primal heuristic <intdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/intdiving/freqofs	frequency offset for calling primal heuristic <intdiving> Range: {0, ..., 65534}	9
heuristics/intdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/intdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05
Options for expert users		

Option	Description	Default
heuristics/intdiving/maxdepth	maximal depth level to call primal heuristic <intdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/intdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/intdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/intdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/intdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/intdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/intdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/intdiving/priority	priority of heuristic <intdiving> Range: {-536870912, ..., 536870911}	-1003500

5.34.4.135 heuristics/intshifting

Option	Description	Default
heuristics/intshifting/freq	frequency for calling primal heuristic <intshifting> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/intshifting/freqofs	frequency offset for calling primal heuristic <intshifting> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/intshifting/maxdepth	maximal depth level to call primal heuristic <intshifting> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/intshifting/priority	priority of heuristic <intshifting> Range: {-536870912, ..., 536870911}	-10000

5.34.4.136 heuristics/linesearchdiving

Option	Description	Default
heuristics/linesearchdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/linesearchdiving/freq	frequency for calling primal heuristic <linesearchdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10

Option	Description	Default
heuristics/linesearchdiving/freqofs	frequency offset for calling primal heuristic <linesearchdiving> Range: {0, ..., 65534}	6
heuristics/linesearchdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/linesearchdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/linesearchdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/linesearchdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05
heuristics/linesearchdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	0
Options for expert users		
heuristics/linesearchdiving/maxdepth	maximal depth level to call primal heuristic <linesearchdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/linesearchdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/linesearchdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/linesearchdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/linesearchdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/linesearchdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/linesearchdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/linesearchdiving/priority	priority of heuristic <linesearchdiving> Range: {-536870912, ..., 536870911}	-1006000

5.34.4.137 heuristics/localbranching

Option	Description	Default
heuristics/localbranching/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: {-1, ..., ∞}	3

Option	Description	Default
heuristics/localbranching/freq	frequency for calling primal heuristic <localbranching> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/localbranching/freqofs	frequency offset for calling primal heuristic <localbranching> Range: {0, ..., 65534}	0
heuristics/localbranching/neighborhoodsize	radius (using Manhattan metric) of the incumbent's neighborhood to be searched Range: {1, ..., ∞}	18
heuristics/localbranching/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	1000
heuristics/localbranching/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.05
Options for expert users		
heuristics/localbranching/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/localbranching/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	1.5
heuristics/localbranching/maxdepth	maximal depth level to call primal heuristic <localbranching> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/localbranching/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	10000
heuristics/localbranching/minimprove	factor by which localbranching should at least improve the incumbent Range: [0, 1]	0.01
heuristics/localbranching/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	1000
heuristics/localbranching/nwaitingnodes	number of nodes without incumbent change that heuristic should wait Range: {0, ..., ∞}	200
heuristics/localbranching/priority	priority of heuristic <localbranching> Range: {-536870912, ..., 536870911}	-1102000
heuristics/localbranching/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	0

5.34.4.138 heuristics/locks

Option	Description	Default
heuristics/locks/freq	frequency for calling primal heuristic <locks> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0

Option	Description	Default
heuristics/locks/freqofs	frequency offset for calling primal heuristic <locks> Range: {0, ..., 65534}	0
heuristics/locks/minfixingrate	minimum percentage of integer variables that have to be fixable Range: [0, 1]	0.65
heuristics/locks/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/locks/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/locks/roundupprobability	probability for rounding a variable up in case of ties Range: [0, 1]	0.67
Options for expert users		
heuristics/locks/copycuts	should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/locks/maxdepth	maximal depth level to call primal heuristic <locks> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/locks/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/locks/maxproprounds	maximum number of propagation rounds to be performed in each propagation call (-1: no limit, -2: parameter settings) Range: {-2, ..., ∞}	2
heuristics/locks/minfixingratelp	minimum fixing rate over all variables (including continuous) to solve LP Range: [0, 1]	0
heuristics/locks/minimprove	factor by which locks heuristic should at least improve the incumbent Range: [0, 1]	0.01
heuristics/locks/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	500
heuristics/locks/priority	priority of heuristic <locks> Range: {-536870912, ..., 536870911}	3000
heuristics/locks/updatelocks	should the locks be updated based on LP rows? Range: boolean	1
heuristics/locks/usefinalsubmip	should a final sub-MIP be solved to costruct a feasible solution if the LP was not roundable? Range: boolean	1

5.34.4.139 heuristics/lpface

Option	Description	Default
heuristics/lpface/freq	frequency for calling primal heuristic <lpface> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	15

Option	Description	Default
heuristics/lpface/freqofs	frequency offset for calling primal heuristic <lpface> Range: {0, ..., 65534}	0
heuristics/lpface/minfixingrate	required percentage of fixed integer variables in sub-MIP to run Range: [0, 1]	0.1
heuristics/lpface/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	200
heuristics/lpface/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
Options for expert users		
heuristics/lpface/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/lpface/dualbasisequations	should dually nonbasic rows be turned into equations? Range: boolean	0
heuristics/lpface/keepsubscip	should the heuristic continue solving the same sub-SCIP? Range: boolean	0
heuristics/lpface/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/lpface/maxdepth	maximal depth level to call primal heuristic <lpface> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/lpface/maxnodes	maximum number of nodes to regard in the sub-problem Range: {0, ..., ∞}	5000
heuristics/lpface/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	50
heuristics/lpface/minpathlen	the minimum active search tree path length along which lower bound hasn't changed before heuristic becomes active Range: {0, ..., 65531}	5
heuristics/lpface/priority	priority of heuristic <lpface> Range: {-536870912, ..., 536870911}	-1104010
heuristics/lpface/subscipobjective	objective function in the sub-SCIP: (z)ero, (r)oot-LP-difference, (i)nference, LP (f)ractionality, (o)riginal Range: f, o, r, z, i	z
heuristics/lpface/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	1

5.34.4.140 heuristics/mpec

Option	Description	Default
heuristics/mpec/freq	frequency for calling primal heuristic <mpec> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	50
heuristics/mpec/freqofs	frequency offset for calling primal heuristic <mpec> Range: {0, ..., 65534}	0
heuristics/mpec/inittheta	initial regularization right-hand side value Range: [0, 0.25]	0.125
heuristics/mpec/maxiter	maximum number of iterations of the MPEC loop Range: {0, ..., ∞}	100
heuristics/mpec/maxnlpcost	maximum cost available for solving NLPs per call of the heuristic Range: [0, ∞]	1e+08
heuristics/mpec/maxnlpiter	maximum number of NLP iterations per solve Range: {0, ..., ∞}	500
heuristics/mpec/maxnunsucc	maximum number of consecutive calls for which the heuristic did not find an improving solution Range: {0, ..., ∞}	10
heuristics/mpec/mingapleft	minimum amount of gap left in order to call the heuristic Range: [0, ∞]	0.05
heuristics/mpec/minimprove	factor by which heuristic should at least improve the incumbent Range: [0, 1]	0.01
heuristics/mpec/sigma	regularization update factor Range: [0, 1]	0.5
heuristics/mpec/subnlptrigger	maximum number of NLP iterations per solve Range: [0, 1]	0.001
Options for expert users		
heuristics/mpec/maxdepth	maximal depth level to call primal heuristic <mpec> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/mpec/priority	priority of heuristic <mpec> Range: {-536870912, ..., 536870911}	-2050000

5.34.4.141 heuristics/multistart

Option	Description	Default
heuristics/multistart/freq	frequency for calling primal heuristic <multistart> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/multistart/freqofs	frequency offset for calling primal heuristic <multistart> Range: {0, ..., 65534}	0
heuristics/multistart/gradlimit	limit for gradient computations for all improvePoint() calls (0 for no limit) Range: [0, ∞]	5e+06
heuristics/multistart/maxboundsize	maximum variable domain size for unbounded variables Range: [0, ∞]	20000
heuristics/multistart/maxiter	number of iterations to reduce the maximum violation of a point Range: {0, ..., ∞}	300

Option	Description	Default
heuristics/multistart/maxncluster	maximum number of considered clusters per heuristic call Range: {0, ..., ∞}	3
heuristics/multistart/maxreldist	maximum distance between two points in the same cluster Range: [0, ∞]	0.15
heuristics/multistart/minimprfac	minimum required improving factor to proceed in improvement of a single point Range: real	0.05
heuristics/multistart/minimpriter	number of iteration when checking the minimum improvement Range: {1, ..., ∞}	10
heuristics/multistart/nrndpoints	number of random points generated per execution call Range: {0, ..., ∞}	100
heuristics/multistart/onlynlps	should the heuristic run only on continuous problems? Range: boolean	1
Options for expert users		
heuristics/multistart/maxdepth	maximal depth level to call primal heuristic <multistart> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/multistart/priority	priority of heuristic <multistart> Range: {-536870912, ..., 536870911}	-2100000

5.34.4.142 heuristics/mutation

Option	Description	Default
heuristics/mutation/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: {-1, ..., ∞}	-1
heuristics/mutation/freq	frequency for calling primal heuristic <mutation> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/mutation/freqofs	frequency offset for calling primal heuristic <mutation> Range: {0, ..., 65534}	8
heuristics/mutation/minfixingrate	percentage of integer variables that have to be fixed Range: [1e-06, 0.999999]	0.8
heuristics/mutation/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/mutation/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
Options for expert users		
heuristics/mutation/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/mutation/maxdepth	maximal depth level to call primal heuristic <mutation> (-1: no limit) Range: {-1, ..., 65534}	-1

Option	Description	Default
heuristics/mutation/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/mutation/minimprove	factor by which mutation should at least improve the incumbent Range: [0, 1]	0.01
heuristics/mutation/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	500
heuristics/mutation/nwaitingnodes	number of nodes without incumbent change that heuristic should wait Range: {0, ..., ∞}	200
heuristics/mutation/priority	priority of heuristic <mutation> Range: {-536870912, ..., 536870911}	-1103010
heuristics/mutation/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	0
heuristics/mutation/useuct	should uct node selection be used at the beginning of the search? Range: boolean	0

5.34.4.143 heuristics/nlpdiving

Option	Description	Default
heuristics/nlpdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/nlpdiving/fixquot	percentage of fractional variables that should be fixed before the next NLP solve Range: [0, 1]	0.2
heuristics/nlpdiving/freq	frequency for calling primal heuristic <nlpdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/nlpdiving/freqofs	frequency offset for calling primal heuristic <nlpdiving> Range: {0, ..., 65534}	3
heuristics/nlpdiving/maxfeasnlps	maximal number of NLPs with feasible solution to solve during one dive Range: {1, ..., ∞}	10
heuristics/nlpdiving/maxnlpiterabs	minimial absolute number of allowed NLP iterations Range: {0, ..., ∞}	200
heuristics/nlpdiving/maxnlpiterrel	additional allowed number of NLP iterations relative to successfully found solutions Range: {0, ..., ∞}	10
heuristics/nlpdiving/minsuccquot	heuristic will not run if less then this percentage of calls succeeded (0.0: no limit) Range: [0, 1]	0.1
heuristics/nlpdiving/nlpfastfail	should the NLP solver stop early if it converges slow? Range: boolean	1

Option	Description	Default
heuristics/nlpdiving/prefercover	should variables in a minimal cover be preferred? Range: boolean	1
heuristics/nlpdiving/solvesubmip	should a sub-MIP be solved if all cover variables are fixed? Range: boolean	0
heuristics/nlpdiving/varselrule	which variable selection should be used? ('f'ractionality, 'c'oefficient, 'p'seudocost, 'g'uided, 'd'ouble, 'v'eclen) Range: f, c, p, g, d, v	d
Options for expert users		
heuristics/nlpdiving/lp	should the LP relaxation be solved before the NLP relaxation? Range: boolean	0
heuristics/nlpdiving/maxdepth	maximal depth level to call primal heuristic <nlpdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/nlpdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/nlpdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/nlpdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/nlpdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/nlpdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/nlpdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/nlpdiving/nlpstart	which point should be used as starting point for the NLP solver? ('n'one, last 'f'easible, from dive's'tart) Range: f, n, s	s
heuristics/nlpdiving/preferlpfracs	prefer variables that are also fractional in LP solution? Range: boolean	0
heuristics/nlpdiving/priority	priority of heuristic <nlpdiving> Range: {-536870912, ..., 536870911}	-1003010

5.34.4.144 heuristics/objpscostdiving

Option	Description	Default
heuristics/objpscostdiving/freq	frequency for calling primal heuristic <objpscostdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	20

Option	Description	Default
heuristics/objpscostdiving/freqofs	frequency offset for calling primal heuristic <objpscostdiving> Range: {0, ..., 65534}	4
heuristics/objpscostdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/objpscostdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to total iteration number Range: [0, 1]	0.01
Options for expert users		
heuristics/objpscostdiving/depthfac	maximal diving depth: number of binary/integer variables times depthfac Range: [0, ∞]	0.5
heuristics/objpscostdiving/depthfacnosol	maximal diving depth factor if no feasible solution was found yet Range: [0, ∞]	2
heuristics/objpscostdiving/maxdepth	maximal depth level to call primal heuristic <objpscostdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/objpscostdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/objpscostdiving/maxsols	total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: {-1, ..., ∞}	-1
heuristics/objpscostdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/objpscostdiving/priority	priority of heuristic <objpscostdiving> Range: {-536870912, ..., 536870911}	-1004000

5.34.4.145 heuristics/octane

Option	Description	Default
heuristics/octane/freq	frequency for calling primal heuristic <octane> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/octane/freqofs	frequency offset for calling primal heuristic <octane> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/octane/ffirst	number of 0-1-points to be tested at first whether they violate a common row Range: {1, ..., ∞}	10
heuristics/octane/fmax	number of 0-1-points to be tested as possible solutions by OCTANE Range: {1, ..., ∞}	100
heuristics/octane/maxdepth	maximal depth level to call primal heuristic <octane> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/octane/priority	priority of heuristic <octane> Range: {-536870912, ..., 536870911}	-1008000
heuristics/octane/useavgnbray	should the weighted average of the nonbasic cone be used as one ray direction? Range: boolean	1

Option	Description	Default
heuristics/octane/useavgray	should the average of the basic cone be used as one ray direction? Range: boolean	1
heuristics/octane/useavgwgtray	should the weighted average of the basic cone be used as one ray direction? Range: boolean	1
heuristics/octane/usediffray	should the difference between the root solution and the current LP solution be used as one ray direction? Range: boolean	0
heuristics/octane/usefracspace	execute OCTANE only in the space of fractional variables (TRUE) or in the full space? Range: boolean	1
heuristics/octane/useobjray	should the inner normal of the objective be used as one ray direction? Range: boolean	1

5.34.4.146 heuristics/ofins

Option	Description	Default
heuristics/ofins/freq	frequency for calling primal heuristic <ofins> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/ofins/freqofs	frequency offset for calling primal heuristic <ofins> Range: {0, ..., 65534}	0
heuristics/ofins/maxchange	maximal rate of change per coefficient to get fixed Range: [0, 1]	0.04
heuristics/ofins/maxchangerate	maximal rate of changed coefficients Range: [0, 1]	0.5
heuristics/ofins/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/ofins/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
Options for expert users		
heuristics/ofins/addallsols	should all subproblem solutions be added to the original SCIP? Range: boolean	0
heuristics/ofins/copycuts	should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/ofins/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/ofins/maxdepth	maximal depth level to call primal heuristic <ofins> (-1: no limit) Range: {-1, ..., 65534}	0
heuristics/ofins/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/ofins/minimprove	factor by which RENS should at least improve the incumbent Range: [0, 1]	0.01

Option	Description	Default
heuristics/ofins/minnodes	minimum number of nodes required to start the sub-problem Range: {0, ..., ∞}	50
heuristics/ofins/priority	priority of heuristic <ofins> Range: {-536870912, ..., 536870911}	60000

5.34.4.147 heuristics/oneopt

Option	Description	Default
heuristics/oneopt/freq	frequency for calling primal heuristic <oneopt> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	1
heuristics/oneopt/freqofs	frequency offset for calling primal heuristic <oneopt> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/oneopt/beforepresol	should the heuristic be called before presolving? Range: boolean	0
heuristics/oneopt/duringroot	should the heuristic be called before and during the root node? Range: boolean	1
heuristics/oneopt/forcelpconstruction	should the construction of the LP be forced even if LP solving is deactivated? Range: boolean	0
heuristics/oneopt/maxdepth	maximal depth level to call primal heuristic <oneopt> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/oneopt/priority	priority of heuristic <oneopt> Range: {-536870912, ..., 536870911}	-20000
heuristics/oneopt/useloop	should the heuristic continue to run as long as improvements are found? Range: boolean	1
heuristics/oneopt/weightedobj	should the objective be weighted with the potential shifting value when sorting the shifting candidates? Range: boolean	1

5.34.4.148 heuristics/padm

Option	Description	Default
heuristics/padm/assignlinking	should linking constraints be assigned? Range: boolean	1
heuristics/padm/freq	frequency for calling primal heuristic <padm> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/padm/freqofs	frequency offset for calling primal heuristic <padm> Range: {0, ..., 65534}	0
heuristics/padm/reoptimize	should the problem get reoptimized with the original objective function? Range: boolean	1

Option	Description	Default
heuristics/padm/timing	should the heuristic run before or after the processing of the node? (0: before, 1: after, 2: both) Range: {0, ..., 2}	0
Options for expert users		
heuristics/padm/admiterations	maximal number of ADM iterations in each penalty loop Range: {1, ..., 100}	4
heuristics/padm/gap	mipgap at start Range: [0, 16]	2
heuristics/padm/maxdepth	maximal depth level to call primal heuristic <padm> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/padm/maxnodes	maximum number of nodes to regard in all subproblems Range: {0, ..., ∞}	5000
heuristics/padm/minnodes	minimum number of nodes to regard in one subproblem Range: {0, ..., ∞}	50
heuristics/padm/nodefac	factor to control nodelimits of subproblems Range: [0, 0.99]	0.8
heuristics/padm/original	should the original problem be used? This is only for testing and not recommended! Range: boolean	0
heuristics/padm/penaltyiterations	maximal number of penalty iterations Range: {1, ..., 100000}	100
heuristics/padm/priority	priority of heuristic <padm> Range: {-536870912, ..., 536870911}	70000
heuristics/padm/scaling	enable sigmoid rescaling of penalty parameters Range: boolean	1

5.34.4.149 heuristics/proximity

Option	Description	Default
heuristics/proximity/freq	frequency for calling primal heuristic <proximity> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/proximity/freqofs	frequency offset for calling primal heuristic <proximity> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/proximity/binvarquot	threshold for percentage of binary variables required to start Range: [0, 1]	0.1
heuristics/proximity/lpitersquot	quotient of sub-MIP LP iterations with respect to LP iterations so far Range: [0, 1]	0.2
heuristics/proximity/maxdepth	maximal depth level to call primal heuristic <proximity> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/proximity/maxlpiters	maximum number of LP iterations to be performed in the subproblem Range: {-1, ..., ∞}	100000

Option	Description	Default
heuristics/proximity/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	10000
heuristics/proximity/mingap	minimum primal-dual gap for which the heuristic is executed Range: [0, ∞]	0.01
heuristics/proximity/minimprove	factor by which proximity should at least improve the incumbent Range: [0, 1]	0.02
heuristics/proximity/minlpiters	minimum number of LP iterations performed in subproblem Range: {0, ..., ∞}	200
heuristics/proximity/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	1
heuristics/proximity/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	50
heuristics/proximity/nodesquot	sub-MIP node limit w.r.t number of original nodes Range: [0, ∞]	0.1
heuristics/proximity/priority	priority of heuristic <proximity> Range: {-536870912, ..., 536870911}	-2000000
heuristics/proximity/restart	should the heuristic immediately run again on its newly found solution? Range: boolean	1
heuristics/proximity/usefinallp	should the heuristic solve a final LP in case of continuous objective variables? Range: boolean	0
heuristics/proximity/uselprows	should subproblem be constructed based on LP row information? Range: boolean	0
heuristics/proximity/useuct	should uct node selection be used at the beginning of the search? Range: boolean	0
heuristics/proximity/waitingnodes	waiting nodes since last incumbent before heuristic is executed Range: {0, ..., ∞}	100

5.34.4.150 heuristics/pscostdiving

Option	Description	Default
heuristics/pscostdiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/pscostdiving/freq	frequency for calling primal heuristic <pscostdiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/pscostdiving/freqofs	frequency offset for calling primal heuristic <pscostdiving> Range: {0, ..., 65534}	2

Option	Description	Default
heuristics/pscostdiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/pscostdiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/pscostdiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/pscostdiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05
heuristics/pscostdiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	1
Options for expert users		
heuristics/pscostdiving/maxdepth	maximal depth level to call primal heuristic <pscostdiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/pscostdiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/pscostdiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/pscostdiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/pscostdiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/pscostdiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/pscostdiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/pscostdiving/priority	priority of heuristic <pscostdiving> Range: {-536870912, ..., 536870911}	-1002000

5.34.4.151 heuristics/randrounding

Option	Description	Default
heuristics/randrounding/freq	frequency for calling primal heuristic <randrounding> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	20
heuristics/randrounding/freqofs	frequency offset for calling primal heuristic <randrounding> Range: {0, ..., 65534}	0

Option	Description	Default
Options for expert users		
heuristics/randrounding/maxdepth	maximal depth level to call primal heuristic <randrounding> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/randrounding/maxproprounds	limit of rounds for each propagation call Range: {-1, ..., ∞}	1
heuristics/randrounding/oncepernode	should the heuristic only be called once per node? Range: boolean	0
heuristics/randrounding/priority	priority of heuristic <randrounding> Range: {-536870912, ..., 536870911}	-200
heuristics/randrounding/propagateonlyroot	should the probing part of the heuristic be applied exclusively at the root node? Range: boolean	1
heuristics/randrounding/usesimplerounding	should the heuristic apply the variable lock strategy of simple rounding, if possible? Range: boolean	0

5.34.4.152 heuristics/rens

Option	Description	Default
heuristics/rens/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: {-1, ..., ∞}	-1
heuristics/rens/freq	frequency for calling primal heuristic <rens> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/rens/freqofs	frequency offset for calling primal heuristic <rens> Range: {0, ..., 65534}	0
heuristics/rens/minfixingrate	minimum percentage of integer variables that have to be fixable Range: [0, 1]	0.5
heuristics/rens/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/rens/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/rens/startsol	solution that is used for fixing values ('l'p relaxation, 'n'lp relaxation) Range: n, l	l
Options for expert users		
heuristics/rens/addallsols	should all subproblem solutions be added to the original SCIP? Range: boolean	0
heuristics/rens/binarybounds	should general integers get binary bounds [floor(.),ceil(.)] ? Range: boolean	1
heuristics/rens/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1

Option	Description	Default
heuristics/rens/extratime	should the RENS sub-CIP get its own full time limit? This is only for testing and not recommended! Range: boolean	0
heuristics/rens/fullscale	should the RENS sub-CIP be solved with cuts, conflicts, strong branching,... This is only for testing and not recommended! Range: boolean	0
heuristics/rens/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/rens/maxdepth	maximal depth level to call primal heuristic <rens> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/rens/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/rens/minimprove	factor by which RENS should at least improve the incumbent Range: [0, 1]	0.01
heuristics/rens/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	50
heuristics/rens/priority	priority of heuristic <rens> Range: {-536870912, ..., 536870911}	-1100000
heuristics/rens/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	0
heuristics/rens/useuct	should uct node selection be used at the beginning of the search? Range: boolean	0

5.34.4.153 heuristics/reoptsols

Option	Description	Default
heuristics/reoptsols/freq	frequency for calling primal heuristic <reoptsols> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/reoptsols/freqofs	frequency offset for calling primal heuristic <reoptsols> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/reoptsols/maxdepth	maximal depth level to call primal heuristic <reoptsols> (-1: no limit) Range: {-1, ..., 65534}	0
heuristics/reoptsols/maxruns	check solutions of the last k runs. (-1: all) Range: {-1, ..., ∞}	-1
heuristics/reoptsols/maxsols	maximal number solutions which should be checked. (-1: all) Range: {-1, ..., ∞}	1000
heuristics/reoptsols/priority	priority of heuristic <reoptsols> Range: {-536870912, ..., 536870911}	40000

5.34.4.154 heuristics/repair

Option	Description	Default
heuristics/repair/filename	file name of a solution to be used as infeasible starting point, [-] if not available Range: string	-
heuristics/repair/freq	frequency for calling primal heuristic <repair> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/repair/freqofs	frequency offset for calling primal heuristic <repair> Range: {0, ..., 65534}	0
heuristics/repair/minfixingrate	minimum percentage of integer variables that have to be fixed Range: [0, 1]	0.3
heuristics/repair/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/repair/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/repair/roundit	True : fractional variables which are not fractional in the given solution are rounded, FALSE : solving process of this heuristic is stopped. Range: boolean	1
heuristics/repair/useobjfactor	should a scaled objective function for original variables be used in repair subproblem? Range: boolean	0
heuristics/repair/useslackvars	should slack variables be used in repair subproblem? Range: boolean	0
heuristics/repair/usevarfix	should variable fixings be used in repair subproblem? Range: boolean	1
Options for expert users		
heuristics/repair/alpha	factor for the potential of var fixings Range: [0, 100]	2
heuristics/repair/maxdepth	maximal depth level to call primal heuristic <repair> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/repair/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/repair/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	50
heuristics/repair/priority	priority of heuristic <repair> Range: {-536870912, ..., 536870911}	-20

5.34.4.155 heuristics/rins

Option	Description	Default
heuristics/rins/freq	frequency for calling primal heuristic <rins> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	25

Option	Description	Default
heuristics/rins/freqofs	frequency offset for calling primal heuristic <rins> Range: {0, ..., 65534}	0
heuristics/rins/minfixingrate	minimum percentage of integer variables that have to be fixed Range: [0, 1]	0.3
heuristics/rins/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/rins/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.3
Options for expert users		
heuristics/rins/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/rins/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	2
heuristics/rins/maxdepth	maximal depth level to call primal heuristic <rins> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/rins/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	5000
heuristics/rins/minimprove	factor by which rins should at least improve the incumbent Range: [0, 1]	0.01
heuristics/rins/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	50
heuristics/rins/nwaitingnodes	number of nodes without incumbent change that heuristic should wait Range: {0, ..., ∞}	200
heuristics/rins/priority	priority of heuristic <rins> Range: {-536870912, ..., 536870911}	-1101000
heuristics/rins/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	0
heuristics/rins/useuct	should uct node selection be used at the beginning of the search? Range: boolean	0

5.34.4.156 heuristics/rootsoldiving

Option	Description	Default
heuristics/rootsoldiving/freq	frequency for calling primal heuristic <rootsoldiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	20
heuristics/rootsoldiving/freqofs	frequency offset for calling primal heuristic <rootsoldiving> Range: {0, ..., 65534}	5

Option	Description	Default
heuristics/rootsoldiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/rootsoldiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.01
Options for expert users		
heuristics/rootsoldiving/alpha	soft rounding factor to fade out objective coefficients Range: [0, 1]	0.9
heuristics/rootsoldiving/depthfac	maximal diving depth: number of binary/integer variables times depthfac Range: [0, ∞]	0.5
heuristics/rootsoldiving/depthfacnosol	maximal diving depth factor if no feasible solution was found yet Range: [0, ∞]	2
heuristics/rootsoldiving/maxdepth	maximal depth level to call primal heuristic <rootsoldiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/rootsoldiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/rootsoldiving/maxsols	total number of feasible solutions found up to which heuristic is called (-1: no limit) Range: {-1, ..., ∞}	-1
heuristics/rootsoldiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/rootsoldiving/priority	priority of heuristic <rootsoldiving> Range: {-536870912, ..., 536870911}	-1005000

5.34.4.157 heuristics/rounding

Option	Description	Default
heuristics/rounding/freq	frequency for calling primal heuristic <rounding> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	1
heuristics/rounding/freqofs	frequency offset for calling primal heuristic <rounding> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/rounding/maxdepth	maximal depth level to call primal heuristic <rounding> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/rounding/oncepernode	should the heuristic only be called once per node? Range: boolean	0
heuristics/rounding/priority	priority of heuristic <rounding> Range: {-536870912, ..., 536870911}	-1000
heuristics/rounding/successfactor	number of calls per found solution that are considered as standard success, a higher factor causes the heuristic to be called more often Range: {-1, ..., ∞}	100

5.34.4.158 heuristics/shiftandpropagate

Option	Description	Default
heuristics/shiftandpropagate/freq	frequency for calling primal heuristic <shiftandpropagate> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/shiftandpropagate/freqofs	frequency offset for calling primal heuristic <shiftandpropagate> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/shiftandpropagate/binlocksfirst	should binary variables with no locks be preferred in the ordering? Range: boolean	0
heuristics/shiftandpropagate/collectstats	should variable statistics be collected during probing? Range: boolean	1
heuristics/shiftandpropagate/cutoffbreaker	The number of cutoffs before heuristic stops Range: {-1, ..., 1000000}	15
heuristics/shiftandpropagate/fixbinlocks	should binary variables with no locks in one direction be fixed to that direction? Range: boolean	1
heuristics/shiftandpropagate/impliscontinuous	should implicit integer variables be treated as continuous variables? Range: boolean	1
heuristics/shiftandpropagate/maxcutoffquot	maximum percentage of allowed cutoffs before stopping the heuristic Range: [0, 2]	0
heuristics/shiftandpropagate/maxdepth	maximal depth level to call primal heuristic <shiftandpropagate> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/shiftandpropagate/minfixingratelp	minimum fixing rate over all variables (including continuous) to solve LP Range: [0, 1]	0
heuristics/shiftandpropagate/normalize	should coefficients and left/right hand sides be normalized by max row coeff? Range: boolean	1
heuristics/shiftandpropagate/nozerofixing	should variables with a zero shifting value be delayed instead of being fixed? Range: boolean	0
heuristics/shiftandpropagate/nproprounds	The number of propagation rounds used for each propagation Range: {-1, ..., 1000}	10
heuristics/shiftandpropagate/onlywithoutsol	Should heuristic only be executed if no primal solution was found, yet? Range: boolean	1
heuristics/shiftandpropagate/preferbinaries	Should binary variables be shifted first? Range: boolean	1
heuristics/shiftandpropagate/priority	priority of heuristic <shiftandpropagate> Range: {-536870912, ..., 536870911}	1000
heuristics/shiftandpropagate/probing	Should domains be reduced by probing? Range: boolean	1

Option	Description	Default
heuristics/shiftandpropagate/relax	Should continuous variables be relaxed? Range: boolean	1
heuristics/shiftandpropagate/selectbest	should the heuristic choose the best candidate in every round? (set to FALSE for static order)? Range: boolean	0
heuristics/shiftandpropagate/sortkey	the key for variable sorting: (n)orms down, norms (u)p, (v)iolations down, viola(t)ions up, or (r)andom Range: n, r, t, u, v	v
heuristics/shiftandpropagate/sortvars	Should variables be sorted for the heuristic? Range: boolean	1
heuristics/shiftandpropagate/stopafterfeasible	Should the heuristic stop calculating optimal shift values when no more rows are violated? Range: boolean	1
heuristics/shiftandpropagate/updateweights	should row weight be increased every time the row is violated? Range: boolean	0

5.34.4.159 heuristics/shifting

Option	Description	Default
heuristics/shifting/freq	frequency for calling primal heuristic <shifting> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/shifting/freqofs	frequency offset for calling primal heuristic <shifting> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/shifting/maxdepth	maximal depth level to call primal heuristic <shifting> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/shifting/priority	priority of heuristic <shifting> Range: {-536870912, ..., 536870911}	-5000

5.34.4.160 heuristics/simplerounding

Option	Description	Default
heuristics/simplerounding/freq	frequency for calling primal heuristic <simplerounding> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	1
heuristics/simplerounding/freqofs	frequency offset for calling primal heuristic <simplerounding> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/simplerounding/maxdepth	maximal depth level to call primal heuristic <simplerounding> (-1: no limit) Range: {-1, ..., 65534}	-1

Option	Description	Default
heuristics/simplerounding/oncepernode	should the heuristic only be called once per node? Range: boolean	0
heuristics/simplerounding/priority	priority of heuristic <simplerounding> Range: {-536870912, ..., 536870911}	-30

5.34.4.161 heuristics/subnlp

Option	Description	Default
heuristics/subnlp/expectinfeas	percentage of NLP solves with infeasible status required to tell NLP solver to expect an infeasible NLP Range: [0, 1]	0
heuristics/subnlp/feastolfactor	factor on SCIP feasibility tolerance for NLP solves if resolving when NLP solution not feasible in CIP Range: [0, 1]	0.1
heuristics/subnlp/forbidfixings	whether to add constraints that forbid specific fixings that turned out to be infeasible Range: boolean	0
heuristics/subnlp/freq	frequency for calling primal heuristic <subnlp> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	1
heuristics/subnlp/freqofs	frequency offset for calling primal heuristic <subnlp> Range: {0, ..., 65534}	0
heuristics/subnlp/iterinit	number of iterations used for initial NLP solves Range: {0, ..., ∞}	300
heuristics/subnlp/itermin	minimal number of iterations for NLP solves Range: {0, ..., ∞}	20
heuristics/subnlp/maxpresolverounds	limit on number of presolve rounds in sub-SCIP (-1 for unlimited, 0 for no presolve) Range: {-1, ..., ∞}	-1
heuristics/subnlp/ninitsolves	number of successful NLP solves until switching to iterlimit guess and using success rate Range: {0, ..., ∞}	2
heuristics/subnlp/nlpverblevel	verbosity level of NLP solver Range: {0, ..., 65535}	0
heuristics/subnlp/nodesfactor	factor on number of nodes in SCIP (plus nodesoffset) to compute itercontingent (higher value runs heuristics more frequently) Range: [0, ∞]	0.3
heuristics/subnlp/nodesoffset	number of nodes added to the current number of nodes when computing itercontingent (higher value runs heuristic more often in early search) Range: {0, ..., ∞}	1600
heuristics/subnlp/presolveemphasis	presolve emphasis in sub-SCIP (0: default, 1: aggressive, 2: fast, 3: off) Range: {0, ..., 3}	2
heuristics/subnlp/setcutoff	whether to set cutoff in sub-SCIP to current primal bound Range: boolean	1

Option	Description	Default
heuristics/subnlp/successrateexp	exponent for power of success rate to be multiplied with itercontingent (lower value decreases impact of success rate) Range: [0, ∞]	1
Options for expert users		
heuristics/subnlp/keepcopy	whether to keep SCIP copy or to create new copy each time heuristic is applied Range: boolean	1
heuristics/subnlp/maxdepth	maximal depth level to call primal heuristic <subnlp> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/subnlp/opttol	absolute optimality tolerance to use for NLP solves Range: [0, 1]	1e-07
heuristics/subnlp/priority	priority of heuristic <subnlp> Range: {-536870912, ..., 536870911}	-2000010

5.34.4.162 heuristics/trivial

Option	Description	Default
heuristics/trivial/freq	frequency for calling primal heuristic <trivial> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/trivial/freqofs	frequency offset for calling primal heuristic <trivial> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/trivial/maxdepth	maximal depth level to call primal heuristic <trivial> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/trivial/priority	priority of heuristic <trivial> Range: {-536870912, ..., 536870911}	10000

5.34.4.163 heuristics/trivialnegation

Option	Description	Default
heuristics/trivialnegation/freq	frequency for calling primal heuristic <trivialnegation> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/trivialnegation/freqofs	frequency offset for calling primal heuristic <trivialnegation> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/trivialnegation/maxdepth	maximal depth level to call primal heuristic <trivialnegation> (-1: no limit) Range: {-1, ..., 65534}	0
heuristics/trivialnegation/priority	priority of heuristic <trivialnegation> Range: {-536870912, ..., 536870911}	39990

5.34.4.164 heuristics/trustregion

Option	Description	Default
heuristics/trustregion/bestsollimit	limit on number of improving incumbent solutions in sub-CIP Range: {-1, ..., ∞}	3
heuristics/trustregion/freq	frequency for calling primal heuristic <trustregion> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/trustregion/freqofs	frequency offset for calling primal heuristic <trustregion> Range: {0, ..., 65534}	0
heuristics/trustregion/minbinvars	the number of binary variables necessary to run the heuristic Range: {1, ..., ∞}	10
heuristics/trustregion/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	1000
heuristics/trustregion/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.05
heuristics/trustregion/objminimprove	the minimum absolute improvement in the objective function value Range: [0, ∞]	0.01
heuristics/trustregion/violpenalty	the penalty for each change in the binary variables from the candidate solution Range: [0, ∞]	100
Options for expert users		
heuristics/trustregion/copycuts	if uselprows == FALSE, should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/trustregion/lplimfac	factor by which the limit on the number of LP depends on the node limit Range: [1, ∞]	1.5
heuristics/trustregion/maxdepth	maximal depth level to call primal heuristic <trustregion> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/trustregion/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	10000
heuristics/trustregion/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	100
heuristics/trustregion/nwaitingnodes	number of nodes without incumbent change that heuristic should wait Range: {0, ..., ∞}	1
heuristics/trustregion/priority	priority of heuristic <trustregion> Range: {-536870912, ..., 536870911}	-1102010
heuristics/trustregion/uselprows	should subproblem be created out of the rows in the LP rows? Range: boolean	0

5.34.4.165 heuristics/trysol

Option	Description	Default
heuristics/trysol/freq	frequency for calling primal heuristic <trysol> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	1
heuristics/trysol/freqofs	frequency offset for calling primal heuristic <trysol> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/trysol/maxdepth	maximal depth level to call primal heuristic <trysol> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/trysol/priority	priority of heuristic <trysol> Range: {-536870912, ..., 536870911}	-3000010

5.34.4.166 heuristics/twoopt

Option	Description	Default
heuristics/twoopt/freq	frequency for calling primal heuristic <twoopt> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/twoopt/freqofs	frequency offset for calling primal heuristic <twoopt> Range: {0, ..., 65534}	0
Options for expert users		
heuristics/twoopt/intopt	Should Integer-2-Optimization be applied or not? Range: boolean	0
heuristics/twoopt/matchingrate	parameter to determine the percentage of rows two variables have to share before they are considered equal Range: [0, 1]	0.5
heuristics/twoopt/maxdepth	maximal depth level to call primal heuristic <twoopt> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/twoopt/maxnslaves	maximum number of slaves for one master variable Range: {-1, ..., 1000000}	199
heuristics/twoopt/priority	priority of heuristic <twoopt> Range: {-536870912, ..., 536870911}	-20100
heuristics/twoopt/waitingnodes	user parameter to determine number of nodes to wait after last best solution before calling heuristic Range: {0, ..., 10000}	0

5.34.4.167 heuristics/undercover

Option	Description	Default
heuristics/undercover/fixingalts	prioritized sequence of fixing values used ('l'p relaxation, 'n'lp relaxation, 'i'ncumbent solution) Range: string	li

Option	Description	Default
heuristics/undercover/freq	frequency for calling primal heuristic <undercover> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/undercover/freqofs	frequency offset for calling primal heuristic <undercover> Range: {0, ..., 65534}	0
heuristics/undercover/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/undercover/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
heuristics/undercover/onlyconvexify	should we only fix variables in order to obtain a convex problem? Range: boolean	0
heuristics/undercover/postnlp	should the NLP heuristic be called to polish a feasible solution? Range: boolean	1
Options for expert users		
heuristics/undercover/beforecuts	should the heuristic be called at root node before cut separation? Range: boolean	1
heuristics/undercover/conflictweight	weight for conflict score in fixing order Range: real	1000
heuristics/undercover/copycuts	should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/undercover/coverbd	should bounddisjunction constraints be covered (or just copied)? Range: boolean	0
heuristics/undercover/coveringobj	objective function of the covering problem (influenced nonlinear 'c'onstraints/'t'erms, 'd'omain size, 'l'ocks, 'm'in of up/down locks, 'u'nit penalties) Range: c, d, l, m, t, u	u
heuristics/undercover/cutoffweight	weight for cutoff score in fixing order Range: [0, ∞]	1
heuristics/undercover/fixingorder	order in which variables should be fixed (increasing 'C'onflict score, decreasing 'c'onflict score, increasing 'V'ariable index, decreasing 'v'ariable index Range: C, c, V, v	v
heuristics/undercover/fixintfirst	should integer variables in the cover be fixed first? Range: boolean	0
heuristics/undercover/inferenceweight	weight for inference score in fixing order Range: real	1
heuristics/undercover/locksrounding	shall LP values for integer vars be rounded according to locks? Range: boolean	1

Option	Description	Default
heuristics/undercover/maxbacktracks	maximum number of backtracks in fix-and-propagate Range: {0, ..., ∞}	6
heuristics/undercover/maxcoversizeconss	maximum coversize (as ratio to the percentage of non-affected constraints) Range: [0, ∞]	∞
heuristics/undercover/maxcoversizevars	maximum coversize (as fraction of total number of variables) Range: [0, 1]	1
heuristics/undercover/maxdepth	maximal depth level to call primal heuristic <undercover> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/undercover/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	500
heuristics/undercover/maxrecovers	maximum number of recoverings Range: {0, ..., ∞}	0
heuristics/undercover/maxreorders	maximum number of reorderings of the fixing order Range: {0, ..., ∞}	1
heuristics/undercover/mincoveredabs	minimum number of nonlinear constraints in the original problem Range: {0, ..., ∞}	5
heuristics/undercover/mincoveredrel	minimum percentage of nonlinear constraints in the original problem Range: [0, 1]	0.15
heuristics/undercover/minimprove	factor by which the heuristic should at least improve the incumbent Range: [-1, 1]	0
heuristics/undercover/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	500
heuristics/undercover/priority	priority of heuristic <undercover> Range: {-536870912, ..., 536870911}	-1110000
heuristics/undercover/recoverdiv	fraction of covering variables in the last cover which need to change their value when recovering Range: [0, 1]	0.9
heuristics/undercover/reusecover	shall the cover be reused if a conflict was added after an infeasible subproblem? Range: boolean	0

5.34.4.168 heuristics/vbounds

Option	Description	Default
heuristics/vbounds/freq	frequency for calling primal heuristic <vbounds> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	0
heuristics/vbounds/freqofs	frequency offset for calling primal heuristic <vbounds> Range: {0, ..., 65534}	0

Option	Description	Default
heuristics/vbounds/minintfixingrate	minimum percentage of integer variables that have to be fixed Range: [0, 1]	0.65
heuristics/vbounds/minmipfixingrate	minimum percentage of variables that have to be fixed within sub-SCIP (integer and continuous) Range: [0, 1]	0.65
heuristics/vbounds/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	500
heuristics/vbounds/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
Options for expert users		
heuristics/vbounds/copycuts	should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
heuristics/vbounds/feasvariant	which variants of the vbounds heuristic that try to stay feasible should be called? (0: off, 1: w/o looking at obj, 2: only fix to best bound, 4: only fix to worst bound Range: {0, ..., 7}	6
heuristics/vbounds/maxbacktracks	maximum number of backtracks during the fixing process Range: {-1, ..., 536870911}	10
heuristics/vbounds/maxdepth	maximal depth level to call primal heuristic <vbounds> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/vbounds/maxnodes	maximum number of nodes to regard in the sub-problem Range: {0, ..., ∞}	5000
heuristics/vbounds/maxproprounds	maximum number of propagation rounds during probing (-1 infinity) Range: {-1, ..., 536870911}	2
heuristics/vbounds/minimprove	factor by which vbounds heuristic should at least improve the incumbent Range: [0, 1]	0.01
heuristics/vbounds/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	500
heuristics/vbounds/priority	priority of heuristic <vbounds> Range: {-536870912, ..., 536870911}	2500
heuristics/vbounds/tightenvariant	which tightening variants of the vbounds heuristic should be called? (0: off, 1: w/o looking at obj, 2: only fix to best bound, 4: only fix to worst bound Range: {0, ..., 7}	7
heuristics/vbounds/uselockfixings	should more variables be fixed based on variable locks if the fixing rate was not reached? Range: boolean	0

5.34.4.169 heuristics/veclendiving

Option	Description	Default
heuristics/veclendiving/backtrack	use one level of backtracking if infeasibility is encountered? Range: boolean	1
heuristics/veclendiving/freq	frequency for calling primal heuristic <veclendiving> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	10
heuristics/veclendiving/freqofs	frequency offset for calling primal heuristic <veclendiving> Range: {0, ..., 65534}	4
heuristics/veclendiving/lpresolvedomchgquot	percentage of immediate domain changes during probing to trigger LP resolve Range: [0, ∞]	0.15
heuristics/veclendiving/lpsolvefreq	LP solve frequency for diving heuristics (0: only after enough domain changes have been found) Range: {0, ..., ∞}	0
heuristics/veclendiving/maxlpiterofs	additional number of allowed LP iterations Range: {0, ..., ∞}	1000
heuristics/veclendiving/maxlpiterquot	maximal fraction of diving LP iterations compared to node LP iterations Range: [0, ∞]	0.05
heuristics/veclendiving/onlylpbranchcands	should only LP branching candidates be considered instead of the slower but more general constraint handler diving variable selection? Range: boolean	0
Options for expert users		
heuristics/veclendiving/maxdepth	maximal depth level to call primal heuristic <veclendiving> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/veclendiving/maxdiveavgquot	maximal quotient (curlowerbound - lowerbound)/(avglowerbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, ∞]	0
heuristics/veclendiving/maxdiveavgquotnosol	maximal AVGQUOT when no solution was found yet (0.0: no limit) Range: [0, ∞]	0
heuristics/veclendiving/maxdiveubquot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where diving is performed (0.0: no limit) Range: [0, 1]	0.8
heuristics/veclendiving/maxdiveubquotnosol	maximal UBQUOT when no solution was found yet (0.0: no limit) Range: [0, 1]	0.1
heuristics/veclendiving/maxreldepth	maximal relative depth to start diving Range: [0, 1]	1
heuristics/veclendiving/minreldepth	minimal relative depth to start diving Range: [0, 1]	0
heuristics/veclendiving/priority	priority of heuristic <veclendiving> Range: {-536870912, ..., 536870911}	-1003100

5.34.4.170 heuristics/zeroobj

Option	Description	Default
heuristics/zeroobj/freq	frequency for calling primal heuristic <zeroobj> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	-1
heuristics/zeroobj/freqofs	frequency offset for calling primal heuristic <zeroobj> Range: {0, ..., 65534}	0
heuristics/zeroobj/nodesofs	number of nodes added to the contingent of the total nodes Range: {0, ..., ∞}	100
heuristics/zeroobj/nodesquot	contingent of sub problem nodes in relation to the number of nodes of the original problem Range: [0, 1]	0.1
Options for expert users		
heuristics/zeroobj/addallsols	should all subproblem solutions be added to the original SCIP? Range: boolean	0
heuristics/zeroobj/maxdepth	maximal depth level to call primal heuristic <zeroobj> (-1: no limit) Range: {-1, ..., 65534}	0
heuristics/zeroobj/maxlpiters	maximum number of LP iterations to be performed in the subproblem Range: {-1, ..., ∞}	5000
heuristics/zeroobj/maxnodes	maximum number of nodes to regard in the subproblem Range: {0, ..., ∞}	1000
heuristics/zeroobj/minimprove	factor by which zeroobj should at least improve the incumbent Range: [0, 1]	0.01
heuristics/zeroobj/minnodes	minimum number of nodes required to start the subproblem Range: {0, ..., ∞}	100
heuristics/zeroobj/onlywithoutsol	should heuristic only be executed if no primal solution was found, yet? Range: boolean	1
heuristics/zeroobj/priority	priority of heuristic <zeroobj> Range: {-536870912, ..., 536870911}	100
heuristics/zeroobj/useuct	should uct node selection be used at the beginning of the search? Range: boolean	0

5.34.4.171 heuristics/zirounding

Option	Description	Default
heuristics/zirounding/freq	frequency for calling primal heuristic <zirounding> (-1: never, 0: only at depth freqofs) Range: {-1, ..., 65534}	1
heuristics/zirounding/freqofs	frequency offset for calling primal heuristic <zirounding> Range: {0, ..., 65534}	0

Option	Description	Default
Options for expert users		
heuristics/zirounding/maxdepth	maximal depth level to call primal heuristic <zirounding> (-1: no limit) Range: {-1, ..., 65534}	-1
heuristics/zirounding/maxroundingloops	determines maximum number of rounding loops Range: {-1, ..., ∞}	2
heuristics/zirounding/minstopncalls	determines the minimum number of calls before percentage-based deactivation of Zirounding is applied Range: {1, ..., ∞}	1000
heuristics/zirounding/priority	priority of heuristic <zirounding> Range: {-536870912, ..., 536870911}	-500
heuristics/zirounding/stoppercentage	if percentage of found solutions falls below this parameter, Zirounding will be deactivated Range: [0, 1]	0.02
heuristics/zirounding/stopziround	flag to determine if Zirounding is deactivated after a certain percentage of unsuccessful calls Range: boolean	1

5.34.4.172 history

Option	Description	Default
history/allowmerge	should variable histories be merged from sub-SCIPs whenever possible? Range: boolean	0
history/allowtransfer	should variable histories be transferred to initialize SCIP copies? Range: boolean	0
history/valuebased	should statistics be collected for variable domain value pairs? Range: boolean	0

5.34.4.173 limits

Option	Description	Default
limits/absgap	solving stops, if the absolute gap = |primalbound - dualbound| is below the given value Range: [0, ∞]	GAMS optca
limits/autorestartnodes	if solve exceeds this number of nodes for the first time, an automatic restart is triggered (-1: no automatic restart) Range: {-1, ..., ∞}	-1
limits/bestsol	solving stops, if the given number of solution improvements were found (-1: no limit) Range: {-1, ..., ∞}	-1

Option	Description	Default
limits/gap	solving stops, if the relative gap = |primal - dual|/MIN(|dual|,|primal|) is below the given value, the gap is 'Infinity', if primal and dual bound have opposite signs Range: [0, ∞]	GAMS optcr
limits/maxorigsol	maximal number of solutions candidates to store in the solution storage of the original problem Range: {0, ..., ∞}	10
limits/maxsol	maximal number of solutions to store in the solution storage Range: {1, ..., ∞}	100
limits/memory	maximal memory usage in MB; reported memory usage is lower than real memory usage! Range: [0, 8.79609e+12]	GAMS workspace
limits/nodes	maximal number of nodes to process (-1: no limit) Range: {-1, ..., ∞}	GAMS nodlim, if > 0, otherwise -1
limits/restarts	solving stops, if the given number of restarts was triggered (-1: no limit) Range: {-1, ..., ∞}	-1
limits/softtime	soft time limit which should be applied after first solution was found (-1.0: disabled) Range: [-1, ∞]	-1
limits/solutions	solving stops, if the given number of solutions were found; this limit is first checked in presolving (-1: no limit) Range: {-1, ..., ∞}	-1
limits/stallnodes	solving stops, if the given number of nodes was processed since the last improvement of the primal solution value (-1: no limit) Range: {-1, ..., ∞}	-1
limits/time	maximal time in seconds to run Range: [0, ∞]	GAMS reslim
limits/totalnodes	maximal number of total nodes (incl. restarts) to process (-1: no limit) Range: {-1, ..., ∞}	-1

5.34.4.174 lp

Option	Description	Default
lp/alwaysgetduals	should the Farkas duals always be collected when an LP is found to be infeasible? Range: boolean	0

Option	Description	Default
lp/initalgorithm	LP algorithm for solving initial LP relaxations (automatic 's'implex, 'p'rimal simplex, 'd'ual simplex, 'b'arrier, barrier with 'c'rossover) Range: s, p, d, b, c	s
lp/pricing	LP pricing strategy ('l'pi default, 'a'uto, 'f'ull pricing, 'p'artial, 's'teepest edge pricing, 'q'uickstart steepest edge pricing, 'd'evex pricing) Range: l, a, f, p, s, q, d	l
lp/resolvealgorithm	LP algorithm for resolving LP relaxations if a starting basis exists (automatic 's'implex, 'p'rimal simplex, 'd'ual simplex, 'b'arrier, barrier with 'c'rossover) Range: s, p, d, b, c	s
lp/solvedepth	maximal depth for solving LP at the nodes (-1: no depth limit) Range: {-1, ..., 65534}	-1
lp/solvefreq	frequency for solving LP at the nodes (-1: never; 0: only root LP) Range: {-1, ..., 65534}	1
lp/solver	LP solver to use (clp, cplex, soplex) Range: string	cplex, if licensed, otherwise soplex
Options for expert users		
lp/checkdualfeas	should LP solutions be checked for dual feasibility, resolving LP when numerical troubles occur? Range: boolean	1
lp/checkfarkas	should infeasibility proofs from the LP be checked? Range: boolean	1
lp/checkprimfeas	should LP solutions be checked for primal feasibility, resolving LP when numerical troubles occur? Range: boolean	1
lp/checkstability	should LP solver's return status be checked for stability? Range: boolean	1
lp/cleanupcols	should new non-basic columns be removed after LP solving? Range: boolean	0
lp/cleanupcolsroot	should new non-basic columns be removed after root LP solving? Range: boolean	0
lp/cleanuprows	should new basic rows be removed after LP solving? Range: boolean	1
lp/cleanuprowsroot	should new basic rows be removed after root LP solving? Range: boolean	1

Option	Description	Default
lp/clearinitialprobinglp	should lp state be cleared at the end of probing mode when lp was initially unsolved, e.g., when called right after presolving? Range: boolean	1
lp/colagelimit	maximum age a dynamic column can reach before it is deleted from the LP (-1: don't delete columns due to aging) Range: {-1, ..., ∞}	10
lp/conditionlimit	maximum condition number of LP basis counted as stable (-1.0: no limit) Range: [-1, ∞]	-1
lp/disablecutoff	disable the cutoff bound in the LP solver? (0: enabled, 1: disabled, 2: auto) Range: {0, ..., 2}	2
lp/fastmip	which FASTMIP setting of LP solver should be used? 0: off, 1: low Range: {0, ..., 1}	1
lp/freesolvalbuffers	should the buffers for storing LP solution values during diving be freed at end of diving? Range: boolean	0
lp/iterlim	iteration limit for each single LP solve (-1: no limit) Range: {-1, ..., ∞}	-1
lp/lexdualalgo	should the lexicographic dual algorithm be used? Range: boolean	0
lp/lexdualbasic	choose fractional basic variables in lexicographic dual algorithm? Range: boolean	0
lp/lexdualmaxrounds	maximum number of rounds in the lexicographic dual algorithm (-1: unbounded) Range: {-1, ..., ∞}	2
lp/lexdualrootonly	should the lexicographic dual algorithm be applied only at the root node Range: boolean	1
lp/lexdualstalling	turn on the lex dual algorithm only when stalling? Range: boolean	1
lp/minmarkowitz	minimal Markowitz threshold to control sparsity/stability in LU factorization Range: [0.0001, 0.9999]	0.01
lp/presolving	should presolving of LP solver be used? Range: boolean	1
lp/refactorinterval	LP refactorization interval (0: auto) Range: {0, ..., ∞}	0

Option	Description	Default
lp/resolveiterfac	factor of average LP iterations that is used as LP iteration limit for LP resolve (-1: unlimited) Range: [-1, ∞]	-1
lp/resolveitermin	minimum number of iterations that are allowed for LP resolve Range: {1, ..., ∞}	1000
lp/resolverestore	should the LP be resolved to restore the state at start of diving (if FALSE we buffer the solution values)? Range: boolean	0
lp/rootiterlim	iteration limit for initial root LP solve (-1: no limit) Range: {-1, ..., ∞}	-1
lp/rowagelimit	maximum age a dynamic row can reach before it is deleted from the LP (-1: don't delete rows due to aging) Range: {-1, ..., ∞}	10
lp/rowrepswitch	simplex algorithm shall use row representation of the basis if number of rows divided by number of columns exceeds this value (-1.0 to disable row representation) Range: [-1, ∞]	1.2
lp/scaling	LP scaling (0: none, 1: normal, 2: aggressive) Range: {0, ..., 2}	1
lp/solutionpolishing	LP solution polishing method (0: disabled, 1: only root, 2: always, 3: auto) Range: {0, ..., 3}	3
lp/threads	number of threads used for solving the LP (0: automatic) Range: {0, ..., 64}	GAMS threads

5.34.4.175 memory

Option	Description	Default
memory/savefac	fraction of maximal memory usage resulting in switch to memory saving mode Range: [0, 1]	0.8
Options for expert users		
memory/arraygrowfac	memory growing factor for dynamically allocated arrays Range: [1, 10]	1.2
memory/arraygrowinit	initial size of dynamically allocated arrays Range: {0, ..., ∞}	4
memory/pathgrowfac	memory growing factor for path array Range: [1, 10]	2
memory/pathgrowinit	initial size of path array Range: {0, ..., ∞}	256

Option	Description	Default
memory/treegrowfac	memory growing factor for tree array Range: [1, 10]	2
memory/treegrowinit	initial size of tree array Range: {0, ..., ∞}	65536

5.34.4.176 misc

Option	Description	Default
misc/allowstrongdualreds	should strong dual reductions be allowed in propagation and presolving? Range: boolean	1
misc/allowweakdualreds	should weak dual reductions be allowed in propagation and presolving? Range: boolean	1
misc/avoidmemout	try to avoid running into memory limit by restricting plugins like heuristics? Range: boolean	1
misc/calcintegral	should SCIP calculate the primal dual integral value? Range: boolean	1
misc/catchctrlc	should the CTRL-C interrupt be caught by SCIP? Range: boolean	1
misc/estimexternmem	should the usage of external memory be estimated? Range: boolean	1
misc/finitesolutionstore	should SCIP try to remove infinite fixings from solutions copied to the solution store? Range: boolean	0
misc/improvingsols	should only solutions be checked which improve the primal bound Range: boolean	0
misc/outputorigsol	should the best solution be transformed to the orignal space and be output in command line run? Range: boolean	1
misc/printreason	should the reason be printed if a given start solution is infeasible Range: boolean	0
misc/referencevalue	objective value for reference purposes Range: real	∞
misc/resetstat	should the statistics be reset if the transformed problem is freed (in case of a Benders' decomposition this parameter should be set to FALSE) Range: boolean	1
misc/scaleobj	should the objective function be scaled so that it is always integer? Range: boolean	1
misc/showdivingstats	should detailed statistics for diving heuristics be shown? Range: boolean	0
misc/transorigsols	should SCIP try to transfer original solutions to the transformed space (after presolving)? Range: boolean	1
misc/transsolsorig	should SCIP try to transfer transformed solutions to the original space (after solving)? Range: boolean	1

Option	Description	Default
misc/useconstable	should a hashtable be used to map from constraint names to constraints? Range: boolean	1
misc/usesmalltables	should smaller hashtables be used? yields better performance for small problems with about 100 variables Range: boolean	0
misc/usesymmetry	bitset describing used symmetry handling technique (0: off; 1: polyhedral (orbitopes and/or symresacks); 2: orbital fixing; 3: orbitopes and orbital fixing; 4: Schreier Sims cuts; 5: Schreier Sims cuts and orbitopes); 6: Schreier Sims cuts and orbital fixing; 7: Schreier Sims cuts, orbitopes, and orbital fixing Range: {0, ..., 7}	7
misc/usevartable	should a hashtable be used to map from variable names to variables? Range: boolean	1

5.34.4.177 nlhdlr/bilinear

Option	Description	Default
nlhdlr/bilinear/enabled	should this nonlinear handler be used Range: boolean	1
nlhdlr/bilinear/maxsepadepth	maximum depth to apply separation Range: {0, ..., ∞}	∞
nlhdlr/bilinear/maxseparounds	maximum number of separation rounds in a local node Range: {0, ..., ∞}	1
nlhdlr/bilinear/maxseparoundsroot	maximum number of separation rounds in the root node Range: {0, ..., ∞}	10
nlhdlr/bilinear/useinteval	whether to use the interval evaluation callback of the nlhdlr Range: boolean	1
nlhdlr/bilinear/usereverseprop	whether to use the reverse propagation callback of the nlhdlr Range: boolean	1

5.34.4.178 nlhdlr/concave

Option	Description	Default
nlhdlr/concave/detectsum	whether to run convexity detection when the root of an expression is a sum Range: boolean	0
nlhdlr/concave/enabled	should this nonlinear handler be used Range: boolean	1
Options for expert users		
nlhdlr/concave/cvxprodcomp	whether to use convexity check on product composition f(h)$*$h Range: boolean	1
nlhdlr/concave/cvxquadratic	whether to use convexity check on quadratics Range: boolean	0

Option	Description	Default
nlhdlr/concave/cvxsignomial	whether to use convexity check on signomials Range: boolean	1
nlhdlr/concave/handletrivial	whether to also handle trivial convex expressions Range: boolean	0

5.34.4.179 nlhdlr/convex

Option	Description	Default
nlhdlr/convex/detectsum	whether to run convexity detection when the root of an expression is a non-quadratic sum Range: boolean	0
nlhdlr/convex/enabled	should this nonlinear handler be used Range: boolean	1
nlhdlr/convex/extendedform	whether to create extended formulations instead of looking for maximal convex expressions Range: boolean	1
Options for expert users		
nlhdlr/convex/cvxprodcomp	whether to use convexity check on product composition f(h)∗h Range: boolean	1
nlhdlr/convex/cvxquadratic	whether to use convexity check on quadratics Range: boolean	1
nlhdlr/convex/cvxsignomial	whether to use convexity check on signomials Range: boolean	1
nlhdlr/convex/handletrivial	whether to also handle trivial convex expressions Range: boolean	0

5.34.4.180 nlhdlr/default

Option	Description	Default
nlhdlr/default/enabled	should this nonlinear handler be used Range: boolean	1

5.34.4.181 nlhdlr/perspective

Option	Description	Default
nlhdlr/perspective/adjrefpoint	whether to adjust the reference point Range: boolean	1
nlhdlr/perspective/convexonly	whether perspective cuts are added only for convex expressions Range: boolean	0
nlhdlr/perspective/enabled	should this nonlinear handler be used Range: boolean	1
nlhdlr/perspective/maxproprounds	maximal number of propagation rounds in probing Range: {-1, ..., ∞}	1
nlhdlr/perspective/mindomreduction	minimal relative reduction in a variable's domain for applying probing Range: [0, 1]	0.1

Option	Description	Default
nlhdlr/perspective/minviolprobing	minimal violation w.r.t. auxiliary variables for applying probing Range: [0, ∞]	1e-05
nlhdlr/perspective/probingfreq	probing frequency (-1 - no probing, 0 - root node only) Range: {-1, ..., ∞}	1
nlhdlr/perspective/probingonlyinsepa	whether to do probing only in separation Range: boolean	1
nlhdlr/perspective/tightenbounds	whether variable semicontinuity is used to tighten variable bounds Range: boolean	1

5.34.4.182 nlhdlr/quadratic

Option	Description	Default
nlhdlr/quadratic/atwhichnodes	determines at which nodes cut is used (if it's -1, it's used only at the root node, if it's n ≥ 0, it's used at every multiple of n Range: {-1, ..., ∞}	1
nlhdlr/quadratic/enabled	should this nonlinear handler be used Range: boolean	1
nlhdlr/quadratic/ignorebadrayrestriction	should cut be generated even with bad numerics when restricting to ray? Range: boolean	1
nlhdlr/quadratic/ignorenhighre	should cut be added even when range / efficacy is large? Range: boolean	1
nlhdlr/quadratic/maxrank	maximal rank a slackvar can have Range: {0, ..., ∞}	∞
nlhdlr/quadratic/mincutviolation	minimal cut violation the generated cuts must fulfill to be added to the LP Range: [0, ∞]	0.0001
nlhdlr/quadratic/minviolation	minimal violation the constraint must fulfill such that a cut is generated Range: [0, ∞]	0.0001
nlhdlr/quadratic/ncutslimit	limit for number of cuts generated consecutively Range: {0, ..., ∞}	2
nlhdlr/quadratic/ncutslimitroot	limit for number of cuts generated at root node Range: {0, ..., ∞}	20
nlhdlr/quadratic/nstrengthlimit	limit for number of rays we do the strengthening for Range: {0, ..., ∞}	∞
nlhdlr/quadratic/useboundsasrays	use bounds of variables in quadratic as rays for intersection cuts Range: boolean	0
nlhdlr/quadratic/useintersectioncuts	whether to use intersection cuts for quadratic constraints to separate Range: boolean	0
nlhdlr/quadratic/usestrengthening	whether the strengthening should be used Range: boolean	0

5.34.4.183 nlhdlr/quotient

Option	Description	Default
nlhdlr/quotient/enabled	should this nonlinear handler be used Range: boolean	1

5.34.4.184 nlhdlr/soc

Option	Description	Default
nlhdlr/soc/compeigenvalues	Should Eigenvalue computations be done to detect complex cases in quadratic constraints? Range: boolean	1
nlhdlr/soc/enabled	should this nonlinear handler be used Range: boolean	1
nlhdlr/soc/mincutefficacy	Minimum efficacy which a cut needs in order to be added. Range: [0, ∞]	1e-05

5.34.4.185 nlp

Option	Description	Default
nlp/disable	should the NLP relaxation be always disabled (also for NLPs/MINLPs)? Range: boolean	0

5.34.4.186 nlpi/ipopt

Option	Description	Default
nlpi/ipopt/hessian_approximation	Indicates what Hessian information is to be used. Valid values if not empty: exact limited-memory Range: string	
nlpi/ipopt/hsllib	Name of library containing HSL routines for load at runtime Range: string	
nlpi/ipopt/linear_solver	Linear solver used for step computations. Valid values if not empty: ma27 ma57 ma77 ma86 ma97 pardiso pardisomkl mumps custom Range: string	ma27, if IpoptH licensed, mumps otherwise
nlpi/ipopt/linear_system_scaling	Method for scaling the linear system. Valid values if not empty: none mc19 slack-based Range: string	mc19, if IpoptH licensed, empty otherwise
nlpi/ipopt/mu_strategy	Update strategy for barrier parameter. Valid values if not empty: monotone adaptive Range: string	

Option	Description	Default
nlpi/ipopt/nlp_scaling_method	Select the technique used for scaling the NLP. Valid values if not empty: none user-scaling gradient-based equilibration-based Range: string	
nlpi/ipopt/optfile	name of Ipopt options file Range: string	
nlpi/ipopt/pardisolib	Name of library containing Pardiso routines (from pardiso-project.org) for load at runtime Range: string	
nlpi/ipopt/print_level	Output verbosity level. -1 to use NLPI or Ipopt default. Range: {-1, ..., 12}	-1
nlpi/ipopt/priority	priority of NLPI <ipopt> Range: {-536870912, ..., 536870911}	1000
nlpi/ipopt/warm_start_push	amount (relative and absolute) by which starting point is moved away from bounds in warmstarts Range: [0, 1]	1e-09

5.34.4.187 nodeselection

Option	Description	Default
nodeselection/childsel	child selection rule ('d'own, 'u'p, 'p'seudo costs, 'i'nference, 'l'p value, 'r'oot LP value difference, 'h'ybrid inference/root LP value difference) Range: d, u, p, i, l, r, h	h

5.34.4.188 nodeselection/bfs

Option	Description	Default
nodeselection/bfs/stdpriority	priority of node selection rule <bfs> in standard mode Range: {-536870912, ..., 1073741823}	100000
Options for expert users		
nodeselection/bfs/maxplungedepth	maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting) Range: {-1, ..., ∞}	-1
nodeselection/bfs/maxplungequot	maximal quotient (curlowerbound - lowerbound)/(cutoffbound - lowerbound) where plunging is performed Range: [0, ∞]	0.25
nodeselection/bfs/memsavepriority	priority of node selection rule <bfs> in memory saving mode Range: {-536870912, ..., 536870911}	0
nodeselection/bfs/minplungedepth	minimal plunging depth, before new best node may be selected (-1 for dynamic setting) Range: {-1, ..., ∞}	-1

5.34.4.189 nodeselection/breadthfirst

Option	Description	Default
nodeselection/breadthfirst/stdpriority	priority of node selection rule <breadthfirst> in standard mode Range: {-536870912, ..., 1073741823}	-10000
Options for expert users		
nodeselection/breadthfirst/memsavepriority	priority of node selection rule <breadthfirst> in memory saving mode Range: {-536870912, ..., 536870911}	-1000000

5.34.4.190 nodeselection/dfs

Option	Description	Default
nodeselection/dfs/stdpriority	priority of node selection rule <dfs> in standard mode Range: {-536870912, ..., 1073741823}	0
Options for expert users		
nodeselection/dfs/memsavepriority	priority of node selection rule <dfs> in memory saving mode Range: {-536870912, ..., 536870911}	100000

5.34.4.191 nodeselection/estimate

Option	Description	Default
nodeselection/estimate/bestnodefreq	frequency at which the best node instead of the best estimate is selected (0: never) Range: {0, ..., ∞}	10
nodeselection/estimate/breadthfirstdepth	depth until breadth-first search is applied Range: {-1, ..., ∞}	-1
nodeselection/estimate/plungeoffset	number of nodes before doing plunging the first time Range: {0, ..., ∞}	0
nodeselection/estimate/stdpriority	priority of node selection rule <estimate> in standard mode Range: {-536870912, ..., 1073741823}	200000
Options for expert users		
nodeselection/estimate/maxplungedepth	maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting) Range: {-1, ..., ∞}	-1
nodeselection/estimate/maxplungequot	maximal quotient (estimate - lowerbound)/(cutoffbound - lowerbound) where plunging is performed Range: [0, ∞]	0.25
nodeselection/estimate/memsavepriority	priority of node selection rule <estimate> in memory saving mode Range: {-536870912, ..., 536870911}	100

Option	Description	Default
nodeselection/estimate/minplungedepth	minimal plunging depth, before new best node may be selected (-1 for dynamic setting) Range: {-1, ..., ∞}	-1

5.34.4.192 nodeselection/hybridestim

Option	Description	Default
nodeselection/hybridestim/bestnodefreq	frequency at which the best node instead of the hybrid best estimate / best bound is selected (0: never) Range: {0, ..., ∞}	1000
nodeselection/hybridestim/stdpriority	priority of node selection rule <hybridestim> in standard mode Range: {-536870912, ..., 1073741823}	50000
Options for expert users		
nodeselection/hybridestim/estimweight	weight of estimate value in node selection score (0: pure best bound search, 1: pure best estimate search) Range: [0, 1]	0.1
nodeselection/hybridestim/maxplungedepth	maximal plunging depth, before new best node is forced to be selected (-1 for dynamic setting) Range: {-1, ..., ∞}	-1
nodeselection/hybridestim/maxplungequot	maximal quotient (estimate - lowerbound)/(cutoffbound - lowerbound) where plunging is performed Range: [0, ∞]	0.25
nodeselection/hybridestim/memsavepriority	priority of node selection rule <hybridestim> in memory saving mode Range: {-536870912, ..., 536870911}	50
nodeselection/hybridestim/minplungedepth	minimal plunging depth, before new best node may be selected (-1 for dynamic setting) Range: {-1, ..., ∞}	-1

5.34.4.193 nodeselection/restartdfs

Option	Description	Default
nodeselection/restartdfs/countonlyleaves	count only leaf nodes (otherwise all nodes)? Range: boolean	1
nodeselection/restartdfs/selectbestfreq	frequency for selecting the best node instead of the deepest one Range: {0, ..., ∞}	100
nodeselection/restartdfs/stdpriority	priority of node selection rule <restartdfs> in standard mode Range: {-536870912, ..., 1073741823}	10000
Options for expert users		

Option	Description	Default
nodeselection/restartdfs/memsavepriority	priority of node selection rule <restartdfs> in memory saving mode Range: {-536870912, ..., 536870911}	50000

5.34.4.194 nodeselection/uct

Option	Description	Default
nodeselection/uct/stdpriority	priority of node selection rule <uct> in standard mode Range: {-536870912, ..., 1073741823}	10
Options for expert users		
nodeselection/uct/memsavepriority	priority of node selection rule <uct> in memory saving mode Range: {-536870912, ..., 536870911}	0
nodeselection/uct/nodelimit	maximum number of nodes before switching to default rule Range: {0, ..., 1000000}	31
nodeselection/uct/useestimate	should the estimate (TRUE) or lower bound of a node be used for UCT score? Range: boolean	0
nodeselection/uct/weight	weight for visit quotient of node selection rule Range: [0, 1]	0.1

5.34.4.195 numerics

Option	Description	Default
numerics/checkfeastolfac	feasibility tolerance factor; for checking the feasibility of the best solution Range: [0, ∞]	1
numerics/dualfeastol	feasibility tolerance for reduced costs in LP solution Range: [1e-17, 0.001]	1e-07
numerics/epsilon	absolute values smaller than this are considered zero Range: [1e-20, 0.001]	1e-09
numerics/feastol	feasibility tolerance for constraints Range: [1e-17, 0.001]	1e-06
numerics/lpfeastolfactor	factor w.r.t. primal feasibility tolerance that determines default (and maximal) primal feasibility tolerance of LP solver Range: [1e-06, 1]	1
numerics/sumepsilon	absolute values of sums smaller than this are considered zero Range: [1e-17, 0.001]	1e-06
Options for expert users		
numerics/barrierconvtol	LP convergence tolerance used in barrier algorithm Range: [1e-17, 0.001]	1e-10
numerics/boundstreps	minimal relative improve for strengthening bounds Range: [1e-17, ∞]	0.05
numerics/hugeval	values larger than this are considered huge and should be handled separately (e.g., in activity computation) Range: [0, ∞]	1e+15

Option	Description	Default
numerics/pseudocostdelta	minimal objective distance value to use for branching pseudo cost updates Range: [0, ∞]	0.0001
numerics/pseudocosteps	minimal variable distance value to use for branching pseudo cost updates Range: [1e-17, 1]	0.1
numerics/recomputefac	minimal decrease factor that causes the recomputation of a value (e.g., pseudo objective) instead of an update Range: [0, ∞]	1e+07

5.34.4.196 presolving

Option	Description	Default
presolving/maxrestarts	maximal number of restarts (-1: unlimited) Range: {-1, ..., ∞}	-1
presolving/maxrounds	maximal number of presolving rounds (-1: unlimited, 0: off) Range: {-1, ..., ∞}	-1
Options for expert users		
presolving/abortfac	abort presolve, if at most this fraction of the problem was changed in last presolve round Range: [0, 1]	0.0008
presolving/clqtablefac	limit on number of entries in clique table relative to number of problem nonzeros Range: [0, ∞]	2
presolving/donotaggr	should aggregation of variables be forbidden? Range: boolean	0
presolving/donotmultaggr	should multi-aggregation of variables be forbidden? Range: boolean	0
presolving/immrestartfac	fraction of integer variables that were fixed in the root node triggering an immediate restart with preprocessing Range: [0, 1]	0.1
presolving/restartfac	fraction of integer variables that were fixed in the root node triggering a restart with preprocessing after root node evaluation Range: [0, 1]	0.025
presolving/restartminred	minimal fraction of integer variables removed after restart to allow for an additional restart Range: [0, 1]	0.1
presolving/subrestartfac	fraction of integer variables that were globally fixed during the solving process triggering a restart with preprocessing Range: [0, 1]	1

5.34.4.197 presolving/boundshift

Option	Description	Default
presolving/boundshift/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	0
Options for expert users		

Option	Description	Default
presolving/boundshift/flipping	is flipping allowed (multiplying with -1)? Range: boolean	1
presolving/boundshift/integer	shift only integer ranges? Range: boolean	1
presolving/boundshift/maxshift	absolute value of maximum shift Range: {0, ..., ∞}	∞
presolving/boundshift/priority	priority of presolver <boundshift> Range: {-536870912, ..., 536870911}	7900000
presolving/boundshift/timing	timing mask of presolver <boundshift> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	4

5.34.4.198 presolving/convertinttobin

Option	Description	Default
presolving/convertinttobin/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	0
Options for expert users		
presolving/convertinttobin/maxdomainsize	absolute value of maximum domain size for converting an integer variable to binaries variables Range: {0, ..., ∞}	∞
presolving/convertinttobin/onlypoweroftwo	should only integer variables with a domain size of 2^p - 1 be converted(, there we don't need an knapsack-constraint for restricting the sum of the binaries) Range: boolean	0
presolving/convertinttobin/priority	priority of presolver <convertinttobin> Range: {-536870912, ..., 536870911}	6000000
presolving/convertinttobin/samelocksinbothdirections	should only integer variables with uplocks equals downlocks be converted Range: boolean	0
presolving/convertinttobin/timing	timing mask of presolver <convertinttobin> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	4

5.34.4.199 presolving/domcol

Option	Description	Default
presolving/domcol/continuousred	should reductions for continuous variables be performed? Range: boolean	1
presolving/domcol/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1
presolving/domcol/nummaxpairs	maximal number of pair comparisons Range: {1024, ..., 1000000000}	1048576

Option	Description	Default
presolving/domcol/numminpairs	minimal number of pair comparisons Range: {100, ..., 1048576}	1024
presolving/domcol/predbndstr	should predictive bound strengthening be applied? Range: boolean	0
Options for expert users		
presolving/domcol/priority	priority of presolver <domcol> Range: {-536870912, ..., 536870911}	-1000
presolving/domcol/timing	timing mask of presolver <domcol> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16

5.34.4.200 presolving/dualagg

Option	Description	Default
presolving/dualagg/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	0
Options for expert users		
presolving/dualagg/priority	priority of presolver <dualagg> Range: {-536870912, ..., 536870911}	-12000
presolving/dualagg/timing	timing mask of presolver <dualagg> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16

5.34.4.201 presolving/dualcomp

Option	Description	Default
presolving/dualcomp/componlydisvars	should only discrete variables be compensated? Range: boolean	0
presolving/dualcomp/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
presolving/dualcomp/priority	priority of presolver <dualcomp> Range: {-536870912, ..., 536870911}	-50
presolving/dualcomp/timing	timing mask of presolver <dualcomp> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16

5.34.4.202 presolving/dualinfer

Option	Description	Default
presolving/dualinfer/maxdualbndloops	maximal number of dual bound strengthening loops Range: {-1, ..., ∞}	12
presolving/dualinfer/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	0

Option	Description	Default
presolving/dualinfer/maxrowsupport	Maximum number of row's non-zeros for changing inequality to equality Range: {2, ..., ∞}	3
presolving/dualinfer/twocolcombine	use convex combination of columns for determining dual bounds Range: boolean	1
Options for expert users		
presolving/dualinfer/maxcombinefails	maximal number of consecutive useless column combines Range: {-1, ..., ∞}	1000
presolving/dualinfer/maxconsiderednonzeros	maximal number of considered non-zeros within one column (-1: no limit) Range: {-1, ..., ∞}	100
presolving/dualinfer/maxhashfac	Maximum number of hashlist entries as multiple of number of columns in the problem (-1: no limit) Range: {-1, ..., ∞}	10
presolving/dualinfer/maxpairfac	Maximum number of processed column pairs as multiple of the number of columns in the problem (-1: no limit) Range: {-1, ..., ∞}	1
presolving/dualinfer/maxretrievefails	maximal number of consecutive useless hashtable retrieves Range: {-1, ..., ∞}	1000
presolving/dualinfer/priority	priority of presolver <dualinfer> Range: {-536870912, ..., 536870911}	-3000
presolving/dualinfer/timing	timing mask of presolver <dualinfer> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16

5.34.4.203 presolving/dualsparsify

Option	Description	Default
presolving/dualsparsify/maxbinfillin	maximal fillin for binary variables (-1: unlimited) Range: {-1, ..., ∞}	1
presolving/dualsparsify/maxcontfillin	maximal fillin for continuous variables (-1: unlimited) Range: {-1, ..., ∞}	1
presolving/dualsparsify/maxintfillin	maximal fillin for integer variables including binaries (-1: unlimited) Range: {-1, ..., ∞}	1
presolving/dualsparsify/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1
presolving/dualsparsify/mineliminatednonzeros	minimal eliminated nonzeros within one column if we need to add a constraint to the problem Range: {0, ..., ∞}	100
Options for expert users		

Option	Description	Default
presolving/dualsparsify/enablecopy	should dualsparsify presolver be copied to sub-SCIPs? Range: boolean	1
presolving/dualsparsify/maxconsiderednonzeros	maximal number of considered nonzeros within one column (-1: no limit) Range: {-1, ..., ∞}	70
presolving/dualsparsify/maxretrievefac	limit on the number of useless vs. useful hashtable retrieves as a multiple of the number of constraints Range: [0, ∞]	100
presolving/dualsparsify/preservegoodlocks	should we preserve good locked properties of variables (at most one lock in one direction)? Range: boolean	0
presolving/dualsparsify/preserveintcoefs	should we forbid cancellations that destroy integer coefficients? Range: boolean	0
presolving/dualsparsify/priority	priority of presolver <dualsparsify> Range: {-536870912, ..., 536870911}	-240000
presolving/dualsparsify/timing	timing mask of presolver <dualsparsify> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16
presolving/dualsparsify/waitingfac	number of calls to wait until next execution as a multiple of the number of useless calls Range: [0, ∞]	2

5.34.4.204 presolving/gateextraction

Option	Description	Default
presolving/gateextraction/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
presolving/gateextraction/onlysetpart	should we only try to extract set-partitioning constraints and no and-constraints Range: boolean	0
presolving/gateextraction/priority	priority of presolver <gateextraction> Range: {-536870912, ..., 536870911}	1000000
presolving/gateextraction/searchequations	should we try to extract set-partitioning constraint out of one logicor and one corresponding set-packing constraint Range: boolean	1
presolving/gateextraction/sorting	order logicor contraints to extract big-gates before smaller ones (-1), do not order them (0) or order them to extract smaller gates at first (1) Range: {-1, ..., 1}	1
presolving/gateextraction/timing	timing mask of presolver <gateextraction> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16

5.34.4.205 presolving/implics

Option	Description	Default
presolving/implics/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
presolving/implics/priority	priority of presolver <implics> Range: {-536870912, ..., 536870911}	-10000
presolving/implics/timing	timing mask of presolver <implics> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	8

5.34.4.206 presolving/inttobinary

Option	Description	Default
presolving/inttobinary/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
presolving/inttobinary/priority	priority of presolver <inttobinary> Range: {-536870912, ..., 536870911}	7000000
presolving/inttobinary/timing	timing mask of presolver <inttobinary> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	4

5.34.4.207 presolving/milp

Option	Description	Default
presolving/milp/hugebound	absolute bound value that is considered too huge for activitity based calculations Range: [0, ∞]	1e+08
presolving/milp/markowitztolerance	the markowitz tolerance used for substitutions Range: [0, 1]	0.01
presolving/milp/maxbadgesizepar	maximal badge size in Probing in PaPILO if PaPILO is executed in parallel mode Range: {-1, ..., ∞}	-1
presolving/milp/maxbadgesizeseq	maximal badge size in Probing in PaPILO if PaPILO is executed in sequential mode Range: {-1, ..., ∞}	15000
presolving/milp/maxfillinpersubstitution	maximal possible fillin for substitutions to be considered Range: {-2147483648, ..., ∞}	3
presolving/milp/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1

Option	Description	Default
presolving/milp/modifyconsfac	modify SCIP constraints when the number of nonzeros or rows is at most this factor times the number of nonzeros or rows before presolving Range: [0, 1]	0.8
presolving/milp/randomseed	the random seed used for randomization of tie breaking Range: {-2147483648, ..., ∞}	0
presolving/milp/threads	maximum number of threads presolving may use (0: automatic) Range: {0, ..., ∞}	GAMS threads
Options for expert users		
presolving/milp/enabledomcol	should the dominated column presolver be enabled within the presolve library? Range: boolean	1
presolving/milp/enabledualinfer	should the dualinfer presolver be enabled within the presolve library? Range: boolean	1
presolving/milp/enablemultiaggr	should the multi-aggregation presolver be enabled within the presolve library? Range: boolean	1
presolving/milp/enableparallelrows	should the parallel rows presolver be enabled within the presolve library? Range: boolean	1
presolving/milp/enableprobing	should the probing presolver be enabled within the presolve library? Range: boolean	1
presolving/milp/enablesparsify	should the sparsify presolver be enabled within the presolve library? Range: boolean	0
presolving/milp/maxshiftperrow	maximal amount of nonzeros allowed to be shifted to make space for substitutions Range: {0, ..., ∞}	10
presolving/milp/priority	priority of presolver <milp> Range: {-536870912, ..., 536870911}	9999999
presolving/milp/probfilename	filename to store the problem before MILP presolving starts Range: string	-
presolving/milp/timing	timing mask of presolver <milp> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	8

5.34.4.208 presolving/qpkktref

Option	Description	Default
presolving/qpkktref/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	0
Options for expert users		

Option	Description	Default
presolving/qpkktref/addkktbinary	if TRUE then allow binary variables for KKT update Range: boolean	0
presolving/qpkktref/priority	priority of presolver <qpkktref> Range: {-536870912, ..., 536870911}	-1
presolving/qpkktref/timing	timing mask of presolver <qpkktref> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	8
presolving/qpkktref/updatequadbounded	if TRUE then only apply the update to QPs with bounded variables; if the variables are not bounded then a finite optimal solution might not exist and the KKT conditions would then be invalid Range: boolean	1
presolving/qpkktref/updatequadindef	if TRUE then apply quadratic constraint update even if the quadratic constraint matrix is known to be indefinite Range: boolean	0

5.34.4.209 presolving/redvub

Option	Description	Default
presolving/redvub/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	0
Options for expert users		
presolving/redvub/priority	priority of presolver <redvub> Range: {-536870912, ..., 536870911}	-9000000
presolving/redvub/timing	timing mask of presolver <redvub> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16

5.34.4.210 presolving/sparsify

Option	Description	Default
presolving/sparsify/maxbinfillin	maximal fillin for binary variables (-1: unlimited) Range: {-1, ..., ∞}	0
presolving/sparsify/maxcontfillin	maximal fillin for continuous variables (-1: unlimited) Range: {-1, ..., ∞}	0
presolving/sparsify/maxintfillin	maximal fillin for integer variables including binaries (-1: unlimited) Range: {-1, ..., ∞}	0
presolving/sparsify/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		

Option	Description	Default
presolving/sparsify/cancellinear	should we cancel nonzeros in constraints of the linear constraint handler? Range: boolean	1
presolving/sparsify/enablecopy	should sparsify presolver be copied to sub-SCIPs? Range: boolean	1
presolving/sparsify/maxconsiderednonzeros	maximal number of considered non-zeros within one row (-1: no limit) Range: {-1, ..., ∞}	70
presolving/sparsify/maxnonzeros	maximal support of one equality to be used for cancelling (-1: no limit) Range: {-1, ..., ∞}	-1
presolving/sparsify/maxretrievefac	limit on the number of useless vs. useful hashtable retrieves as a multiple of the number of constraints Range: [0, ∞]	100
presolving/sparsify/preserveintcoefs	should we forbid cancellations that destroy integer coefficients? Range: boolean	1
presolving/sparsify/priority	priority of presolver <sparsify> Range: {-536870912, ..., 536870911}	-24000
presolving/sparsify/rowsort	order in which to process inequalities ('n'o sorting, 'i'ncreasing nonzeros, 'd'ecreasing nonzeros) Range: n, i, d	d
presolving/sparsify/timing	timing mask of presolver <sparsify> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16
presolving/sparsify/waitingfac	number of calls to wait until next execution as a multiple of the number of useless calls Range: [0, ∞]	2

5.34.4.211 presolving/stuffing

Option	Description	Default
presolving/stuffing/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	0
Options for expert users		
presolving/stuffing/priority	priority of presolver <stuffing> Range: {-536870912, ..., 536870911}	-100
presolving/stuffing/timing	timing mask of presolver <stuffing> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16

5.34.4.212 presolving/trivial

Option	Description	Default
presolving/trivial/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
presolving/trivial/priority	priority of presolver <trivial> Range: {-536870912, ..., 536870911}	9000000
presolving/trivial/timing	timing mask of presolver <trivial> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	4

5.34.4.213 presolving/tworowbnd

Option	Description	Default
presolving/tworowbnd/maxcombinefails	maximal number of consecutive useless row combines Range: {-1, ..., ∞}	1000
presolving/tworowbnd/maxconsiderednonzeros	maximal number of considered non-zeros within one row (-1: no limit) Range: {-1, ..., ∞}	100
presolving/tworowbnd/maxhashfac	Maximum number of hashlist entries as multiple of number of rows in the problem (-1: no limit) Range: {-1, ..., ∞}	10
presolving/tworowbnd/maxpairfac	Maximum number of processed row pairs as multiple of the number of rows in the problem (-1: no limit) Range: {-1, ..., ∞}	1
presolving/tworowbnd/maxretrievefails	maximal number of consecutive useless hashtable retrieves Range: {-1, ..., ∞}	1000
presolving/tworowbnd/maxrounds	maximal number of presolving rounds the presolver participates in (-1: no limit) Range: {-1, ..., ∞}	0
Options for expert users		
presolving/tworowbnd/enablecopy	should tworowbnd presolver be copied to sub-SCIPs? Range: boolean	1
presolving/tworowbnd/priority	priority of presolver <tworowbnd> Range: {-536870912, ..., 536870911}	-2000
presolving/tworowbnd/timing	timing mask of presolver <tworowbnd> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {4, ..., 60}	16

5.34.4.214 propagating

Option	Description	Default
propagating/abortoncutoff	should propagation be aborted immediately? setting this to FALSE could help conflict analysis to produce more conflict constraints Range: boolean	1
propagating/maxrounds	maximal number of propagation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	100
propagating/maxroundsroot	maximal number of propagation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	1000

5.34.4.215 propagating/dualfix

Option	Description	Default
propagating/dualfix/freq	frequency for calling propagator <dualfix> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
propagating/dualfix/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
propagating/dualfix/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	0
propagating/dualfix/presolpriority	presolving priority of propagator <dualfix> Range: {-536870912, ..., 536870911}	8000000
propagating/dualfix/presoltiming	timing mask of the presolving method of propagator <dualfix> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	4
propagating/dualfix/priority	priority of propagator <dualfix> Range: {-536870912, ..., 536870911}	8000000
propagating/dualfix/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	1

5.34.4.216 propagating/genvbounds

Option	Description	Default
propagating/genvbounds/freq	frequency for calling propagator <genvbounds> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
propagating/genvbounds/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		

Option	Description	Default
propagating/genvbounds/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	0
propagating/genvbounds/global	apply global propagation? Range: boolean	1
propagating/genvbounds/presolpriority	presolving priority of propagator <genvbounds> Range: {-536870912, ..., 536870911}	-2000000
propagating/genvbounds/presoltiming	timing mask of the presolving method of propagator <genvbounds> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	4
propagating/genvbounds/priority	priority of propagator <genvbounds> Range: {-536870912, ..., 536870911}	3000000
propagating/genvbounds/propasconss	should genvbounds be transformed to (linear) constraints? Range: boolean	0
propagating/genvbounds/propinrootnode	apply genvbounds in root node if no new incumbent was found? Range: boolean	1
propagating/genvbounds/sort	sort genvbounds and wait for bound change events? Range: boolean	1
propagating/genvbounds/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	15

5.34.4.217 propagating/nlobbt

Option	Description	Default
propagating/nlobbt/addlprows	should non-initial LP rows be used? Range: boolean	1
propagating/nlobbt/freq	frequency for calling propagator <nlobbt> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
propagating/nlobbt/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
propagating/nlobbt/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	1
propagating/nlobbt/feastolfac	factor for NLP feasibility tolerance Range: [0, 1]	0.01
propagating/nlobbt/itlimitfactor	LP iteration limit for nlobbt will be this factor times total LP iterations in root node Range: [0, ∞]	2
propagating/nlobbt/minlinearfrac	minimum (convex nlrows)/(linear nlrows) threshold to apply propagator Range: [0, ∞]	0.02

Option	Description	Default
propagating/nlobbt/minnonconvexfrac	(convex nlrows)/(nonconvex nlrows) threshold to apply propagator Range: [0, ∞]	0.2
propagating/nlobbt/nlpiterlimit	iteration limit of NLP solver; 0 for no limit Range: {0, ..., ∞}	500
propagating/nlobbt/nlptimelimit	time limit of NLP solver; 0.0 for no limit Range: [0, ∞]	0
propagating/nlobbt/nlpverblevel	verbosity level of NLP solver Range: {0, ..., 5}	0
propagating/nlobbt/presolpriority	presolving priority of propagator <nlobbt> Range: {-536870912, ..., 536870911}	0
propagating/nlobbt/presoltiming	timing mask of the presolving method of propagator <nlobbt> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	28
propagating/nlobbt/priority	priority of propagator <nlobbt> Range: {-536870912, ..., 536870911}	-1100000
propagating/nlobbt/relobjtolfac	factor for NLP relative objective tolerance Range: [0, 1]	0.01
propagating/nlobbt/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	4

5.34.4.218 propagating/obbt

Option	Description	Default
propagating/obbt/boundstreps	minimal relative improve for strengthening bounds Range: [0, 1]	0.001
propagating/obbt/conditionlimit	maximum condition limit used in LP solver (-1.0: no limit) Range: [-1, ∞]	-1
propagating/obbt/dualfeastol	feasibility tolerance for reduced costs used in obbt; this value is used if SCIP's dual feastol is greater Range: [0, ∞]	1e-09
propagating/obbt/freq	frequency for calling propagator <obbt> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
propagating/obbt/itlimitfactor	multiple of root node LP iterations used as total LP iteration limit for obbt (≤ 0: no limit) Range: real	10
propagating/obbt/itlimitfactorbilin	multiple of OBBT LP limit used as total LP iteration limit for solving bilinear inequality LPs (< 0 for no limit) Range: real	3
propagating/obbt/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1

Option	Description	Default
propagating/obbt/minitlimit	minimum LP iteration limit Range: {0, ..., ∞}	5000
propagating/obbt/minnonconvexity	minimum absolute value of nonconvex eigenvalues for a bilinear term Range: [0, ∞]	0.1
Options for expert users		
propagating/obbt/applyfilterrounds	try to filter bounds in so-called filter rounds by solving auxiliary LPs? Range: boolean	0
propagating/obbt/applytrivialfilter	try to filter bounds with the LP solution after each solve? Range: boolean	1
propagating/obbt/createbilinineqs	solve auxiliary LPs in order to find valid inequalities for bilinear terms? Range: boolean	1
propagating/obbt/creategenvbounds	should obbt try to provide genvbounds if possible? Range: boolean	1
propagating/obbt/createlincons	create linear constraints from inequalities for bilinear terms? Range: boolean	0
propagating/obbt/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	1
propagating/obbt/genvbdsduringfilter	should we try to generate genvbounds during trivial and aggressive filtering? Range: boolean	1
propagating/obbt/genvbdsduringsepa	try to create genvbounds during separation process? Range: boolean	1
propagating/obbt/minfilter	minimal number of filtered bounds to apply another filter round Range: {1, ..., ∞}	2
propagating/obbt/normalize	should coefficients in filtering be normalized w.r.t. the domains sizes? Range: boolean	1
propagating/obbt/onlynonconvexvars	only apply obbt on non-convex variables Range: boolean	1
propagating/obbt/orderingalgo	select the type of ordering algorithm which should be used (0: no special ordering, 1: greedy, 2: greedy reverse) Range: {0, ..., 2}	1
propagating/obbt/presolpriority	presolving priority of propagator <obbt> Range: {-536870912, ..., 536870911}	0
propagating/obbt/presoltiming	timing mask of the presolving method of propagator <obbt> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	28
propagating/obbt/priority	priority of propagator <obbt> Range: {-536870912, ..., 536870911}	-1000000
propagating/obbt/propagatefreq	trigger a propagation round after that many bound tightenings (0: no propagation) Range: {0, ..., ∞}	0

Option	Description	Default
propagating/obbt/sepamaxiter	maximum number of iteration spend to separate an obbt LP solution Range: {0, ..., ∞}	10
propagating/obbt/sepaminiter	minimum number of iteration spend to separate an obbt LP solution Range: {0, ..., ∞}	0
propagating/obbt/separatesol	should the obbt LP solution be separated? Range: boolean	0
propagating/obbt/tightcontboundsprobing	should continuous bounds be tightened during the probing mode? Range: boolean	0
propagating/obbt/tightintboundsprobing	should integral bounds be tightened during the probing mode? Range: boolean	1
propagating/obbt/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	4

5.34.4.219 propagating/probing

Option	Description	Default
propagating/probing/freq	frequency for calling propagator <probing> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
propagating/probing/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1
propagating/probing/maxruns	maximal number of runs, probing participates in (-1: no limit) Range: {-1, ..., ∞}	1
Options for expert users		
propagating/probing/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	1
propagating/probing/maxdepth	maximal depth until propagation is executed(-1: no limit) Range: {-1, ..., ∞}	-1
propagating/probing/maxfixings	maximal number of fixings found, until probing is interrupted (0: don't iterrupt) Range: {0, ..., ∞}	25
propagating/probing/maxsumuseless	maximal number of probings without fixings, until probing is aborted (0: don't abort) Range: {0, ..., ∞}	0
propagating/probing/maxtotaluseless	maximal number of successive probings without fixings, bound changes, and implications, until probing is aborted (0: don't abort) Range: {0, ..., ∞}	50
propagating/probing/maxuseless	maximal number of successive probings without fixings, until probing is aborted (0: don't abort) Range: {0, ..., ∞}	1000

Option	Description	Default
propagating/probing/presolpriority	presolving priority of propagator <probing> Range: {-536870912, ..., 536870911}	-100000
propagating/probing/presoltiming	timing mask of the presolving method of propagator <probing> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	16
propagating/probing/priority	priority of propagator <probing> Range: {-536870912, ..., 536870911}	-100000
propagating/probing/proprounds	maximal number of propagation rounds in probing subproblems (-1: no limit, 0: auto) Range: {-1, ..., ∞}	-1
propagating/probing/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	4

5.34.4.220 propagating/pseudoobj

Option	Description	Default
propagating/pseudoobj/freq	frequency for calling propagator <pseudoobj> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
propagating/pseudoobj/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
propagating/pseudoobj/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	0
propagating/pseudoobj/force	should the propagator be forced even if active pricer are present? Range: boolean	0
propagating/pseudoobj/maximplvars	maximum number of binary variables the implications are used if turned on (-1: unlimited)? Range: {-1, ..., ∞}	50000
propagating/pseudoobj/maxnewvars	number of variables added after the propagator is reinitialized? Range: {0, ..., ∞}	1000
propagating/pseudoobj/maxvarsfrac	maximal fraction of non-binary variables with non-zero objective without a bound reduction before aborted Range: [0, 1]	0.1
propagating/pseudoobj/minuseless	minimal number of successive non-binary variable propagations without a bound reduction before aborted Range: {0, ..., ∞}	100
propagating/pseudoobj/presolpriority	presolving priority of propagator <pseudoobj> Range: {-536870912, ..., 536870911}	6000000

Option	Description	Default
propagating/pseudoobj/presoltiming	timing mask of the presolving method of propagator <pseudoobj> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	4
propagating/pseudoobj/priority	priority of propagator <pseudoobj> Range: {-536870912, ..., 536870911}	3000000
propagating/pseudoobj/propcutoffbound	propagate new cutoff bound directly globally Range: boolean	1
propagating/pseudoobj/propfullinroot	whether to propagate all non-binary variables when we are propagating the root node Range: boolean	1
propagating/pseudoobj/propuseimplics	use implications to strengthen the propagation of binary variable (increasing the objective change)? Range: boolean	1
propagating/pseudoobj/respropuseimplics	use implications to strengthen the resolve propagation of binary variable (increasing the objective change)? Range: boolean	1
propagating/pseudoobj/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	7

5.34.4.221 propagating/redcost

Option	Description	Default
propagating/redcost/continuous	should reduced cost fixing be also applied to continuous variables? Range: boolean	0
propagating/redcost/freq	frequency for calling propagator <redcost> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
propagating/redcost/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1
propagating/redcost/useimplics	should implications be used to strength the reduced cost for binary variables? Range: boolean	0
Options for expert users		
propagating/redcost/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	0
propagating/redcost/force	should the propagator be forced even if active pricer are present? Range: boolean	0
propagating/redcost/presolpriority	presolving priority of propagator <redcost> Range: {-536870912, ..., 536870911}	0
propagating/redcost/presoltiming	timing mask of the presolving method of propagator <redcost> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	28

Option	Description	Default
propagating/redcost/priority	priority of propagator <redcost> Range: {-536870912, ..., 536870911}	1000000
propagating/redcost/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	6

5.34.4.222 propagating/rootredcost

Option	Description	Default
propagating/rootredcost/freq	frequency for calling propagator <rootredcost> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
propagating/rootredcost/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1
Options for expert users		
propagating/rootredcost/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	0
propagating/rootredcost/force	should the propagator be forced even if active pricer are present? Range: boolean	0
propagating/rootredcost/onlybinary	should only binary variables be propagated? Range: boolean	0
propagating/rootredcost/presolpriority	presolving priority of propagator <rootredcost> Range: {-536870912, ..., 536870911}	0
propagating/rootredcost/presoltiming	timing mask of the presolving method of propagator <rootredcost> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	28
propagating/rootredcost/priority	priority of propagator <rootredcost> Range: {-536870912, ..., 536870911}	10000000
propagating/rootredcost/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	5

5.34.4.223 propagating/symmetry

Option	Description	Default
propagating/symmetry/freq	frequency for calling propagator <symmetry> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
propagating/symmetry/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1

Option	Description	Default
Options for expert users		
propagating/symmetry/addconflictcuts	Should Schreier Sims constraints be added if we use a conflict based rule? Range: boolean	1
propagating/symmetry/addconsstiming	timing of adding constraints (0 = before presolving, 1 = during presolving, 2 = after presolving) Range: {0, ..., 2}	2
propagating/symmetry/addstrongsbcs	Should strong SBCs for enclosing orbit of symmetric subgroups be added if orbitopes are not used? Range: boolean	0
propagating/symmetry/addsymresacks	Add inequalities for symresacks for each generator? Range: boolean	0
propagating/symmetry/addweaksbcs	Should we add weak SBCs for enclosing orbit of symmetric subgroups? Range: boolean	1
propagating/symmetry/checksymmetries	Should all symmetries be checked after computation? Range: boolean	0
propagating/symmetry/compresssymmetries	Should non-affected variables be removed from permutation to save memory? Range: boolean	1
propagating/symmetry/compressthreshold	Compression is used if percentage of moved vars is at most the threshold. Range: [0, 1]	0.5
propagating/symmetry/conssaddlp	Should the symmetry breaking constraints be added to the LP? Range: boolean	1
propagating/symmetry/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	0
propagating/symmetry/detectorbitopes	Should we check whether the components of the symmetry group can be handled by orbitopes? Range: boolean	1
propagating/symmetry/detectsubgroups	Should we try to detect symmetric subgroups of the symmetry group on binary variables? Range: boolean	1
propagating/symmetry/displaynorbitvars	Should the number of variables affected by some symmetry be displayed? Range: boolean	0
propagating/symmetry/doubleequations	Double equations to positive/negative version? Range: boolean	0
propagating/symmetry/maxgenerators	limit on the number of generators that should be produced within symmetry detection (0 = no limit) Range: {0, ..., ∞}	1500

Option	Description	Default
propagating/symmetry/maxnconsssubgroup	maximum number of constraints up to which subgroup structures are detected Range: {0, ..., ∞}	500000
propagating/symmetry/ofsymcomptiming	timing of symmetry computation for orbital fixing (0 = before presolving, 1 = during presolving, 2 = at first call) Range: {0, ..., 2}	2
propagating/symmetry/onlybinarysymmetry	Is only symmetry on binary variables used? Range: boolean	1
propagating/symmetry/performpresolving	run orbital fixing during presolving? Range: boolean	0
propagating/symmetry/preferlessrows	Shall orbitopes with less rows be preferred in detection? Range: boolean	1
propagating/symmetry/presolpriority	presolving priority of propagator <symmetry> Range: {-536870912, ..., 536870911}	-10000000
propagating/symmetry/presoltiming	timing mask of the presolving method of propagator <symmetry> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	16
propagating/symmetry/priority	priority of propagator <symmetry> Range: {-536870912, ..., 536870911}	-1000000
propagating/symmetry/recomputerestart	recompute symmetries after a restart has occured? (0 = never) Range: {0, ..., 0}	0
propagating/symmetry/sstaddcuts	Should Schreier Sims constraints be added? Range: boolean	1
propagating/symmetry/sstleaderrule	rule to select the leader in an orbit (0: first var; 1: last var; 2: var having most conflicting vars in orbit; 3: var having most conflicting vars in problem) Range: {0, ..., 3}	0
propagating/symmetry/sstleadervartype	bitset encoding which variable types can be leaders (1: binary; 2: integer; 4: impl. int; 8: continuous);if multiple types are allowed, take the one with most affected vars Range: {1, ..., 15}	14
propagating/symmetry/sstmixedcomponents	Should Schreier Sims constraints be added if a symmetry component contains variables of different types? Range: boolean	1
propagating/symmetry/ssttiebreakrule	rule to select the orbit in Schreier Sims inequalities (variable in 0: minimum size orbit; 1: maximum size orbit; 2: orbit with most variables in conflict with leader) Range: {0, ..., 2}	1
propagating/symmetry/symfixnonbinaryvars	Whether all non-binary variables shall be not affected by symmetries if OF is active? Range: boolean	0

Option	Description	Default
propagating/symmetry/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	1
propagating/symmetry/usecolumnsparsity	Should the number of conss a variable is contained in be exploited in symmetry detection? Range: boolean	0
propagating/symmetry/usedynamicprop	whether dynamic propagation should be used for full orbitopes Range: boolean	1

5.34.4.224 propagating/vbounds

Option	Description	Default
propagating/vbounds/detectcycles	should cycles in the variable bound graph be identified? Range: boolean	0
propagating/vbounds/dotoposort	should the bounds be topologically sorted in advance? Range: boolean	1
propagating/vbounds/freq	frequency for calling propagator <vbounds> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	1
propagating/vbounds/maxcliquesexhaustive	maximum number of cliques per variable to run clique table analysis in exhaustive presolving Range: [0, ∞]	100
propagating/vbounds/maxcliquesmedium	maximum number of cliques per variable to run clique table analysis in medium presolving Range: [0, ∞]	50
propagating/vbounds/maxprerounds	maximal number of presolving rounds the propagator participates in (-1: no limit) Range: {-1, ..., ∞}	-1
propagating/vbounds/minnewcliques	minimum percentage of new cliques to trigger another clique table analysis Range: [0, 1]	0.1
propagating/vbounds/usebdwidening	should bound widening be used to initialize conflict analysis? Range: boolean	1
Options for expert users		
propagating/vbounds/delay	should propagator be delayed, if other propagators found reductions? Range: boolean	0
propagating/vbounds/presolpriority	presolving priority of propagator <vbounds> Range: {-536870912, ..., 536870911}	-90000
propagating/vbounds/presoltiming	timing mask of the presolving method of propagator <vbounds> (4:FAST, 8:MEDIUM, 16:EXHAUSTIVE, 32:FINAL) Range: {2, ..., 60}	24

Option	Description	Default
propagating/vbounds/priority	priority of propagator <vbounds> Range: {-536870912, ..., 536870911}	3000000
propagating/vbounds/sortcliques	should cliques be regarded for the topological sort? Range: boolean	0
propagating/vbounds/timingmask	timing when propagator should be called (1:BEFORELP, 2:DURINGLPLOOP, 4:AFTERLPLOOP, 15:ALWAYS)) Range: {1, ..., 15}	5
propagating/vbounds/usecliques	should cliques be propagated? Range: boolean	0
propagating/vbounds/useimplics	should implications be propagated? Range: boolean	0
propagating/vbounds/usevbounds	should vbounds be propagated? Range: boolean	1

5.34.4.225 randomization

Option	Description	Default
randomization/lpseed	random seed for LP solver, e.g. for perturbations in the simplex (0: LP default) Range: {0, ..., ∞}	0
randomization/permutationseed	seed value for permuting the problem after reading/transformation (0: no permutation) Range: {0, ..., ∞}	0
randomization/randomseedshift	global shift of all random seeds in the plugins and the LP random seed Range: {0, ..., ∞}	0
Options for expert users		
randomization/permuteconss	should order of constraints be permuted (depends on permutationseed)? Range: boolean	1
randomization/permutevars	should order of variables be permuted (depends on permutationseed)? Range: boolean	0

5.34.4.226 separating

Option	Description	Default
separating/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separation (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/maxcoefratio	maximal ratio between coefficients in strongcg, cmir, and flowcover cuts Range: [1, ∞]	10000
separating/maxcoefratiofacrowprep	maximal ratio between coefficients (as factor of 1/feastol) to ensure in rowprep cleanup Range: [0, ∞]	10

Option	Description	Default
separating/maxcuts	maximal number of cuts separated per separation round (0: disable local separation) Range: {0, ..., ∞}	100
separating/maxcutsroot	maximal number of separated cuts at the root node (0: disable root node separation) Range: {0, ..., ∞}	2000
separating/maxlocalbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying local separation (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/maxroundsroot	maximal number of separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/maxstallrounds	maximal number of consecutive separation rounds without objective or integrality improvement in local nodes (-1: no additional restriction) Range: {-1, ..., ∞}	1
separating/maxstallroundsroot	maximal number of consecutive separation rounds without objective or integrality improvement in the root node (-1: no additional restriction) Range: {-1, ..., ∞}	10
separating/minactivityquot	minimum cut activity quotient to convert cuts into constraints during a restart (0.0: all cuts are converted) Range: [0, 1]	0.8
separating/minefficacy	minimal efficacy for a cut to enter the LP Range: [0, ∞]	0.0001
separating/minefficacyroot	minimal efficacy for a cut to enter the LP in the root node Range: [0, ∞]	0.0001
separating/poolfreq	separation frequency for the global cut pool (-1: disable global cut pool, 0: only separate pool at the root) Range: {-1, ..., 65534}	10
Options for expert users		
separating/cutagelimit	maximum age a cut can reach before it is deleted from the global cut pool, or -1 to keep all cuts Range: {-1, ..., ∞}	80
separating/cutselrestart	cut selection during restart ('a'ge, activity 'q'uotient) Range: a, q	a
separating/cutselsubscip	cut selection for sub SCIPs ('a'ge, activity 'q'uotient) Range: a, q	a
separating/efficacynorm	row norm to use for efficacy calculation ('e'uclidean, 'm'aximum, 's'um, 'd'iscrete) Range: e, m, s, d	e
separating/filtercutpoolrel	should cutpool separate only cuts with high relative efficacy? Range: boolean	1

Option	Description	Default
separating/maxaddrounds	maximal additional number of separation rounds in subsequent price-and-cut loops (-1: no additional restriction) Range: {-1, ..., ∞}	1
separating/maxroundsrootsubrun	maximal number of separation rounds in the root node of a subsequent run (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/maxruns	maximal number of runs for which separation is enabled (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/orthofunc	function used for calc. scalar prod. in orthogonality test ('e'uclidean, 'd'iscrete) Range: e, d	e

5.34.4.227 separating/aggregation

Option	Description	Default
separating/aggregation/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight? Range: boolean	1
separating/aggregation/freq	frequency for calling separator <aggregation> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
separating/aggregation/maxrounds	maximal number of cmir separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/aggregation/maxroundsroot	maximal number of cmir separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/aggregation/maxsepacuts	maximal number of cmir cuts separated per separation round Range: {0, ..., ∞}	100
separating/aggregation/maxsepacutsroot	maximal number of cmir cuts separated per separation round in the root node Range: {0, ..., ∞}	500
Options for expert users		
separating/aggregation/aggrtol	tolerance for bound distances used to select continuous variable in current aggregated constraint to be eliminated Range: [0, ∞]	0.01
separating/aggregation/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/aggregation/densityoffset	additional number of variables allowed in row on top of density Range: {0, ..., ∞}	100
separating/aggregation/densityscore	weight of row density in the aggregation scoring of the rows Range: [0, ∞]	0.0001

Option	Description	Default
separating/aggregation/expbackoff	base for exponential increase of frequency at which separator <aggregation> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/aggregation/fixintegralrhs	should an additional variable be complemented if f0 = 0? Range: boolean	1
separating/aggregation/maxaggdensity	maximal density of aggregated row Range: [0, 1]	0.2
separating/aggregation/maxaggrs	maximal number of aggregations for each row per separation round Range: {0, ..., ∞}	3
separating/aggregation/maxaggrsroot	maximal number of aggregations for each row per separation round in the root node Range: {0, ..., ∞}	6
separating/aggregation/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <aggregation> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/aggregation/maxfails	maximal number of consecutive unsuccessful aggregation tries (-1: unlimited) Range: {-1, ..., ∞}	20
separating/aggregation/maxfailsroot	maximal number of consecutive unsuccessful aggregation tries in the root node (-1: unlimited) Range: {-1, ..., ∞}	100
separating/aggregation/maxrowdensity	maximal density of row to be used in aggregation Range: [0, 1]	0.05
separating/aggregation/maxrowfac	maximal row aggregation factor Range: [0, ∞]	10000
separating/aggregation/maxslack	maximal slack of rows to be used in aggregation Range: [0, ∞]	0
separating/aggregation/maxslackroot	maximal slack of rows to be used in aggregation in the root node Range: [0, ∞]	0.1
separating/aggregation/maxtestdelta	maximal number of different deltas to try (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/aggregation/maxtries	maximal number of rows to start aggregation with per separation round (-1: unlimited) Range: {-1, ..., ∞}	200
separating/aggregation/maxtriesroot	maximal number of rows to start aggregation with per separation round in the root node (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/aggregation/priority	priority of separator <aggregation> Range: {-536870912, ..., 536870911}	-3000

Option	Description	Default
separating/aggregation/slackscore	weight of slack in the aggregation scoring of the rows Range: [0, ∞]	0.001
separating/aggregation/trynegscaling	should negative values also be tested in scaling? Range: boolean	1

5.34.4.228 separating/cgmip

Option	Description	Default
separating/cgmip/addviolationcons	add constraint to subscip that only allows violated cuts (otherwise add obj. limit)? Range: boolean	0
separating/cgmip/addviolconshdlr	add constraint handler to filter out violated cuts? Range: boolean	0
separating/cgmip/cmirownbounds	tell CMIR-generator which bounds to used in rounding? Range: boolean	0
separating/cgmip/conshdlrusenorm	should the violation constraint handler use the norm of a cut to check for feasibility? Range: boolean	1
separating/cgmip/contconvert	Convert some integral variables to be continuous to reduce the size of the sub-MIP? Range: boolean	0
separating/cgmip/contconvfrac	fraction of integral variables converted to be continuous (if contconvert) Range: [0, 1]	0.1
separating/cgmip/contconvmin	minimum number of integral variables before some are converted to be continuous Range: {-1, ..., ∞}	100
separating/cgmip/decisiontree	Use decision tree to turn separation on/off? Range: boolean	0
separating/cgmip/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight? Range: boolean	1
separating/cgmip/earlyterm	terminate separation if a violated (but possibly sub-optimal) cut has been found? Range: boolean	1
separating/cgmip/freq	frequency for calling separator <cgmip> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
separating/cgmip/genprimalsols	Try to generate primal solutions from Gomory cuts? Range: boolean	0
separating/cgmip/intconvert	Convert some integral variables attaining fractional values to have integral value? Range: boolean	0
separating/cgmip/intconvfrac	fraction of frac. integral variables converted to have integral value (if intconvert) Range: [0, 1]	0.1
separating/cgmip/intconvmin	minimum number of integral variables before some are converted to have integral value Range: {-1, ..., ∞}	100

Option	Description	Default
separating/cgmip/maxdepth	maximal depth at which the separator is applied (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/cgmip/maxnodelimit	maximum number of nodes considered for sub-MIP (-1: unlimited) Range: {-1, ..., ∞}	5000
separating/cgmip/maxrounds	maximal number of cgmip separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	5
separating/cgmip/maxroundsroot	maximal number of cgmip separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	50
separating/cgmip/maxrowage	maximal age of rows to consider if onlyactiverows is false Range: {-1, ..., ∞}	-1
separating/cgmip/minnodelimit	minimum number of nodes considered for sub-MIP (-1: unlimited) Range: {-1, ..., ∞}	500
separating/cgmip/objlone	Should the objective of the sub-MIP minimize the l1-norm of the multipliers? Range: boolean	0
separating/cgmip/objweightsize	Weight each row by its size? Range: boolean	1
separating/cgmip/onlyactiverows	Use only active rows to generate cuts? Range: boolean	0
separating/cgmip/onlyintvars	Generate cuts for problems with only integer variables? Range: boolean	0
separating/cgmip/onlyrankone	Separate only rank 1 inequalities w.r.t. CG-MIP separator? Range: boolean	0
separating/cgmip/output	Should information about the sub-MIP and cuts be displayed? Range: boolean	0
separating/cgmip/primalseparation	only separate cuts that are tight for the best feasible solution? Range: boolean	1
separating/cgmip/skipmultbounds	Skip the upper bounds on the multipliers in the sub-MIP? Range: boolean	1
separating/cgmip/subscipfast	Should the settings for the sub-MIP be optimized for speed? Range: boolean	1
separating/cgmip/usecmir	use CMIR-generator (otherwise add cut directly)? Range: boolean	1
separating/cgmip/usecutpool	use cutpool to store CG-cuts even if the are not efficient? Range: boolean	1
separating/cgmip/useobjlb	Use lower bound on objective function (via primal solution)? Range: boolean	0

Option	Description	Default
separating/cgmip/useobjub	Use upper bound on objective function (via primal solution)? Range: boolean	0
separating/cgmip/usestrongcg	use strong CG-function to strengthen cut? Range: boolean	0
Options for expert users		
separating/cgmip/cutcoefbnd	bounds on the values of the coefficients in the CG-cut Range: [0, ∞]	1000
separating/cgmip/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/cgmip/expbackoff	base for exponential increase of frequency at which separator <cgmip> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/cgmip/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <cgmip> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/cgmip/memorylimit	memory limit for sub-MIP Range: [0, ∞]	∞
separating/cgmip/objweight	weight used for the row combination coefficient in the sub-MIP objective Range: [0, ∞]	0.001
separating/cgmip/priority	priority of separator <cgmip> Range: {-536870912, ..., 536870911}	-1000
separating/cgmip/timelimit	time limit for sub-MIP Range: [0, ∞]	∞

5.34.4.229 separating/clique

Option	Description	Default
separating/clique/freq	frequency for calling separator <clique> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
separating/clique/maxsepacuts	maximal number of clique cuts separated per separation round (-1: no limit) Range: {-1, ..., ∞}	10
Options for expert users		
separating/clique/backtrackfreq	frequency for premature backtracking up to tree level 1 (0: no backtracking) Range: {0, ..., ∞}	1000
separating/clique/cliquedensity	minimal density of cliques to use a dense clique table Range: [0, 1]	0
separating/clique/cliquetablemem	maximal memory size of dense clique table (in kb) Range: [0, 2.09715e+06]	20000
separating/clique/delay	should separator be delayed, if other separators found cuts? Range: boolean	0

Option	Description	Default
separating/clique/expbackoff	base for exponential increase of frequency at which separator <clique> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/clique/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <clique> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/clique/maxtreenodes	maximal number of nodes in branch and bound tree (-1: no limit) Range: {-1, ..., ∞}	10000
separating/clique/maxzeroextensions	maximal number of zero-valued variables extending the clique (-1: no limit) Range: {-1, ..., ∞}	1000
separating/clique/priority	priority of separator <clique> Range: {-536870912, ..., 536870911}	-5000
separating/clique/scaleval	factor for scaling weights Range: [1, ∞]	1000

5.34.4.230 separating/closecuts

Option	Description	Default
separating/closecuts/freq	frequency for calling separator <closecuts> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
Options for expert users		
separating/closecuts/closethres	threshold on number of generated cuts below which the ordinary separation is started Range: {-1, ..., ∞}	50
separating/closecuts/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/closecuts/expbackoff	base for exponential increase of frequency at which separator <closecuts> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/closecuts/inclobjcutoff	include an objective cutoff when computing the relative interior? Range: boolean	0
separating/closecuts/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <closecuts> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/closecuts/maxlpiterfactor	factor for maximal LP iterations in relative interior computation compared to node LP iterations (negative for no limit) Range: [-1, ∞]	10
separating/closecuts/maxunsuccessful	turn off separation in current node after unsuccessful calls (-1 never turn off) Range: {-1, ..., ∞}	0

Option	Description	Default
separating/closecuts/priority	priority of separator <closecuts> Range: {-536870912, ..., 536870911}	1000000
separating/closecuts/recomputerelint	recompute relative interior point in each separation call? Range: boolean	0
separating/closecuts/sepacombvalue	convex combination value for close cuts Range: [0, 1]	0.3
separating/closecuts/separelint	generate close cuts w.r.t. relative interior point (best solution otherwise)? Range: boolean	1

5.34.4.231 separating/cmir

Option	Description	Default
separating/cmir/freq	frequency for calling separator <cmir> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
Options for expert users		
separating/cmir/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/cmir/expbackoff	base for exponential increase of frequency at which separator <cmir> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/cmir/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <cmir> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/cmir/priority	priority of separator <cmir> Range: {-536870912, ..., 536870911}	-100000

5.34.4.232 separating/convexproj

Option	Description	Default
separating/convexproj/freq	frequency for calling separator <convexproj> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
separating/convexproj/maxdepth	maximal depth at which the separator is applied (-1: unlimited) Range: {-1, ..., ∞}	-1
Options for expert users		
separating/convexproj/delay	should separator be delayed, if other separators found cuts? Range: boolean	1
separating/convexproj/expbackoff	base for exponential increase of frequency at which separator <convexproj> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4

Option	Description	Default
separating/convexproj/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <convexproj> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/convexproj/nlpiterlimit	iteration limit of NLP solver; 0 for no limit Range: {0, ..., ∞}	250
separating/convexproj/priority	priority of separator <convexproj> Range: {-536870912, ..., 536870911}	0

5.34.4.233 separating/disjunctive

Option	Description	Default
separating/disjunctive/freq	frequency for calling separator <disjunctive> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
separating/disjunctive/maxrank	maximal rank of a disj. cut that could not be scaled to integral coefficients (-1: unlimited) Range: {-1, ..., ∞}	20
separating/disjunctive/maxrankintegral	maximal rank of a disj. cut that could be scaled to integral coefficients (-1: unlimited) Range: {-1, ..., ∞}	-1
Options for expert users		
separating/disjunctive/delay	should separator be delayed, if other separators found cuts? Range: boolean	1
separating/disjunctive/expbackoff	base for exponential increase of frequency at which separator <disjunctive> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/disjunctive/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <disjunctive> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/disjunctive/maxconfsdelay	delay separation if number of conflict graph edges is larger than predefined value (-1: no limit) Range: {-1, ..., ∞}	100000
separating/disjunctive/maxdepth	node depth of separating bipartite disjunctive cuts (-1: no limit) Range: {-1, ..., ∞}	-1
separating/disjunctive/maxinvcuts	maximal number of cuts investigated per iteration in a branching node Range: {0, ..., ∞}	50
separating/disjunctive/maxinvcutsroot	maximal number of cuts investigated per iteration in the root node Range: {0, ..., ∞}	250

Option	Description	Default
separating/disjunctive/maxrounds	maximal number of separation rounds per iteration in a branching node (-1: no limit) Range: {-1, ..., ∞}	25
separating/disjunctive/maxroundsroot	maximal number of separation rounds in the root node (-1: no limit) Range: {-1, ..., ∞}	100
separating/disjunctive/maxweightrange	maximal valid range max(|weights|)/min(|weights|) of row weights Range: [1, ∞]	1000
separating/disjunctive/priority	priority of separator <disjunctive> Range: {-536870912, ..., 536870911}	10
separating/disjunctive/strengthen	strengthen cut if integer variables are present. Range: boolean	1

5.34.4.234 separating/eccuts

Option	Description	Default
separating/eccuts/cutmaxrange	maximal coef. range of a cut (max coef. divided by min coef.) in order to be added to LP relaxation Range: [0, ∞]	1e+07
separating/eccuts/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight? Range: boolean	1
separating/eccuts/freq	frequency for calling separator <eccuts> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
separating/eccuts/maxdepth	maximal depth at which the separator is applied (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/eccuts/maxrounds	maximal number of eccuts separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	10
separating/eccuts/maxroundsroot	maximal number of eccuts separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	250
separating/eccuts/maxsepacuts	maximal number of edge-concave cuts separated per separation round Range: {0, ..., ∞}	10
separating/eccuts/maxsepacutsroot	maximal number of edge-concave cuts separated per separation round in the root node Range: {0, ..., ∞}	50
separating/eccuts/minviolation	minimal violation of an edge-concave cut to be separated Range: [0, 0.5]	0.3
Options for expert users		
separating/eccuts/delay	should separator be delayed, if other separators found cuts? Range: boolean	0

Option	Description	Default
separating/eccuts/expbackoff	base for exponential increase of frequency at which separator <eccuts> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/eccuts/maxaggrsize	search for edge-concave aggregations of at most this size Range: {3, ..., 5}	4
separating/eccuts/maxbilinterms	maximum number of bilinear terms allowed to be in a quadratic constraint Range: {0, ..., ∞}	500
separating/eccuts/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <eccuts> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/eccuts/maxstallrounds	maximum number of unsuccessful rounds in the edge-concave aggregation search Range: {0, ..., ∞}	5
separating/eccuts/minaggrsize	search for edge-concave aggregations of at least this size Range: {3, ..., 5}	3
separating/eccuts/priority	priority of separator <eccuts> Range: {-536870912, ..., 536870911}	-13000

5.34.4.235 separating/flowcover

Option	Description	Default
separating/flowcover/freq	frequency for calling separator <flowcover> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
Options for expert users		
separating/flowcover/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/flowcover/expbackoff	base for exponential increase of frequency at which separator <flowcover> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/flowcover/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <flowcover> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/flowcover/priority	priority of separator <flowcover> Range: {-536870912, ..., 536870911}	-100000

5.34.4.236 separating/gauge

Option	Description	Default
separating/gauge/freq	frequency for calling separator <gauge> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
Options for expert users		
separating/gauge/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/gauge/expbackoff	base for exponential increase of frequency at which separator <gauge> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/gauge/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <gauge> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/gauge/nlpiterlimit	iteration limit of NLP solver; 0 for no limit Range: {0, ..., ∞}	1000
separating/gauge/priority	priority of separator <gauge> Range: {-536870912, ..., 536870911}	0

5.34.4.237 separating/gomory

Option	Description	Default
separating/gomory/away	minimal integrality violation of a basis variable in order to try Gomory cut Range: [0.0001, 0.5]	0.01
separating/gomory/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight? Range: boolean	1
separating/gomory/freq	frequency for calling separator <gomory> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
separating/gomory/maxrank	maximal rank of a gomory cut that could not be scaled to integral coefficients (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/gomory/maxrankintegral	maximal rank of a gomory cut that could be scaled to integral coefficients (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/gomory/maxrounds	maximal number of gomory separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	5
separating/gomory/maxroundsroot	maximal number of gomory separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	10
separating/gomory/maxsepacuts	maximal number of gomory cuts separated per separation round Range: {0, ..., ∞}	50
separating/gomory/maxsepacutsroot	maximal number of gomory cuts separated per separation round in the root node Range: {0, ..., ∞}	200

Option	Description	Default
Options for expert users		
separating/gomory/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/gomory/delayedcuts	should cuts be added to the delayed cut pool? Range: boolean	0
separating/gomory/expbackoff	base for exponential increase of frequency at which separator <gomory> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/gomory/forcecuts	if conversion to integral coefficients failed still consider the cut Range: boolean	1
separating/gomory/genbothgomscg	Should both Gomory and strong CG cuts be generated (otherwise take best)? Range: boolean	1
separating/gomory/makeintegral	try to scale cuts to integral coefficients Range: boolean	0
separating/gomory/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <gomory> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/gomory/priority	priority of separator <gomory> Range: {-536870912, ..., 536870911}	-1000
separating/gomory/separaterows	separate rows with integral slack Range: boolean	1
separating/gomory/sidetypebasis	choose side types of row (lhs/rhs) based on basis information? Range: boolean	1
separating/gomory/trystrongcg	try to generate strengthened Chvatal-Gomory cuts? Range: boolean	1

5.34.4.238 separating/gomorymi

Option	Description	Default
separating/gomorymi/freq	frequency for calling separator <gomorymi> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
Options for expert users		
separating/gomorymi/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/gomorymi/expbackoff	base for exponential increase of frequency at which separator <gomorymi> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4

Option	Description	Default
separating/gomorymi/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <gomorymi> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/gomorymi/priority	priority of separator <gomorymi> Range: {-536870912, ..., 536870911}	-100000

5.34.4.239 separating/impliedbounds

Option	Description	Default
separating/impliedbounds/freq	frequency for calling separator <impliedbounds> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
Options for expert users		
separating/impliedbounds/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/impliedbounds/expbackoff	base for exponential increase of frequency at which separator <impliedbounds> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/impliedbounds/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <impliedbounds> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/impliedbounds/priority	priority of separator <impliedbounds> Range: {-536870912, ..., 536870911}	-50
separating/impliedbounds/usetwosizecliques	should violated inequalities for cliques with 2 variables be separated? Range: boolean	1

5.34.4.240 separating/interminor

Option	Description	Default
separating/interminor/freq	frequency for calling separator <interminor> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
separating/interminor/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	10
separating/interminor/maxroundsroot	maximal number of separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/interminor/mincutviol	minimum required violation of a cut Range: [0, ∞]	0.0001

Option	Description	Default
separating/interminor/usebounds	whether to also enforce nonegativity bounds of principle minors Range: boolean	0
separating/interminor/usestrengthening	whether to use strengthened intersection cuts to separate minors Range: boolean	0
Options for expert users		
separating/interminor/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/interminor/expbackoff	base for exponential increase of frequency at which separator <interminor> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/interminor/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <interminor> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/interminor/priority	priority of separator <interminor> Range: {-536870912, ..., 536870911}	0

5.34.4.241 separating/intobj

Option	Description	Default
separating/intobj/freq	frequency for calling separator <intobj> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
Options for expert users		
separating/intobj/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/intobj/expbackoff	base for exponential increase of frequency at which separator <intobj> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/intobj/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <intobj> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/intobj/priority	priority of separator <intobj> Range: {-536870912, ..., 536870911}	-100

5.34.4.242 separating/knapsackcover

Option	Description	Default
separating/knapsackcover/freq	frequency for calling separator <knapsackcover> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
Options for expert users		
separating/knapsackcover/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/knapsackcover/expbackoff	base for exponential increase of frequency at which separator <knapsackcover> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/knapsackcover/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <knapsackcover> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/knapsackcover/priority	priority of separator <knapsackcover> Range: {-536870912, ..., 536870911}	-100000

5.34.4.243 separating/mcf

Option	Description	Default
separating/mcf/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight? Range: boolean	1
separating/mcf/freq	frequency for calling separator <mcf> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
separating/mcf/maxsepacuts	maximal number of mcf cuts separated per separation round Range: {-1, ..., ∞}	100
separating/mcf/maxsepacutsroot	maximal number of mcf cuts separated per separation round in the root node – default separation Range: {-1, ..., ∞}	200
Options for expert users		
separating/mcf/checkcutshoreconnectivity	should we separate only if the cuts shores are connected? Range: boolean	1
separating/mcf/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/mcf/expbackoff	base for exponential increase of frequency at which separator <mcf> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/mcf/fixintegralrhs	should an additional variable be complemented if f0 = 0? Range: boolean	1

Option	Description	Default
separating/mcf/maxarcinconsistencyratio	maximum inconsistency ratio of arcs not to be deleted Range: [0, ∞]	0.5
separating/mcf/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <mcf> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/mcf/maxinconsistencyratio	maximum inconsistency ratio for separation at all Range: [0, ∞]	0.02
separating/mcf/maxtestdelta	maximal number of different deltas to try (-1: unlimited) – default separation Range: {-1, ..., ∞}	20
separating/mcf/maxweightrange	maximal valid range max(|weights|)/min(|weights|) of row weights Range: [1, ∞]	1e+06
separating/mcf/modeltype	model type of network (0: auto, 1:directed, 2:undirected) Range: {0, ..., 2}	0
separating/mcf/nclusters	number of clusters to generate in the shrunken network – default separation Range: {2, ..., 32}	5
separating/mcf/priority	priority of separator <mcf> Range: {-536870912, ..., 536870911}	-10000
separating/mcf/separateflowcutset	should we separate flowcutset inequalities on the network cuts? Range: boolean	1
separating/mcf/separateknapsack	should we separate knapsack cover inequalities on the network cuts? Range: boolean	1
separating/mcf/separatesinglenodecuts	should we separate inequalities based on single-node cuts? Range: boolean	1
separating/mcf/trynegscaling	should negative values also be tested in scaling? Range: boolean	0

5.34.4.244 separating/minor

Option	Description	Default
separating/minor/freq	frequency for calling separator <minor> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
separating/minor/ignorepackingconss	whether to ignore circle packing constraints during minor detection Range: boolean	1
separating/minor/maxminorsconst	constant for the maximum number of minors, i.e., max(const, fac ∗ # quadratic terms) Range: {0, ..., ∞}	3000

Option	Description	Default
separating/minor/maxminorsfac	factor for the maximum number of minors, i.e., max(const, fac $*$ # quadratic terms) Range: [0, ∞]	10
separating/minor/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	10
separating/minor/maxroundsroot	maximal number of separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/minor/mincutviol	minimum required violation of a cut Range: [0, ∞]	0.0001
Options for expert users		
separating/minor/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/minor/expbackoff	base for exponential increase of frequency at which separator <minor> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/minor/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <minor> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/minor/priority	priority of separator <minor> Range: {-536870912, ..., 536870911}	0

5.34.4.245 separating/mixing

Option	Description	Default
separating/mixing/freq	frequency for calling separator <mixing> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
separating/mixing/maxnunsuccessful	maximal number of consecutive unsuccessful iterations Range: {-1, ..., ∞}	10
separating/mixing/maxrounds	maximal number of mixing separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/mixing/maxroundsroot	maximal number of mixing separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	-1
Options for expert users		
separating/mixing/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/mixing/expbackoff	base for exponential increase of frequency at which separator <mixing> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4

Option	Description	Default
separating/mixing/iscutsonints	Should general integer variables be used to generate cuts? Range: boolean	0
separating/mixing/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <mixing> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/mixing/priority	priority of separator <mixing> Range: {-536870912, ..., 536870911}	-50
separating/mixing/uselocalbounds	Should local bounds be used? Range: boolean	0

5.34.4.246 separating/oddcycle

Option	Description	Default
separating/oddcycle/freq	frequency for calling separator <oddcycle> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	-1
separating/oddcycle/liftoddcycles	Should odd cycle cuts be lifted? Range: boolean	0
separating/oddcycle/maxrounds	maximal number of oddcycle separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	10
separating/oddcycle/maxroundsroot	maximal number of oddcycle separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	10
separating/oddcycle/maxsepacuts	maximal number of oddcycle cuts separated per separation round Range: {0, ..., ∞}	5000
separating/oddcycle/maxsepacutsroot	maximal number of oddcycle cuts separated per separation round in the root node Range: {0, ..., ∞}	5000
separating/oddcycle/usegls	Should the search method by Groetschel, Lovasz, Schrijver be used? Otherwise use levelgraph method by Hoffman, Padberg. Range: boolean	1
Options for expert users		
separating/oddcycle/addselfarcs	add links between a variable and its negated Range: boolean	1
separating/oddcycle/allowmultiplecuts	Even if a variable is already covered by a cut, still allow another cut to cover it too? Range: boolean	1
separating/oddcycle/cutthreshold	maximal number of other cuts s.t. separation is applied (-1 for direct call) Range: {-1, ..., ∞}	-1
separating/oddcycle/delay	should separator be delayed, if other separators found cuts? Range: boolean	0

Option	Description	Default
separating/oddcycle/expbackoff	base for exponential increase of frequency at which separator <oddcycle> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/oddcycle/includetriangles	separate triangles found as 3-cycles or repaired larger cycles Range: boolean	1
separating/oddcycle/lpliftcoef	Choose lifting candidate by coef∗lpvalue or only by coef? Range: boolean	0
separating/oddcycle/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <oddcycle> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/oddcycle/maxcutslevel	maximal number of oddcycle cuts generated in every level of the level graph Range: {0, ..., ∞}	50
separating/oddcycle/maxcutsroot	maximal number of oddcycle cuts generated per chosen variable as root of the level graph Range: {0, ..., ∞}	1
separating/oddcycle/maxnlevels	maximal number of levels in level graph Range: {0, ..., ∞}	20
separating/oddcycle/maxpernodeslevel	percentage of nodes allowed in the same level of the level graph [0-100] Range: {0, ..., 100}	100
separating/oddcycle/maxreference	minimal weight on an edge (in level graph or bipartite graph) Range: {0, ..., ∞}	0
separating/oddcycle/maxunsucessfull	number of unsuccessful calls at current node Range: {0, ..., ∞}	3
separating/oddcycle/multiplecuts	Even if a variable is already covered by a cut, still try it as start node for a cycle search? Range: boolean	0
separating/oddcycle/offsetnodeslevel	offset of nodes allowed in the same level of the level graph (additional to the percentage of levelnodes) Range: {0, ..., ∞}	10
separating/oddcycle/offsettestvars	offset of variables to try the chosen method on (additional to the percentage of testvars) Range: {0, ..., ∞}	100
separating/oddcycle/percenttestvars	percentage of variables to try the chosen method on [0-100] Range: {0, ..., 100}	0
separating/oddcycle/priority	priority of separator <oddcycle> Range: {-536870912, ..., 536870911}	-15000
separating/oddcycle/recalcliftcoef	Calculate lifting coefficient of every candidate in every step (or only if its chosen)? Range: boolean	1
separating/oddcycle/repaircycles	try to repair violated cycles with double appearance of a variable Range: boolean	1

Option	Description	Default
separating/oddcycle/scalingfactor	factor for scaling of the arc-weights Range: {1, ..., ∞}	1000
separating/oddcycle/sortrootneighbors	sort level of the root neighbors by fractionality (maxfrac) Range: boolean	1
separating/oddcycle/sortswitch	use sorted variable array (unsorted(0), maxlp(1), minlp(2), maxfrac(3), minfrac(4)) Range: {0, ..., 4}	3

5.34.4.247 separating/rapidlearning

Option	Description	Default
separating/rapidlearning/freq	frequency for calling separator <rapidlearning> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	5
Options for expert users		
separating/rapidlearning/applybdchgs	should the found global bound deductions be applied in the original SCIP? Range: boolean	1
separating/rapidlearning/applyconflicts	should the found conflicts be applied in the original SCIP? Range: boolean	1
separating/rapidlearning/applyinfervals	should the inference values be used as initialization in the original SCIP? Range: boolean	1
separating/rapidlearning/applyprimalsol	should the incumbent solution be copied to the original SCIP? Range: boolean	1
separating/rapidlearning/applysolved	should a solved status be copied to the original SCIP? Range: boolean	1
separating/rapidlearning/checkdegeneracy	should local LP degeneracy be checked? Range: boolean	1
separating/rapidlearning/checkdualbound	should the progress on the dual bound be checked? Range: boolean	0
separating/rapidlearning/checkexec	check whether rapid learning should be executed Range: boolean	1
separating/rapidlearning/checkleaves	should the ratio of leaves proven to be infeasible and exceeding the cutoff bound be checked? Range: boolean	0
separating/rapidlearning/checknsols	should the number of solutions found so far be checked? Range: boolean	1
separating/rapidlearning/checkobj	should the (local) objective function be checked? Range: boolean	0
separating/rapidlearning/contvars	should rapid learning be applied when there are continuous variables? Range: boolean	0

Option	Description	Default
separating/rapidlearning/contvarsquot	maximal portion of continuous variables to apply rapid learning Range: [0, 1]	0.3
separating/rapidlearning/copycuts	should all active cuts from cutpool be copied to constraints in subproblem? Range: boolean	1
separating/rapidlearning/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/rapidlearning/expbackoff	base for exponential increase of frequency at which separator <rapidlearning> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/rapidlearning/lpiterquot	maximal fraction of LP iterations compared to node LP iterations Range: [0, ∞]	0.2
separating/rapidlearning/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <rapidlearning> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/rapidlearning/maxcalls	maximum number of overall calls Range: {0, ..., ∞}	100
separating/rapidlearning/maxnconss	maximum problem size (constraints) for which rapid learning will be called Range: {0, ..., ∞}	10000
separating/rapidlearning/maxnodes	maximum number of nodes considered in rapid learning run Range: {0, ..., ∞}	5000
separating/rapidlearning/maxnvars	maximum problem size (variables) for which rapid learning will be called Range: {0, ..., ∞}	10000
separating/rapidlearning/mindegeneracy	minimal degeneracy threshold to allow local rapid learning Range: [0, 1]	0.7
separating/rapidlearning/mininflpratio	minimal threshold of inf/obj leaves to allow local rapid learning Range: [0, ∞]	10
separating/rapidlearning/minnodes	minimum number of nodes considered in rapid learning run Range: {0, ..., ∞}	500
separating/rapidlearning/minvarconsratio	minimal ratio of unfixed variables in relation to basis size to allow local rapid learning Range: [1, ∞]	2
separating/rapidlearning/nwaitingnodes	number of nodes that should be processed before rapid learning is executed locally based on the progress of the dualbound Range: {0, ..., ∞}	100
separating/rapidlearning/priority	priority of separator <rapidlearning> Range: {-536870912, ..., 536870911}	-1200000

Option	Description	Default
separating/rapidlearning/reducedinfer	should the inference values only be used when rapidlearning found other reductions? Range: boolean	0

5.34.4.248 separating/rlt

Option	Description	Default
separating/rlt/addtopool	if set to true, globally valid RLT cuts are added to the global cut pool Range: boolean	1
separating/rlt/detecthidden	if set to true, hidden products are detected and separated by McCormick cuts Range: boolean	0
separating/rlt/freq	frequency for calling separator <rlt> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	0
separating/rlt/hiddenrlt	whether RLT cuts (TRUE) or only McCormick inequalities (FALSE) should be added for hidden products Range: boolean	0
separating/rlt/maxncuts	maximal number of rlt-cuts that are added per round (-1: unlimited) Range: {-1, ..., ∞}	-1
separating/rlt/maxrounds	maximal number of separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	1
separating/rlt/maxroundsroot	maximal number of separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	10
separating/rlt/maxunknownterms	maximal number of unknown bilinear terms a row is still used with (-1: unlimited) Range: {-1, ..., ∞}	0
separating/rlt/maxusedvars	maximal number of variables used to compute rlt cuts (-1: unlimited) Range: {-1, ..., ∞}	100
separating/rlt/onlycontrows	if set to true, only continuous rows are used for rlt cuts Range: boolean	0
separating/rlt/onlyeqrows	if set to true, only equality rows are used for rlt cuts Range: boolean	0
separating/rlt/onlyoriginal	if set to true, only original rows and variables are used Range: boolean	1
separating/rlt/useinsubscip	if set to true, rlt is also used in sub-scips Range: boolean	0
separating/rlt/useprojection	if set to true, projected rows are checked first Range: boolean	0
Options for expert users		
separating/rlt/badscore	threshold for score of cut relative to best score to be discarded Range: [0, 1]	0.5

Option	Description	Default
separating/rlt/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/rlt/dircutoffdistweight	weight of directed cutoff distance in cut score calculation Range: [0, 1]	0
separating/rlt/efficacyweight	weight of efficacy in cut score calculation Range: [0, 1]	1
separating/rlt/expbackoff	base for exponential increase of frequency at which separator <rlt> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/rlt/goodmaxparall	maximum parallelism for good cuts Range: [0, 1]	0.1
separating/rlt/goodscore	threshold for score of cut relative to best score to be considered good, so that less strict filtering is applied Range: [0, 1]	1
separating/rlt/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <rlt> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/rlt/maxparall	maximum parallelism for non-good cuts Range: [0, 1]	0.1
separating/rlt/objparalweight	weight of objective parallelism in cut score calculation Range: [0, 1]	0
separating/rlt/priority	priority of separator <rlt> Range: {-536870912, ..., 536870911}	10

5.34.4.249 separating/strongcg

Option	Description	Default
separating/strongcg/freq	frequency for calling separator <strongcg> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
Options for expert users		
separating/strongcg/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/strongcg/expbackoff	base for exponential increase of frequency at which separator <strongcg> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/strongcg/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <strongcg> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	0
separating/strongcg/priority	priority of separator <strongcg> Range: {-536870912, ..., 536870911}	-100000

5.34.4.250 separating/zerohalf

Option	Description	Default
separating/zerohalf/dynamiccuts	should generated cuts be removed from the LP if they are no longer tight? Range: boolean	1
separating/zerohalf/freq	frequency for calling separator <zerohalf> (-1: never, 0: only in root node) Range: {-1, ..., 65534}	10
separating/zerohalf/initseed	initial seed used for random tie-breaking in cut selection Range: {0, ..., ∞}	24301
separating/zerohalf/maxcutcands	maximal number of zerohalf cuts considered per separation round Range: {0, ..., ∞}	2000
separating/zerohalf/maxrounds	maximal number of zerohalf separation rounds per node (-1: unlimited) Range: {-1, ..., ∞}	5
separating/zerohalf/maxroundsroot	maximal number of zerohalf separation rounds in the root node (-1: unlimited) Range: {-1, ..., ∞}	20
separating/zerohalf/maxsepacuts	maximal number of zerohalf cuts separated per separation round Range: {0, ..., ∞}	20
separating/zerohalf/maxsepacutsroot	maximal number of zerohalf cuts separated per separation round in the root node Range: {0, ..., ∞}	100
Options for expert users		
separating/zerohalf/badscore	threshold for score of cut relative to best score to be discarded Range: [0, 1]	0.5
separating/zerohalf/delay	should separator be delayed, if other separators found cuts? Range: boolean	0
separating/zerohalf/densityoffset	additional number of variables allowed in row on top of density Range: {0, ..., ∞}	100
separating/zerohalf/dircutoffdistweight	weight of directed cutoff distance in cut score calculation Range: [0, 1]	0
separating/zerohalf/efficacyweight	weight of efficacy in cut score calculation Range: [0, 1]	1
separating/zerohalf/expbackoff	base for exponential increase of frequency at which separator <zerohalf> is called (1: call at each multiple of frequency) Range: {1, ..., 100}	4
separating/zerohalf/goodmaxparall	maximum parallelism for good cuts Range: [0, 1]	0.1
separating/zerohalf/goodscore	threshold for score of cut relative to best score to be considered good, so that less strict filtering is applied Range: [0, 1]	1

Option	Description	Default
separating/zerohalf/maxbounddist	maximal relative distance from current node's dual bound to primal bound compared to best node's dual bound for applying separator <zerohalf> (0.0: only on current best node, 1.0: on all nodes) Range: [0, 1]	1
separating/zerohalf/maxparall	maximum parallelism for non-good cuts Range: [0, 1]	0.1
separating/zerohalf/maxrowdensity	maximal density of row to be used in aggregation Range: [0, 1]	0.05
separating/zerohalf/maxslack	maximal slack of rows to be used in aggregation Range: [0, ∞]	0
separating/zerohalf/maxslackroot	maximal slack of rows to be used in aggregation in the root node Range: [0, ∞]	0
separating/zerohalf/minviol	minimal violation to generate zerohalfcut for Range: [0, ∞]	0.1
separating/zerohalf/objparalweight	weight of objective parallelism in cut score calculation Range: [0, 1]	0
separating/zerohalf/priority	priority of separator <zerohalf> Range: {-536870912, ..., 536870911}	-6000

5.34.4.251 solvingphases

Option	Description	Default
solvingphases/enabled	should the event handler adapt the solver behavior? Range: boolean	0
solvingphases/fallback	should the event handler fall back from optimal phase? Range: boolean	0
solvingphases/feassetname	settings file for feasibility phase – precedence over emphasis settings Range: string	-
solvingphases/improvesetname	settings file for improvement phase – precedence over emphasis settings Range: string	-
solvingphases/interruptoptimal	should the event handler interrupt the solving process after optimal solution was found? Range: boolean	0
solvingphases/nodeoffset	node offset for rank-1 and estimate transitions Range: {1, ..., ∞}	50
solvingphases/optimalvalue	optimal solution value for problem Range: real	∞
solvingphases/proofsetname	settings file for proof phase – precedence over emphasis settings Range: string	-
solvingphases/testmode	should the event handler test all phase transitions? Range: boolean	0
solvingphases/transitionmethod	transition method: Possible options are 'e'stimate,'l'ogarithmic regression,'o'ptimal-value based,'r'ank-1 Range: e, l, o, r	r

Option	Description	Default
solvingphases/useemphsettings	should emphasis settings for the solving phases be used, or settings files? Range: boolean	1
solvingphases/userestart1to2	should a restart be applied between the feasibility and improvement phase? Range: boolean	0
solvingphases/userestart2to3	should a restart be applied between the improvement and the proof phase? Range: boolean	0
solvingphases/xtype	x-type for logarithmic regression - (t)ime, (n)odes, (l)p iterations Range: l, n, t	n

5.34.4.252 table/branchrules

Option	Description	Default
table/branchrules/active	is statistics table <branchrules> active Range: boolean	1

5.34.4.253 table/compression

Option	Description	Default
table/compression/active	is statistics table <compression> active Range: boolean	1

5.34.4.254 table/concurrentsolver

Option	Description	Default
table/concurrentsolver/active	is statistics table <concurrentsolver> active Range: boolean	1

5.34.4.255 table/conflict

Option	Description	Default
table/conflict/active	is statistics table <conflict> active Range: boolean	1

5.34.4.256 table/cons_nonlinear

Option	Description	Default
table/cons_nonlinear/active	is statistics table <cons_nonlinear> active Range: boolean	0

5.34.4.257 table/constiming

Option	Description	Default
table/constiming/active	is statistics table <constiming> active Range: boolean	1

5.34.4.258 table/constraint

Option	Description	Default
table/constraint/active	is statistics table <constraint> active Range: boolean	1

5.34.4.259 table/cutsel

Option	Description	Default
table/cutsel/active	is statistics table <cutsel> active Range: boolean	1

5.34.4.260 table/estim

Option	Description	Default
table/estim/active	is statistics table <estim> active Range: boolean	1

5.34.4.261 table/exprhdlr

Option	Description	Default
table/exprhdlr/active	is statistics table <exprhdlr> active Range: boolean	1

5.34.4.262 table/heuristics

Option	Description	Default
table/heuristics/active	is statistics table <heuristics> active Range: boolean	1

5.34.4.263 table/lp

Option	Description	Default
table/lp/active	is statistics table <lp> active Range: boolean	1

5.34.4.264 table/neighborhood

Option	Description	Default
table/neighborhood/active	is statistics table <neighborhood> active Range: boolean	1

5.34.4.265 table/nlhdlr

Option	Description	Default
table/nlhdlr/active	is statistics table <nlhdlr> active Range: boolean	1

5.34.4.266 table/nlhdlr_bilinear

Option	Description	Default
table/nlhdlr_bilinear/active	is statistics table <nlhdlr_bilinear> active Range: boolean	0

5.34.4.267 table/nlhdlr_quadratic

Option	Description	Default
table/nlhdlr_quadratic/active	is statistics table <nlhdlr_quadratic> active Range: boolean	0

5.34.4.268 table/nlp

Option	Description	Default
table/nlp/active	is statistics table <nlp> active Range: boolean	1

5.34.4.269 table/nlpi

Option	Description	Default
table/nlpi/active	is statistics table <nlpi> active Range: boolean	1

5.34.4.270 table/orbitalfixing

Option	Description	Default
table/orbitalfixing/active	is statistics table <orbitalfixing> active Range: boolean	1

5.34.4.271 table/origprob

Option	Description	Default
table/origprob/active	is statistics table <origprob> active Range: boolean	1

5.34.4.272 table/presolvedprob

Option	Description	Default
table/presolvedprob/active	is statistics table <presolvedprob> active Range: boolean	1

5.34.4.273 table/presolver

Option	Description	Default
table/presolver/active	is statistics table <presolver> active Range: boolean	1

5.34.4.274 table/pricer

Option	Description	Default
table/pricer/active	is statistics table <pricer> active Range: boolean	1

5.34.4.275 table/propagator

Option	Description	Default
table/propagator/active	is statistics table <propagator> active Range: boolean	1

5.34.4.276 table/relaxator

Option	Description	Default
table/relaxator/active	is statistics table <relaxator> active Range: boolean	1

5.34.4.277 table/root

Option	Description	Default
table/root/active	is statistics table <root> active Range: boolean	1

5.34.4.278 table/separator

Option	Description	Default
table/separator/active	is statistics table <separator> active Range: boolean	1

5.34.4.279 table/solution

Option	Description	Default
table/solution/active	is statistics table <solution> active Range: boolean	1

5.34.4.280 table/status

Option	Description	Default
table/status/active	is statistics table <status> active Range: boolean	1

5.34.4.281 table/timing

Option	Description	Default
table/timing/active	is statistics table <timing> active Range: boolean	1

5.34.4.282 table/tree

Option	Description	Default
table/tree/active	is statistics table <tree> active Range: boolean	1

5.34.4.283 timing

Option	Description	Default
timing/clocktype	default clock type (1: CPU user seconds, 2: wall clock time) Range: {1, ..., 2}	2
timing/enabled	is timing enabled? Range: boolean	1
timing/nlpieval	should time for evaluation in NLP solves be measured? Range: boolean	0
timing/rareclockcheck	should clock checks of solving time be performed less frequently (note: time limit could be exceeded slightly) Range: boolean	0
timing/reading	belongs reading time to solving time? Range: boolean	0
timing/statistictiming	should timing for statistic output be performed? Range: boolean	1

5.35 SHOT

SHOT (**S**upporting **H**yperplane **O**ptimization **T**oolkit) is a deterministic solver for mixed-integer nonlinear programming problems (MINLPs).

Originally, SHOT was intended for convex MINLP problems only, but now also has functionality to solve nonconvex MINLP problems as a heuristic method without providing any guarantees of global optimality. SHOT can solve certain nonconvex problem types to global optimality as well.

SHOT has mainly been developed by Andreas Lundell (Åbo Akademi University, Finland) and Jan Kronqvist (Imperial College London, UK). For more details, see [126] [122] [112] [113] [121].

SHOT supports GAMS equations that use the following intrinsic functions: abs, cos, cvPower, div, exp, log, log10, log2, pi, power, rPower, sin, sqr, sqrt, vcPower.

5.35.1 Algorithm

SHOT is based on iteratively creating a tighter polyhedral approximation of the nonlinear feasible set by generating supporting hyperplanes or cutting planes. These linearized problems are then solved with a mixed-integer linear programming (MIP) solver. GAMS/SHOT uses CPLEX, if a GAMS/CPLEX license is available, and otherwise CBC. Users with a license from Gurobi can also select Gurobi as MIP solver. If CPLEX or Gurobi is used, the subproblems can also include quadratic and bilinear nonlinearities directly.

The solution to the outer approximation problem provides a dual bound (i.e., a lower bound when solving a minimization problem) to the optimal value of the original problem if it is convex. If the problem is nonconvex, convergence to the global optimal solution cannot be guaranteed (but might be achieved for certain classes of problems, cf. [122]).

To get a primal bound (i.e., an upper bound when solving a minimization problem) on the optimal value, SHOT utilizes the following heuristics:

- Solving nonlinear programming (NLP) problems where the integer variables have been fixed to valid values. This is done by calling an NLP solver, which is either Ipopt, one of the GAMS NLP solvers, or SHOT itself.
- By checking solutions from the MIP solver's solution pool for points that fulfill also the nonlinear constraints in the original MINLP problem.
- By performing root searches.

When a termination criterion like a tolerance on the relative or absolute objective gap or a time limit is fulfilled, SHOT terminates and returns the current primal solution to GAMS. If the original problem is convex and SHOT could close the objective gap, then this is a global optimal solution to the problem. If it is nonconvex, then there is normally no guarantee that such a solution can be found. However, SHOT will always, in addition to a primal solution, return a valid dual bound on the solution in model attribute objest, unless Model.Convexity.AssumeConvex has been enabled.

5.35.2 Usage

The following statement can be used inside your GAMS program to specify using SHOT

```
Option MINLP = SHOT;     { or MIQCP }
```

The above statement should appear before the Solve statement. If SHOT was specified as the default MINLP or MIQCP solver during GAMS installation, the above statement is not necessary.

5.35.2.1 Specification of SHOT Options

GAMS/SHOT supports the GAMS parameters reslim, iterlim, nodlim, optcr, optca, cutoff, and threads.

Options can be specified by a SHOT options file. A SHOT options file consists of one option or comment per line. An asterik (∗) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (=) and any amount of white space (blanks or tabs).

A small example for a shot.opt file is:

```
Dual.CutStrategy = 1
Dual.MIP.Solver = 2
Output.Console.DualSolver.Show = true
```

It causes GAMS/SHOT to use the Extended Cutting Plane (ECP) method instead of the Extended Supporting Hyperplane (EHP) method, changes the MIP solver to CBC, and enables showing the output of the solver that computes dual bounds (typically the MIP solver).

Attention

SHOT requires options to be specified using exactly the names as specified in the documentation. That is, also casing matters.

5.35.3 List of SHOT Options

In the following, we give a detailed list of all SHOT options.

5.35.3.1 Solver output

These settings control how much and what output is shown to the user from the solver.

Option	Description	Default
Output.Debug.Path	The folder where to save the debug information Range: string	
Output.GAMS.AlternateSolutionsFile	Name of GAMS GDX file to write alternative solutions to Range: string	
Output.Console.Iteration.Detail	When should the fixed strategy be used 0: Full 1: On objective gap update 2: On objective gap update and all primal NLP calls	1
Output.Console.LogLevel	Log level for console output 0: Trace 1: Debug 2: Info 3: Warning 4: Error 5: Critical 6: Off	2
Output.Console.DualSolver.Show	Show output from dual solver on console Range: boolean	0
Output.Console.PrimalSolver.Show	Show output from primal solver on console Range: boolean	0
Output.Debug.Enable	Use debug functionality Range: boolean	0

5.35.3.2 Subsolver functionality

These settings allow for more direct control of the different subsolvers utilized in SHOT.

Option	Description	Default
Subsolver.Cplex.WorkDirectory	Directory for swap file Range: string	
Subsolver.GAMS.NLP.OptionsFilename	Options file for the NLP solver in GAMS Range: string	
Subsolver.GAMS.NLP.Solver	NLP solver to use in GAMS (auto: SHOT chooses) Range: string	auto
Subsolver.Cbc.NodeStrategy	Node strategy 0: depth 1: downdepth 2: downfewest 3: fewest 4: hybrid 5: updepth 6: upfewest	4
Subsolver.Cbc.Scaling	Whether to scale problem 0: automatic 1: dynamic 2: equilibrium 3: geometric 4: off 5: rowsonly	4
Subsolver.Cbc.Strategy	This turns on newer features 0: easy problems 1: default 2: aggressive	1
Subsolver.Cplex.FeasOptMode	Strategy to use for the feasibility repair 0: Minimize the sum of all required relaxations in first phase only 1: Minimize the sum of all required relaxations in first phase and execute second phase to find optimum among minimal relaxations 2: Minimize the number of constraints and bounds requiring relaxation in first phase only 3: Minimize the sum of squares of required relaxations in first phase only 4: Minimize the sum of squares of required relaxations in first phase and execute second phase to find optimum among minimal relaxations	0
Subsolver.Cplex.MIPEmphasis	Sets the MIP emphasis 0: Balanced 1: Feasibility 2: Optimality 3: Best bound 4: Hidden feasible	1
Subsolver.Cplex.MemoryEmphasis	Try to conserve memory when possible Range: {0, ..., 1}	0

Option	Description	Default
Subsolver.Cplex.NodeFile	Where to store the node file 0: No file 1: Compressed in memory 2: On disk 3: Compressed on disk	1
Subsolver.Cplex.NumericalEmphasis	Emphasis on numerical stability Range: {0, ..., 1}	1
Subsolver.Cplex.OptimalityTarget	Specifies how CPLEX treats nonconvex quadratics 0: Automatic 1: Searches for a globally optimal solution to a convex model 2: Searches for a solution that satisfies first-order optimality conditions, but is not necessarily globally optimal 3: Searches for a globally optimal solution to a nonconvex model	0
Subsolver.Cplex.ParallelMode	Controls how much time and memory should be used when filling the solution pool -1: Opportunistic 0: Automatic 1: Deterministic	0
Subsolver.Cplex.Probe	Sets the MIP probing level -1: No probing 0: Automatic 1: Moderate 2: Aggressive 3: Very aggressive	0
Subsolver.Cplex.SolutionPoolIntensity	Controls how much time and memory should be used when filling the solution pool 0: Automatic 1: Mild 2: Moderate 3: Aggressive 4: Very aggressive	0
Subsolver.Cplex.SolutionPoolReplace	How to replace solutions in the solution pool when full 0: Replace oldest 1: Replace worst 2: Find diverse	0
Subsolver.Gurobi.MIPFocus	MIP focus 0: Automatic 1: Feasibility 2: Optimality 3: Best bound	0
Subsolver.Gurobi.NumericFocus	MIP focus 0: Automatic 1: Mild 2: Moderate 3: Aggressive	1
Subsolver.Gurobi.PoolSearchMode	Finds extra solutions 0: No extra effort 1: Try to find solutions 2: Find n best solutions	0

Option	Description	Default
Subsolver.Gurobi.PoolSolutions	Determines how many MIP solutions are stored Range: {1, ..., 2000000000}	10
Subsolver.Gurobi.ScaleFlag	Controls model scaling -1: Automatic 0: Off 1: Mild 2: Moderate 3: Aggressive	-1
Subsolver.Ipopt.LinearSolver	Ipopt linear subsolver 0: Default 1: MA27 2: MA57 3: MA86 4: MA97 5: MUMPS	5
Subsolver.Ipopt.MaxIterations	Maximum number of iterations Range: {0, ..., ∞}	1000
Subsolver.Rootsearch.MaxIterations	Maximal root search iterations Range: {0, ..., ∞}	100
Subsolver.Rootsearch.Method	Root search method to use 0: TOMS748 1: Bisection	0
Subsolver.Cplex.SolutionPoolGap	Sets the relative gap filter on objective values in the solution pool Range: [0, 1e+75]	1e+75
Subsolver.Cplex.WorkMemory	Memory limit for when to start swapping to disk Range: [0, 1e+75]	0
Subsolver.Gurobi.Heuristics	The relative amount of time spent in MIP heuristics. Range: [0, 1]	0.05
Subsolver.Ipopt.ConstraintViolationTolerance	Constraint violation tolerance in Ipopt Range: real	1e-08
Subsolver.Ipopt.RelativeConvergenceTolerance	Relative convergence tolerance Range: real	1e-08
Subsolver.Rootsearch.ActiveConstraintTolerance	Epsilon constraint tolerance for root search Range: [0, ∞]	0
Subsolver.Rootsearch.TerminationTolerance	Epsilon lambda tolerance for root search Range: [0, ∞]	1e-16
Subsolver.SHOT.ReuseHyperplanes.Fraction	The fraction of generated hyperplanes to reuse. Range: [0, 1]	0.1
Subsolver.Cbc.AutoScale	Whether to scale objective, rhs and bounds of problem if they look odd (experimental) Range: boolean	0
Subsolver.Cbc.DeterministicParallelMode	Run Cbc with multiple threads in deterministic mode Range: boolean	0
Subsolver.Cplex.AddRelaxedLazyConstraintsAsLocal	Whether to add lazy constraints generated in relaxed points as local or global Range: boolean	0

Option	Description	Default
Subsolver.Cplex.UseGenericCallback	Use the new generic callback in the single-tree strategy Range: boolean	0
Subsolver.SHOT.ReuseHyperplanes.Use	Reuse valid generated hyperplanes in main dual model. Range: boolean	1
Subsolver.SHOT.UseFBBT	Do FBBT on NLP problem. Range: boolean	1

5.35.3.3 Dual strategy

These settings control the various functionality of the dual strategy in SHOT, i.e., the polyhedral outer approximation utilizing the ESH or ECP algorithms.

Option	Description	Default
Dual.CutStrategy	Dual cut strategy 0: ESH 1: ECP	0
Dual.ESH.InteriorPoint.CuttingPlane.IterationLimit	Iteration limit for minimax cutting plane solver Range: {1, ..., ∞}	100
Dual.ESH.InteriorPoint.CuttingPlane.IterationLimitSubsolver	Iteration limit for minimization subsolver Range: {0, ..., ∞}	100
Dual.ESH.InteriorPoint.UsePrimalSolution	Utilize primal solution as interior point 0: No 1: Add as new 2: Replace old 3: Use avarage	1
Dual.HyperplaneCuts.MaxPerIteration	Maximal number of hyperplanes to add per iteration Range: {0, ..., ∞}	200
Dual.HyperplaneCuts.ObjectiveRootSearch	When to use the objective root search 0: Always 1: IfConvex 2: Never	1
Dual.MIP.InfeasibilityRepair.IterationLimit	Max number of infeasible problems repaired without primal objective value improvement Range: {0, ..., ∞}	100
Dual.MIP.NumberOfThreads	Number of threads to use in MIP solver: 0: Automatic Range: {0, ..., 999}	GAMS threads
Dual.MIP.Presolve.Frequency	When to call the MIP presolve 0: Never 1: Once 2: Always	1
Dual.MIP.SolutionLimit.ForceOptimal.Iteration	Iterations without dual bound updates for forcing optimal MIP solution Range: {0, ..., ∞}	10000

Option	Description	Default
Dual.MIP.SolutionLimit.IncreaseIterations	Max number of iterations between MIP solution limit increases Range: {0, ..., ∞}	50
Dual.MIP.SolutionLimit.Initial	Initial MIP solution limit Range: {1, ..., ∞}	1
Dual.MIP.SolutionPool.Capacity	The maximum number of solutions in the solution pool Range: {0, ..., ∞}	100
Dual.MIP.Solver	Which MIP solver to use 0: Cplex 1: Gurobi 2: Cbc	Cplex, if licensed, otherwise Cbc
Dual.ReductionCut.MaxIterations	Max number of primal cut reduction without primal improvement Range: {0, ..., ∞}	5
Dual.Relaxation.Frequency	The frequency to solve an LP problem: 0: Disable Range: {0, ..., ∞}	0
Dual.Relaxation.IterationLimit	The max number of relaxed LP problems to solve initially Range: {0, ..., ∞}	200
Dual.Relaxation.MaxLazyConstraints	Max number of lazy constraints to add in relaxed solutions in single-tree strategy Range: {0, ..., ∞}	0
Dual.TreeStrategy	The main strategy to use 0: Multi-tree 1: Single-tree	1
Dual.ESH.InteriorPoint.CuttingPlane.ConstraintSelectionFactor	The fraction of violated constraints to generate cutting planes for Range: [0, 1]	0.25
Dual.ESH.InteriorPoint.CuttingPlane.TerminationToleranceAbs	Absolute termination tolerance between LP and linesearch objective Range: [0, ∞]	1
Dual.ESH.InteriorPoint.CuttingPlane.TerminationToleranceRel	Relative termination tolerance between LP and linesearch objective Range: [0, ∞]	1
Dual.ESH.InteriorPoint.CuttingPlane.TimeLimit	Time limit for minimax solver Range: [0, ∞]	10
Dual.ESH.InteriorPoint.MinimaxObjectiveLowerBound	Lower bound for minimax objective variable Range: [-∞, 0]	-1e+12
Dual.ESH.InteriorPoint.MinimaxObjectiveUpperBound	Upper bound for minimax objective variable Range: real	0.1

Option	Description	Default
Dual.ESH.Rootsearch.ConstraintTolerance	Constraint tolerance for when not to add individual hyperplanes Range: [0, ∞]	1e-08
Dual.HyperplaneCuts.ConstraintSelectionFactor	The fraction of violated constraints to generate supporting hyperplanes / cutting planes for Range: [0, 1]	0.5
Dual.HyperplaneCuts.MaxConstraintFactor	Rootsearch performed on constraints with values larger than this factor times the maximum value Range: [1e-06, 1]	0.1
Dual.MIP.CutOff.InitialValue	Initial cutoff value to use Range: real	GAMS cutoff
Dual.MIP.CutOff.Tolerance	An extra tolerance for the objective cutoff value (to prevent infeasible subproblems) Range: real	1e-05
Dual.MIP.InfeasibilityRepair.TimeLimit	Time limit when reparing infeasible problem Range: [0, ∞]	10
Dual.MIP.NodeLimit	Node limit to use for MIP solver in single-tree strategy Range: [0, ∞]	GAMS nodlim
Dual.MIP.OptimalityTolerance	The reduced-cost tolerance for optimality in the MIP solver Range: [1e-09, 0.01]	1e-06
Dual.MIP.SolutionLimit.ForceOptimal.Time	Time (s) without dual bound updates for forcing optimal MIP solution Range: [0, ∞]	1000
Dual.MIP.SolutionLimit.UpdateTolerance	The constraint tolerance for when to update MIP solution limit Range: [0, ∞]	0.001
Dual.ReductionCut.ReductionFactor	The factor used to reduce the cutoff value Range: [0, 1]	0.001
Dual.Relaxation.TerminationTolerance	Time limit (s) when solving LP problems initially Range: real	0.5
Dual.Relaxation.TimeLimit	Time limit (s) when solving LP problems initially Range: [0, ∞]	30
Dual.ESH.InteriorPoint.CuttingPlane.Reuse	Reuse valid cutting planes in main dual model Range: boolean	0
Dual.ESH.Rootsearch.UniqueConstraints	Allow only one hyperplane per constraint per iteration Range: boolean	0

Option	Description	Default
Dual.ESH.Rootsearch.UseMaxFunction	Perform rootsearch on max function, otherwise on individual constraints Range: boolean	0
Dual.HyperplaneCuts.Delay	Add hyperplane cuts to model only after optimal MIP solution Range: boolean	1
Dual.HyperplaneCuts.SaveHyperplanePoints	Whether to save the points in the generated hyperplanes list Range: boolean	0
Dual.HyperplaneCuts.UseIntegerCuts	Add integer cuts for infeasible integer-combinations for binary problems Range: boolean	0
Dual.MIP.CutOff.UseInitialValue	Use the initial cutoff value Range: boolean	1, if cutoff is set
Dual.MIP.InfeasibilityRepair.IntegerCuts	Allow feasibility repair of integer cuts Range: boolean	1
Dual.MIP.InfeasibilityRepair.Use	Enable the infeasibility repair strategy for nonconvex problems Range: boolean	1
Dual.MIP.Presolve.RemoveRedundantConstraints	Remove redundant constraints (as determined by presolve) Range: boolean	0
Dual.MIP.Presolve.UpdateObtainedBounds	Update bounds (from presolve) to the MIP model Range: boolean	1
Dual.MIP.UpdateObjectiveBounds	Update nonlinear objective variable bounds to primal/dual bounds Range: boolean	0
Dual.ReductionCut.Use	Enable the dual reduction cut strategy for nonconvex problems Range: boolean	1
Dual.Relaxation.Use	Initially solve continuous dual relaxations Range: boolean	1

5.35.3.4 Optimization model

These settings control various aspects of SHOT's representation for and handling of the provided optimization model.

Option	Description	Default
Model.BoundTightening.FeasibilityBased.MaxIterations	Maximal number of bound tightening iterations Range: {0, ..., ∞}	5
Model.BoundTightening.InitialPOA.CutStrategy	Dual cut strategy 0: ESH 1: ECP	1
Model.BoundTightening.InitialPOA.IterationLimit	Iteration limit for POA Range: {0, ..., ∞}	50

Option	Description	Default
Model.BoundTightening.InitialPOA.StagnationIterationLimit	Limit for iterations without significant progress Range: {0, ..., ∞}	5
Model.Reformulation.Bilinear.IntegerFormulation	Reformulate integer bilinear terms 0: No 1: No if nonconvex quadratic terms allowed by MIP solver 2: Yes	1
Model.Reformulation.Bilinear.IntegerFormulation.MaxDomain	Do not reformulate integer variables in bilinear terms which can assume more than this number of discrete values Range: {2, ..., ∞}	100
Model.Reformulation.Constraint.PartitionNonlinearTerms	When to partition nonlinear sums in constraints 0: Always 1: If result is convex 2: Never	1
Model.Reformulation.Constraint.PartitionQuadraticTerms	When to partition quadratic sums in constraints 0: Always 1: If result is convex 2: Never	1
Model.Reformulation.Monomials.Formulation	How to reformulate binary monomials 0: None 1: Simple 2: Costa and Liberti	1
Model.Reformulation.ObjectiveFunction.PartitionNonlinearTerms	When to partition nonlinear sums in objective function 0: Always 1: If result is convex 2: Never	1
Model.Reformulation.ObjectiveFunction.PartitionQuadraticTerms	When to partition quadratic sums in objective function 0: Always 1: If result is convex 2: Never	1
Model.Reformulation.Quadratics.EigenValueDecomposition.Formulation	Which formulation to use in eigenvalue decomposition 0: Term coefficient is included in reformulation 1: Term coefficient remains	0
Model.Reformulation.Quadratics.ExtractStrategy	How to extract quadratic terms from nonlinear expressions 0: Do not extract 1: Extract to same objective or constraint 2: Extract to quadratic equality constraint if nonconvex 3: Extract to quadratic equality constraint even if convex	1
Model.Reformulation.Quadratics.Strategy	How to treat quadratic functions 0: All nonlinear 1: Use quadratic objective 2: Use convex quadratic objective and constraints 3: Use nonconvex quadratic objective and constraints	2

Option	Description	Default
Model.BoundTightening.FeasibilityBased.TimeLimit	Time limit for bound tightening Range: [0, ∞]	2
Model.BoundTightening.InitialPOA.ConstraintTolerance	Constraint termination tolerance Range: real	0.1
Model.BoundTightening.InitialPOA.ObjectiveConstraintTolerance	Objective constraint termination tolerance Range: real	0.001
Model.BoundTightening.InitialPOA.ObjectiveGapAbsolute	Absolute objective gap termination level Range: real	0.1
Model.BoundTightening.InitialPOA.ObjectiveGapRelative	Relative objective gap termination level Range: real	0.1
Model.BoundTightening.InitialPOA.StagnationConstraintTolerance	Tolerance factor for when no progress is made Range: real	0.01
Model.BoundTightening.InitialPOA.TimeLimit	Time limit for initial POA Range: real	5
Model.Convexity.Quadratics.EigenValueTolerance	Convexity tolerance for the eigenvalues of the Hessian matrix for quadratic terms Range: [0, ∞]	1e-05
Model.Reformulation.Quadratics.EigenValueDecomposition.Tolerance	Variables with eigenvalues smaller than this value will be ignored Range: [0, ∞]	1e-06
Model.Variables.Continuous.MaximumUpperBound	Maximum upper bound for continuous variables Range: real	1e+50
Model.Variables.Continuous.MinimumLowerBound	Minimum lower bound for continuous variables Range: real	-1e+50
Model.Variables.Integer.MaximumUpperBound	Maximum upper bound for integer variables Range: real	2e+09
Model.Variables.Integer.MinimumLowerBound	Minimum lower bound for integer variables Range: real	-2e+09
Model.Variables.NonlinearObjectiveVariable.Bound	Max absolute bound for the auxiliary nonlinear objective variable Range: real	1e+12
Model.BoundTightening.FeasibilityBased.Use	Peform feasibility-based bound tightening Range: boolean	1
Model.BoundTightening.FeasibilityBased.UseNonlinear	Peform feasibility-based bound tightening on nonlinear expressions Range: boolean	1
Model.BoundTightening.InitialPOA.Use	Create an initial polyhedral outer approximation Range: boolean	0
Model.Convexity.AssumeConvex	Assume that the problem is convex Range: boolean	0
Model.Reformulation.Bilinear.AddConvexEnvelope	Add convex envelopes (subject to original bounds) to bilinear terms Range: boolean	0
Model.Reformulation.Monomials.Extract	Extract monomial terms from nonlinear expressions Range: boolean	1
Model.Reformulation.ObjectiveFunction.Epigraph.Use	Reformulates a nonlinear objective as an auxiliary constraint Range: boolean	0

Option	Description	Default
Model.Reformulation.Quadratics.EigenValueDecomposition.Method	Whether to use the eigen value decomposition of convex quadratic functions Range: boolean	0
Model.Reformulation.Quadratics.EigenValueDecomposition.Use	Whether to use the eigenvalue decomposition of convex quadratic functions Range: boolean	0
Model.Reformulation.Signomials.Extract	Extract signomial terms from nonlinear expressions Range: boolean	1

5.35.3.5 Modeling system

These settings control functionality used in the interfaces to different modeling environments.

Option	Description	Default
ModelingSystem.GAMS.QExtractAlg	Extraction algorithm for quadratic equations in GAMS interface 0: automatic 1: threepass 2: doubleforward 3: concurrent	0

5.35.3.6 Primal heuristics

These settings control the primal heuristics used in SHOT.

Option	Description	Default
Primal.FixedInteger.CallStrategy	When should the fixed strategy be used 0: Use each iteration 1: Based on iteration or time 2: Based on iteration or time, and for all feasible MIP solutions	2
Primal.FixedInteger.Frequency.Iteration	Max number of iterations between calls Range: {0, ..., ∞}	10
Primal.FixedInteger.IterationLimit	Max number of iterations per call Range: {0, ..., ∞}	10000000
Primal.FixedInteger.Solver	NLP solver to use 0: Ipopt 1: GAMS 2: SHOT	1
Primal.FixedInteger.Source	Source of fixed MIP solution point 0: All 1: First 2: All feasible 3: First and all feasible 4: With smallest constraint deviation	3
Primal.FixedInteger.SourceProblem	Which problem formulation to use for NLP problem 0: Original problem 1: Reformulated problem 2: Both	0

Option	Description	Default
Primal.FixedInteger.DualPointGap.Relative	If the objective gap between the MIP point and dual solution is less than this the fixed strategy is activated Range: [0, ∞]	0.001
Primal.FixedInteger.Frequency.Time	Max duration (s) between calls Range: [0, ∞]	5
Primal.FixedInteger.TimeLimit	Time limit (s) per NLP problem Range: [0, ∞]	10
Primal.Tolerance.Integer	Integer tolerance for accepting primal solutions Range: real	1e-05
Primal.Tolerance.LinearConstraint	Linear constraint tolerance for accepting primal solutions Range: real	1e-06
Primal.Tolerance.NonlinearConstraint	Nonlinear constraint tolerance for accepting primal solutions Range: real	1e-05
Primal.FixedInteger.CreateInfeasibilityCut	Create a cut from an infeasible solution point Range: boolean	0
Primal.FixedInteger.Frequency.Dynamic	Dynamically update the call frequency based on success Range: boolean	1
Primal.FixedInteger.OnlyUniqueIntegerCombinations	Whether to resolve with the same integer combination, e.g. for nonconvex problems with different continuous variable starting points Range: boolean	1
Primal.FixedInteger.Use	Use the fixed integer primal strategy Range: boolean	1
Primal.FixedInteger.Warmstart	Warm start the NLP solver Range: boolean	1
Primal.Rootsearch.Use	Use a rootsearch to find primal solutions Range: boolean	1
Primal.Tolerance.TrustLinearConstraintValues	Trust that subsolvers (NLP, MIP) give primal solutions that respect linear constraints Range: boolean	1

5.35.3.7 Termination

These settings control when SHOT will terminate the solution process.

Option	Description	Default
Termination.DualStagnation.IterationLimit	Max number of iterations without significant dual objective value improvement Range: {0, ..., ∞}	∞
Termination.IterationLimit	Iteration limit for main strategy Range: {1, ..., ∞}	GAMS iterlim
Termination.PrimalStagnation.IterationLimit	Max number of iterations without significant primal objective value improvement Range: {0, ..., ∞}	50

Option	Description	Default
Termination.ConstraintTolerance	Termination tolerance for nonlinear constraints Range: [0, ∞]	1e-08
Termination.DualStagnation.ConstraintTolerance	Min absolute difference between max nonlinear constraint errors in subsequent iterations for termination Range: [0, ∞]	1e-06
Termination.ObjectiveConstraintTolerance	Termination tolerance for the nonlinear objective constraint Range: [0, ∞]	1e-08
Termination.ObjectiveGap.Absolute	Absolute gap termination tolerance for objective function Range: [0, ∞]	GAMS optca
Termination.ObjectiveGap.Relative	Relative gap termination tolerance for objective function Range: [0, ∞]	GAMS optcr
Termination.TimeLimit	Time limit (s) for solver Range: [0, ∞]	GAMS reslim

5.35.3.8 Strategy

Overall strategy parameters used in SHOT.

Option	Description	Default
Strategy.UseRecommendedSettings	Modifies some settings to their recommended values based on the strategy Range: boolean	1

5.36 SNOPT

Author

Philip E. Gill; Department of Mathematics, University of California, San Diego, La Jolla, CA

Walter Murray, Michael A. Saunders; Department of EESOR, Stanford University, Stanford, CA

5.36.1 Introduction

This section describes the GAMS interface to the general-purpose NLP solver SNOPT, (Sparse Nonlinear Optimizer) which implements a sequential quadratic programming (SQP) method for solving constrained optimization problems with smooth nonlinear functions in the objective and constraints. The optimization problem is assumed to be stated in the form

$$\begin{aligned} \text{NP}: \quad & \text{minimize or maximize } f_0(x) \\ & f(x) \sim b_1 \\ \text{subject to} \quad & A_L x \sim b_2 \\ & l \leq x \leq u, \end{aligned} \tag{1}$$

where $x \in \Re^n$, $f_0(x)$ is a linear or nonlinear smooth objective function, l and u are constant lower and upper bounds, $f(x)$ is a set of nonlinear constraint functions, A_L is a sparse matrix, $\sim$ is a vector of relational operators ($\leq$, $\geq$ or $=$), and b_1 and b_2 are right-hand side constants. $f(x) \sim b_1$s are the nonlinear constraints of the model and $A_L x \sim b_2$ form the linear constraints.

The gradients of f_0 and f_i are automatically provided by GAMS, using its automatic differentiation engine.

The bounds may include special values `-INF` or `+INF` to indicate $l_j = -\infty$ or $u_j = +\infty$ for appropriate j. Free variables have both bounds infinite and fixed variables have $l_j = u_j$.

SNOPT uses a sequential quadratic programming (SQP) algorithm that obtains search directions from a sequence of quadratic programming subproblems. Each QP subproblem minimizes a quadratic model of a certain Lagrangian function subject to a linearization of the constraints. An augmented Lagrangian merit function is reduced along each search direction to ensure convergence from any starting point.

SNOPT is most efficient if only some of the variables enter nonlinearly, or if the number of active constraints (including simple bounds) is nearly as large as the number of variables. SNOPT requires relatively few evaluations of the problem functions.

5.36.1.1 Problem Types

If the nonlinear functions are absent, the problem is a *linear program* (LP) and SNOPT applies the primal simplex method [41] . Sparse basis factors are maintained by LUSOL [82] as in MINOS [141] .

If only the objective is nonlinear, the problem is *linearly constrained* (LC) and tends to solve more easily than the general case with nonlinear constraints (NC). Note that GAMS models have an objective variable instead of an objective function. The GAMS/SNOPT link will try to substitute out the objective variable and reformulate the model such that SNOPT will see a true objective function.

For both linearly and nonlinearly constrained problems SNOPT applies a sparse sequential quadratic programming (SQP) method [85] using limited-memory quasi-Newton approximations to the Hessian of the Lagrangian. The merit function for steplength control is an augmented Lagrangian, as in the dense SQP solver NPSOL [81] [84] .

In general, SNOPT requires less matrix computation than NPSOL and fewer evaluations of the functions than the nonlinear algorithms in MINOS [139] [140] . It is suitable for nonlinear problems with thousands of constraints and variables, and is most efficient if only some of the variables enter nonlinearly, or there are relatively few degrees of freedom at a solution (i.e., many constraints are active). However, unlike previous versions of SNOPT, there is no limit on the number of degrees of freedom.

5.36.1.2 Selecting the SNOPT Solver

If SNOPT is not specified as the default solver for the desired model type (e.g. NLP), then the following statement can be used in your GAMS model to select SNOPT:

```
option nlp=SNOPT;
```

The option statement should appear before the `solve` statement. To be complete, we mention that the solver can be also specified on the command line, as in:

```
> gams camcge nlp=snopt
```

This will override the global default, but if an algorithm option has been specified inside the model, then that specification takes precedence.

If the model contains non-smooth functions like abs (x), or $\max(x, y)$ you can try to get it solved by SNOPT using

```
option dnlp=SNOPT;
```

These models have discontinuous derivatives however, and SNOPT was not designed for solving such models. Discontinuities in the gradients can sometimes be tolerated if they appear away from an optimum.

5.36.2 Description of the method

Here we briefly describe the main features of the SQP algorithm used in SNOPT and introduce some terminology. The SQP algorithm is fully described by by Gill, Murray and Saunders [86] .

5.36.2.1 Objective function reconstruction

The first step GAMS/SNOPT performs is to try to reconstruct the objective function. In GAMS, optimization models minimize or maximize an objective variable. SNOPT however works with an objective function. One way of dealing with this is to add a dummy linear function with just the objective variable. Consider the following GAMS fragment:

```
obj.. z =e= sum(i, sqr[r(i)]);

model m /all/;
solve m using nlp minimizing z;
```

This can be cast in form (1) by saying minimize z subject to $z = \sum_i r_i^2$ and the other constraints in the model. Although simple, this approach is not always preferable. Especially when all constraints are linear it is important to minimize the nonlinear expression $\sum_i r_i^2$ directly. This can be achieved by a simple reformulation: z can be substituted out. The substitution mechanism carries out the formulation if all of the following conditions hold:

- the objective variable z is a free continuous variable (no bounds are defined on z),
- z appears linearly in the objective function,
- the objective function is formulated as an equality constraint,
- z is only present in the objective function and not in other constraints.

For many models it is very important that the nonlinear objective function be used by SNOPT. For instance the model `chem.gms` from the model library solves in 16 iterations. When we add the bound

```
energy.lo = 0;
```

on the objective variable `energy` and thus preventing it from being substituted out, SNOPT will not be able to find a feasible point for the given starting point.

This reformulation mechanism has been extended for substitutions along the diagonal. For example, the GAMS model

```
variables x,y,z;
equations e1,e2;
e1..z =e= y;
e2..y =e= sqr(1+x);
model m /all/;
option nlp=snopt;
solve m using nlp minimizing z;
```

will be reformulated as an *unconstrained* optimization problem

$$\min f(x) = (1+x)^2.$$

5.36.2.2 Constraints and slack variables

Problem (1) contains n variables in x. Let m be the number of components of $f(x)$ and $A_L x$ combined. The upper and lower bounds on those terms define the general constraints of the problem. SNOPT converts the general constraints to equalities by introducing a set of slack variables $s = (s_1, s_2, ..., s_m)^T$. For example, the linear constraint $5 \leq 2x_1 + 3x_2 \leq +\infty$ is replaced by $2x_1 + 3x_2 - s_1 = 0$ together with the bounded slack $5 \leq s_1 \leq +\infty$. Problem (1) can be written in the equivalent form

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & \begin{pmatrix} f(x) \\ A_L x \end{pmatrix} - s = 0, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u. \end{array}$$

where a maximization problem is cast into a minimization by multiplying the objective function by -1.

The linear and nonlinear general constraints become equalities of the form $f(x) - s_N = 0$ and $A_L x - s_L = 0$, where s_L and s_N are known as the *linear* and *nonlinear* slacks.

5.36.2.3 Major iterations

The basic structure of SNOPT's solution algorithm involves *major* and *minor* iterations. The major iterations generate a sequence of iterates (x_k) that satisfy the linear constraints and converge to a point that satisfies the first-order conditions for optimality. At each iterate $\{x_k\}$ a QP subproblem is used to generate a search direction towards the next iterate $\{x_{k+1}\}$. The constraints of the subproblem are formed from the linear constraints $A_L x - s_L = 0$ and the nonlinear constraint linearization

$$f(x_k) + f'(x_k)(x - x_k) - s_N = 0,$$

where $f'(x_k)$ denotes the *Jacobian*: a matrix whose rows are the first derivatives of $f(x)$ evaluated at x_k. The QP constraints therefore comprise the m linear constraints

$$\begin{array}{rrrl} f'(x_k)x & -s_N & = & -f(x_k) + f'(x_k)x_k, \\ A_L x & -s_L & = & 0, \end{array}$$

where x and s are bounded by l and u as before. If the $m \times n$ matrix A and m-vector b are defined as

$$A = \begin{pmatrix} f'(x_k) \\ A_L \end{pmatrix} \quad \text{and} \quad b = \begin{pmatrix} -f(x_k) + f'(x_k)x_k \\ 0 \end{pmatrix},$$

then the QP subproblem can be written as

$$\begin{array}{lll} \text{QP}_k : & \min\limits_{x,s} & q(x, x_k) = g_k^T(x - x_k) + \dfrac{1}{2}(x - x_k)^T H_k (x - x_k) \\ & \text{subject to} & Ax - s = b, \quad l \leq \begin{pmatrix} x \\ s \end{pmatrix} \leq u, \end{array} \tag{2}$$

where $q(x, x_k)$ is a quadratic approximation to a modified Lagrangian function [85] . The matrix H_k is a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration. If some of the variables enter the Lagrangian linearly the Hessian will have some zero rows and columns. If the nonlinear variables appear first, then only the leading n_1 rows and columns of the Hessian need be approximated, where n_1 is the number of nonlinear variables.

5.36.2.4 Minor iterations

Solving the QP subproblem is itself an iterative procedure. Here, the iterations of the QP solver SQOPT [87] form the *minor* iterations of the SQP method.

SQOPT uses a reduced-Hessian active-set method implemented as a reduced-gradient method similar to that in MINOS [139] .

At each minor iteration, the constraints $Ax - s = b$ are partitioned into the form

$$Bx_B + Sx_S + Nx_N = b,$$

where the *basis matrix* B is square and nonsingular and the matrices S, N are the remaining columns of $(A\ -I)$. The vectors x_B, x_S, x_N are the associated basic, superbasic, and nonbasic components of the variables (x, s).

The term *active-set method* arises because the nonbasic variables x_N are temporarily frozen at their upper or lower bounds, and their bounds are considered to be active. Since the general constraints are satisfied also, the set of active constraints takes the form

$$\begin{pmatrix} B & S & N \\ & & I \end{pmatrix} \begin{pmatrix} x_B \\ x_S \\ x_N \end{pmatrix} = \begin{pmatrix} b \\ x_N \end{pmatrix},$$

where x_N represents the current values of the nonbasic variables. (In practice, nonbasic variables are sometimes frozen at values strictly between their bounds.) The reduced-gradient method chooses to move the superbasic variables in a direction that will improve the objective function. The basic variables "tag along" to keep $Ax - s = b$ satisfied, and the nonbasic variables remain unaltered until one of them is chosen to become superbasic.

At a nonoptimal feasible point (x, s) we seek a search direction p such that $(x, s) + p$ remains on the set of active constraints yet improves the QP objective. If the new point is to be feasible, we must have $Bp_B + Sp_S + Np_N = 0$ and $p_N = 0$. Once p_S is specified, p_B is uniquely determined from the system $Bp_B = -Sp_S$. It follows that the superbasic variables may be regarded as independent variables that are free to move in any desired direction. The number of superbasic variables (n_S say) therefore indicates the number of degrees of freedom remaining after the constraints have been satisfied. In broad terms, n_S is a measure of how nonlinear the problem is. In particular, n_S need not be more than one for linear problems.

5.36.2.5 The reduced Hessian and reduced gradient

The dependence of p on p_S may be expressed compactly as $p = Zp_S$, where Z is a matrix that spans the null space of the active constraints:

$$Z = P \begin{pmatrix} -B^{-1}S \\ I \\ 0 \end{pmatrix} \tag{3}$$

where P permutes the columns of $(A\ -I)$ into the order $(B\ S\ N)$. Minimizing $q(x, x_k)$ with respect to p_S now involves a quadratic function of p_S:

$$g^T Zp_S + \frac{1}{2} p_S^T Z^T HZp_S, \tag{4}$$

where g and H are expanded forms of g_k and H_k defined for all variables (x, s). This is a quadratic with Hessian $Z^T HZ$ (the reduced Hessian) and constant vector $Z^T g$ (the reduced gradient). If the reduced Hessian is nonsingular, p_S is computed from the system

$$Z^T HZp_S = -Z^T g. \tag{5}$$

The matrix Z is used only as an operator, i.e., it is not stored explicitly. Products of the form Zv or $Z^T g$ are obtained by solving with B or B^T . The package LUSOL [82] is used to maintain sparse LU factors of B as the BSN partition changes. From the definition of Z, we see that the reduced gradient can be computed from

$$B^T\pi = g_B, \;\; Z^T g = g_S - S^T\pi,$$

where π is an estimate of the *dual variables* associated with the m equality constraints $Ax - s = b$, and g_B is the basic part of g.

By analogy with the elements of $Z^T g$, we define a vector of reduced gradients (or reduced costs) for all variables in (x, s):

$$d = g - \begin{pmatrix} A^T \\ -I \end{pmatrix}\pi, \;\; \text{so that } d_S = Z^T g.$$

At a feasible point, the reduced gradients for the slacks s are the dual variables π.

The optimality conditions for subproblem QP $_k$ (2) may be written in terms of d. The current point is optimal if $d_j \geq 0$ for all nonbasic variables at their lower bounds, $d_j \leq 0$ for all nonbasic variables at their upper bounds, and $d_j = 0$ for all superbasic variables ($d_S = 0$). In practice, SNOPT requests an *approximate* QP solution $(\widehat{x}_k, \widehat{s}_k, \widehat{\pi}_k)$ with slightly relaxed conditions on d_j.

If $d_S = 0$, no improvement can be made with the current BSN partition, and a nonbasic variable with non-optimal reduced gradient is selected to be added to S. The iteration is then repeated with n_S increased by one. At all stages, if the step $(x, s) + p$ would cause a basic or superbasic variable to violate one of its bounds, a shorter step $(x, s) + \alpha p$ is taken, one of the variables is made nonbasic, and n_S is decreased by one.

The process of computing and testing reduced gradients d_N is known as *pricing* (a term introduced in the context of the simplex method for linear programming). Pricing the jth variable means computing $d_j = g_j - a_j^T\pi$, where a_j is the jth column of $(A - I)$. If there are significantly more variables than general constraints (i.e., $n \gg m$), pricing can be computationally expensive. In this case, a strategy known as *partial pricing* can be used to compute and test only a subset of d_N.

Solving the reduced Hessian system (5) is sometimes expensive. With the option `QPSolver Cholesky`, an upper-triangular matrix R is maintained satisfying $R^T R = Z^T H Z$. Normally, R is computed from $Z^T H Z$ at the start of phase 2 and is then updated as the BSN sets change. For efficiency the dimension of R should not be excessive (say, $n_S \leq 1000$). This is guaranteed if the number of nonlinear variables is "moderate". Other `QPSolver` options are available for problems with many degrees of freedom.

5.36.2.6 The merit function

After a QP subproblem has been solved, new estimates of the NLP solution are computed using a linesearch on the augmented Lagrangian merit function

$$\mathcal{M}(x, s, \pi) = f(x) - \pi^T\big(F(x) - s_N\big) + \frac{1}{2}\big(F(x) - s_N\big)^T D\big(F(x) - s_N\big), \tag{6}$$

where D is a diagonal matrix of penalty parameters. If (x_k, s_k, π_k) denotes the current solution estimate and $(\widehat{x}_k, \widehat{s}_k, \widehat{\pi}_k)$ denotes the optimal QP solution, the linesearch determines a step α_k ($0 < \alpha_k \leq 1$) such that the new point

$$\begin{pmatrix} x_{k+1} \\ s_{k+1} \\ \pi_{k+1} \end{pmatrix} = \begin{pmatrix} x_k \\ s_k \\ \pi_k \end{pmatrix} + \alpha_k \begin{pmatrix} \widehat{x}_k - x_k \\ \widehat{s}_k - s_k \\ \widehat{\pi}_k - \pi_k \end{pmatrix} \tag{7}$$

gives a *sufficient decrease* in the merit function. When necessary, the penalties in D are increased by the minimum-norm perturbation that ensures descent for $\mathcal{M}$ [84] . As in NPSOL, s_N is adjusted to minimize the merit function as a function of s prior to the solution of the QP subproblem. For more details, see [81] [54] .

5.36.2.7 Treatment of constraint infeasibilities

SNOPT makes explicit allowance for infeasible constraints. Infeasible *linear* constraints are detected first by solving a problem of the form

$$\begin{aligned} \text{FLP}: \qquad & \text{minimize} \quad e^T(v+w) \\ & \text{subject to} \quad \ell \le \begin{pmatrix} x \\ A_L x - v + w \end{pmatrix} \le u,\ v \ge 0,\ w \ge 0, \end{aligned}$$

where e is a vector of ones. This is equivalent to minimizing the sum of the general linear constraint violations subject to the simple bounds. (In the linear programming literature, the approach is often called *elastic programming*. We also describe it as minimizing the ℓ_1 norm of the infeasibilities.)

If the linear constraints are infeasible ($v \neq 0$ or $w \neq 0$), SNOPT terminates without computing the nonlinear functions.

If the linear constraints are feasible, all subsequent iterates satisfy the linear constraints. (Such a strategy allows linear constraints to be used to define a region in which the functions can be safely evaluated.) SNOPT proceeds to solve NP (1) as given, using search directions obtained from a sequence of quadratic programming subproblems (2).

If a QP subproblem proves to be infeasible or unbounded (or if the dual variables π for the nonlinear constraints become large), SNOPT enters "elastic" mode and solves the problem

$$\begin{aligned} \text{NP}(\gamma): \qquad & \text{minimize} \quad f_0(x) + \gamma e^T(v+w) \\ & \text{subject to} \quad \ell \le \begin{pmatrix} x \\ f(x) - v + w \\ A_L x \end{pmatrix} \le u,\ v \ge 0,\ w \ge 0, \end{aligned}$$

where γ is a nonnegative parameter (the *elastic weight*), and $f(x) + \gamma e^T(v+w)$ is called a *composite objective*. If γ is sufficiently large, this is equivalent to minimizing the sum of the nonlinear constraint violations subject to the linear constraints and bounds. A similar ℓ_1 formulation of NP is fundamental to the S ℓ_1QP algorithm of Fletcher [64] . See also Conn [38] .

The initial value of γ is controlled by the optional parameter elastic weight.

5.36.3 Starting points and advanced bases

A good starting point may be essential for solving nonlinear models. We show how such a starting point can be specified in a GAMS environment, and how SNOPT will use this information.

A related issue is the use of "restart" information in case a number of related models are solved in a row. Starting from an optimal point of a previous solve statement is in such situations often beneficial. In a GAMS environment this means reusing primal and dual information, which is stored in the `.L` and `.M` fields of variables and equations.

5.36.3.1 Starting points

To specify a starting point for SNOPT use the `.L` level values in GAMS. For example, to set all variables $x_{i,j} := 1$ use `x.l(i,j)=1;`. The default values for level values are zero.

Setting a good starting point can be crucial for getting good results. As an (artificial) example consider the problem where we want to find the smallest circle that contains a number of points (x_i, y_i):

$$\begin{array}{lll} \text{Example :} & \text{minimize} & r \\ & \text{subject to} & (x_i - a)^2 + (y_i - b)^2 \leq r^2,\ r \geq 0. \end{array}$$

This problem can be modeled in GAMS as follows.

```
set i points /p1*p10/;

parameters
    x(i)    x coordinates,
    y(i)    y coordinates;

* fill with random data
x(i) = uniform(1,10);
y(i) = uniform(1,10);

variables
    a       x coordinate of center of circle
    b       y coordinate of center of circle
    r       radius;

equations
    e(i)    points must be inside circle;


e(i).. sqr(x(i)-a) + sqr(y(i)-b) =l= sqr(r);

r.lo = 0;

model m /all/;
option nlp=snopt;
solve m using nlp minimizing r;
```

Without help, SNOPT will not be able to find an optimal solution. The problem will be declared infeasible. In this case, providing a good starting point is very easy. If we define

$$\begin{aligned} x_{\min} &= \min_i x_i, \\ y_{\min} &= \min_i y_i, \\ x_{\max} &= \max_i x_i, \\ y_{\max} &= \max_i y_i, \end{aligned}$$

then good estimates are

$$\begin{aligned} a &= (x_{\min} + x_{\max})/2, \\ b &= (y_{\min} + y_{\max})/2, \\ r &= \sqrt{(a - x_{\min})^2 + (b - y_{\min})^2}. \end{aligned}$$

Thus we include in our model:

```
parameters xmin,ymin,xmax,ymax;
xmin = smin(i, x(i));
ymin = smin(i, x(i));
xmax = smax(i, x(i));
ymax = smax(i, y(i));

* set starting point
a.l = (xmin+xmax)/2;
b.l = (ymin+ymax)/2;
r.l = sqrt( sqr(a.l-xmin) + sqr(b.l-ymin) );
```

and now the model solves very easily.

Level values can also be set implicitly as a result of assigning bounds, since GAMS will project variable levels onto their bounds as part of executing a solve statement. For example, when a variable is bounded away from zero by a statement like `Y.LO = 1;` and `Y` is at its default level of zero, the `SOLVE` statement will set the level `Y.L` to 1.

Note: another way to formulate the model would be to minimize r^2 instead of r. This allows SNOPT to solve the problem even with the default starting point.

5.36.3.2 Advanced basis

GAMS automatically passes on level values and basis information from one solve to the next. Thus, when we have two solve statements in a row, with just a few changes in between SNOPT will typically need very few iterations to find an optimal solution in the second solve. For instance, when we add a second solve to the `fawley.gms` model from the model library:

```
Model exxon /all/;
...
Solve exxon maximizing profit using lp;
Solve exxon maximizing profit using lp;
```

we observe the following iteration counts:

```
               S O L V E      S U M M A R Y

     MODEL   exxon               OBJECTIVE  profit
     TYPE    LP                  DIRECTION  MAXIMIZE
     SOLVER  SNOPT               FROM LINE  278

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL
**** OBJECTIVE VALUE             2899.2528

 RESOURCE USAGE, LIMIT          0.016      1000.000
 ITERATION COUNT, LIMIT        24         10000

.....

               S O L V E      S U M M A R Y

     MODEL   exxon               OBJECTIVE  profit
     TYPE    LP                  DIRECTION  MAXIMIZE
```

```
     SOLVER  SNOPT               FROM LINE  279

**** SOLVER STATUS     1 NORMAL COMPLETION
**** MODEL STATUS      1 OPTIMAL
**** OBJECTIVE VALUE              2899.2528

 RESOURCE USAGE, LIMIT          0.000      1000.000
 ITERATION COUNT, LIMIT         0         10000
```

The first `solve` takes 24 iterations, while the second `solve` needs exactly zero iterations.

Basis information is passed on using the marginals of the variables and equations. In general the rule is:

- `X.M = 0`: basic
- `X.M` $\neq 0$: nonbasic if level value is at bound, superbasic otherwise

A marginal value of `EPS` means that the numerical value of the marginal is zero, but that the status is nonbasic or superbasic. The user can specify a basis by assigning zero or nonzero values to the `.M` values. It is further noted that if too many `.M` values are zero, the basis is rejected. This happens for instance when two subsequent models are too different. This decision is made based on the value of the GAMS bratio option.

5.36.4 GAMS Options

The usual GAMS options (e.g. reslim, sysout) can be used to control GAMS/SNOPT. For more details, see section Controlling a Solver via GAMS Options. We highlight some of the details of this usage below for cases of special interest.

iterlim

Sets the *minor* iteration limit. SNOPT will stop as soon as the number of *minor iterations* exceeds the iteration limit, in which case the current solution will be reported.

domlim

Sets the domain error limit. Domain errors are evaluation errors in the nonlinear functions. An example of a domain error is trying to evaluate $\sqrt{x}$ for $x < 0$. Other examples include taking logs of negative numbers, and evaluating the real power x^y for $x < \varepsilon$ (x^y is evaluated as $\exp(y \log x)$). When such an error occurs the count of domain errors is incremented: SNOPT will stop if this count exceeds the limit. If the limit has not been reached, reasonable estimates for the function (and derivatives, if requested) are returned and SNOPT continues. For example, in the case of $\sqrt{x}, x < 0$ a zero is passed back for the function value and a large value for the derivative. In many cases SNOPT will be able to recover from these domain errors, especially when they happen at some intermediate point. Nevertheless it is best to add appropriate bounds or linear constraints to ensure that these domain errors don't occur. For example, when an expression $\log(x)$ is present in the model, add a statement like `x.lo = 0.001;`.

bratio

Ratio used in basis acceptance test. When a previous solution or solution estimate exists, GAMS automatically passes this solution to SNOPT so that it can reconstruct an advanced basis. When too many new (i.e. uninitialized with level and/or marginal values) variables or constraints enter the model, it may be better not to use existing basis information, but to instead *crash* a new basis. The `bratio` determines how quickly an existing basis is discarded. A value of 1.0 will discard any basis, while a value of 0.0 will retain any basis.

workfactor

By default, GAMS/SNOPT computes an estimate of the amount of workspace needed by SNOPT, and passes this workspace on to SNOPT for use in solving the model. This estimate is based on the model statistics: number of (nonlinear) equations, number of (nonlinear) variables, number of (nonlinear) nonzeroes, etc. In most cases this is sufficient to solve the model. In some rare cases SNOPT may need more memory, and the user can provide this by specifying a value of workfactor greater than 1. The computed memory estimate is multiplied by the workfactor to determine the amount of workspace made available to SNOPT for the solve.

5.36.5 SNOPT Options

The performance of GAMS/SNOPT is controlled by a number of parameters or "options." Each option has a default value that should be appropriate for most problems. For special situations it is possible to specify non-default values for some or all of the options via the SNOPT option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

Note that the option file is not case sensitive. The tables below contain summary information about the SNOPT options, default values, and links to more detailed explanations.

5.36.5.1 Printing

Option	Description	Default
major print level	Amount of information printed during optimization (listing file)	1
minor print level	Amount of information printed during optimization (listing file)	1
print frequency	Number of iterations between each log line (listing file)	100
solution	Prints SNOPT solution (listing file)	NO
summary frequency	Number of iterations between each log line (log file)	100
suppress parameters	Suppress printing of parameters (listing file)	
system information	Provides additional information on the progress of the iterations (listing file)	NO
timing level	Amount of timing information (listing file)	3

5.36.5.2 Problem specification

Option	Description	Default
feasible point	Ignore objective function and find a feasible point	
infinite bound	Bounds larger than this number are considered Infinity	1.0e20

5.36.5.3 Convergence tolerances

Option	Description	Default
major feasibility tolerance	Feasibility tolerance for nonlinear constraints	1.0e-6
major optimality tolerance	Specifies the final accuracy of the dual variables	1.0e-6
minor feasibility tolerance	Feasibility tolerance for all variables and linear constraints	1.0e-6

5.36.5.4 Derivative checking

Option	Description	Default
verify level	Finite-difference checks on the derivatives	-1

5.36.5.5 Scaling

Option	Description	Default
scale option	Controls problem scaling	auto
scale print	Print scaling factors (listing file)	
scale tolerance	Scale tolerance	0.9

5.36.5.6 Other tolerances

Option	Description	Default
crash tolerance	Allow crash procedure to ignore small elements in eligible columns	0.1
linesearch tolerance	Controls accuracy of steplength selected	0.9
pivot tolerance	Used to keep the basis non-singular	3.67e-11

5.36.5.7 QP subproblems

Option	Description	Default
crash option	Controls the basis crash algorithm	auto: 0 or 3
elastic weight	Control for elastic mode	1.0e4
iterations limit	Minor iteration limit	GAMS iterlim
partial price	Number of segments in partial pricing strategy	auto
qpsolver	Controls method used for QP subproblems	Cholesky

5.36.5.8 SQP method

Option	Description	Default
central difference interval	Not applicable: GAMS provides analytic derivatives	6.0e-6

Option	Description	Default
cold start	Ignore advanced basis and use CRASH procedure	
derivative linesearch	Linesearch method (safeguarded cubic interpolation) with use of derivatives	
difference interval	Not applicable: GAMS provides analytic derivatives	1.5e-8
function precision	Relative accuracy with which the nonlinear functions are evaluated	3.00e-13
major iterations limit	Max number of major iterations	GAMS iterlim
major step limit	Limits the change in x during a linesearch	2.0
minor iterations limit	Max number of minor iterations between linearizations of nonlinear constraints	500
new superbasics limit	Limit on new superbasics when a QP subproblem is solved	99
nonderivative linesearch	Linesearch method (safeguarded quadratic interpolation) without use of derivatives	
penalty parameter	Initial penalty parameter	0
proximal point method	Controls promimal point method used for solving linear constraints	1
reduced hessian dimension	Size of Hessian matrix	auto
superbasics limit	Maximum number of superbasics	1
unbounded objective value	Determines when a problem is called unbounded	1.0e15
unbounded step size	Determines when a problem is called unbounded	1.0e18
violation limit	Limit on maximum constraint violation after the linesearch	10
warm start	Use advanced basis provided by GAMS	

5.36.5.9 Hessian approximation

Option	Description	Default
hessian frequency	How often the full Hessian is reset to the identity matrix	999999
hessian full memory	Approximate Hessian is treated as a dense matrix	
hessian limited memory	Limited-memory procedure is used to update a diagonal Hessian approximation	
hessian updates	How often the limited memory Hessian is reset	10

5.36.5.10 Frequencies

Option	Description	Default
check frequency	Controls frequency of linear constraint satisfaction test	60
expand frequency	Setting for anti-cycling mechanism	10000
factorization frequency	Number of iterations between basis factorizations	auto: 100 or 50

5.36.5.11 LUSOL options

Option	Description	Default
LU complete pivoting	LUSOL pivoting strategy	
LU density tolerance	Controls when to move to a dense factorization	0.6
LU factor tolerance	Trade-off between stability and sparsity in basis factorization	auto
LU partial pivoting	LUSOL pivoting strategy	yes
LU rook pivoting	LUSOL pivoting strategy	
LU singularity tolerance	Protection against ill-conditioned basis matrices	3.2e-11
LU update tolerance	Trade-off between stability and sparsity in basis factorization	auto

central difference interval *(real)*: Not applicable: GAMS provides analytic derivatives ↩

Default: `6.0e-6`

check frequency *(integer)*: Controls frequency of linear constraint satisfaction test ↩

Every i^{th} minor iteration after the most recent basis factorization, a numerical test is made to see if the current solution x satisfies the general linear constraints (including linearized nonlinear constraints, if any). The constraints are of the form $Ax - s = b$, where s is the set of slack variables. To perform the numerical test, the residual vector $r = b - Ax + s$ is computed. If the largest component of r is judged to be too large, the current basis is refactorized and the basic variables are recomputed to satisfy the general constraints more accurately.

`Check frequency 1` is useful for debugging purposes, but otherwise this option should not be needed.

Range: {1, ..., ∞}

Default: `60`

cold start *(no value)*: Ignore advanced basis and use CRASH procedure ↩

Requests that the CRASH procedure be used to choose an initial basis. This option takes precedence over the GAMS bratio option.

crash option *(integer)*: Controls the basis crash algorithm ↩

Except on restarts, a CRASH procedure is used to select an initial basis from certain rows and columns of the constraint matrix ($A \quad -I$). The `Crash option` i determines which rows and columns of A are eligible initially, and how many times CRASH is called. Columns of $-I$ are used to pad the basis where necessary.

If $i \geq 1$, certain slacks on inequality rows are selected for the basis first. (If $i \geq 2$, numerical values are used to exclude slacks that are close to a bound.) CRASH then makes several passes through the columns of A, searching for a basis matrix that is essentially triangular. A column is assigned to "pivot" on a particular row if the column contains a suitably large element in a row that has not yet been assigned. (The pivot elements ultimately form the diagonals of the triangular basis.) For the remaining unassigned rows, slack variables are inserted to complete the basis.

By default, `crash option 3` is used for linearly constrained problems and `crash option 0` for problems with nonlinear constraints.

Default: `auto: 0 or 3`

value	meaning
0	Initial basis will be a slack basis. The initial basis contains only slack variables: $B = I$.
1	One phase CRASH. CRASH is called once, looking for a triangular basis in all rows and columns of the matrix A.
2	Two phase CRASH. CRASH is called twice (if there are nonlinear constraints). The first call looks for a triangular basis in linear rows, and the iteration proceeds with simplex iterations until the linear constraints are satisfied. The Jacobian is then evaluated for the first major iteration and CRASH is called again to find a triangular basis in the nonlinear rows (retaining the current basis for linear rows).
3	Three phase CRASH. CRASH is called up to three times (if there are nonlinear constraints). The first two calls treat *linear equalities* and *linear inequalities* separately. As before, the last call treats nonlinear rows before the first major iteration.

crash tolerance *(real)*: Allow crash procedure to ignore small elements in eligible columns ↩

The `Crash tolerance` r allows the starting procedure CRASH to ignore certain small nonzeros in each column of A. If $a_{\max}$ is the largest element in column j, other nonzeros a_{ij} in the column are ignored if $|a_{ij}| \leq a_{\max} \times r$. (To be meaningful, r should be in the range $0 \leq r < 1$.)

When $r > 0.0$, the basis obtained by CRASH may not be strictly triangular, but it is likely to be nonsingular and almost triangular. The intention is to obtain a starting basis containing more columns of A and fewer (arbitrary) slacks. A feasible solution may be reached sooner on some problems.

For example, suppose the first m columns of A are the matrix shown under LU_factor_tolerance, i.e. a tridiagonal matrix with entries $-1, 2, -1$. To help CRASH choose all m columns for the initial basis, we would specify `Crash tolerance` r for some value of $r > 0.5$.

Range: [`0`, `1`]

Default: `0.1`

derivative linesearch *(no value)*: Linesearch method (safeguarded cubic interpolation) with use of derivatives ↩

At each major iteration a linesearch is used to improve the merit function. A `Derivative linesearch` uses safeguarded cubic interpolation and requires both function and gradient values to compute estimates of the step size α_k.

difference interval *(real)*: Not applicable: GAMS provides analytic derivatives ↩

This alters the interval h_1 that is used to estimate gradients by forward differences in the following circumstances:

- In the initial ("cheap") phase of verifying the problem derivatives.
- For verifying the problem derivatives.
- For estimating missing derivatives.

In all cases, a derivative with respect to x_j is estimated by perturbing that component of x to the value $x_j + h_1(1 + |x_j|)$, and then evaluating $f_0(x)$ or $f(x)$ at the perturbed point. The resulting gradient estimates should be accurate to $O(h_1)$ unless the functions are badly scaled. Judicious alteration of h_1 may sometimes lead to greater accuracy. This option has limited use in a GAMS environment as GAMS provides analytical gradients.

Default: `1.5e-8`

elastic weight *(real)*: Control for elastic mode ↩

The *elastic_weight* ω determines the initial weight γ associated with problem NP(γ).

At any given major iteration k, elastic mode is started if the QP subproblem is infeasible or if the QP dual variables are larger in magnitude than $\omega(1 + \|g(x_k)\|_2)$, where g is the objective gradient. In either case, the QP is re-solved in elastic mode with $\gamma = \omega(1 + \|g(x_k)\|_2)$.

Thereafter, γ is increased (subject to a maximum allowable value) at any point that is optimal for problem NP(γ), but not feasible for NP. After the rth such increase, $\gamma = \omega 10^r(1+\|g(x_{k1})\|_2)$, where x_{k1} is the iterate at which γ was first needed.

Default: `1.0e4`

expand frequency *(integer)*: Setting for anti-cycling mechanism ↩

This option is part of the EXPAND anti-cycling procedure [83] designed to make progress even on highly degenerate problems.

For linear models, the strategy is to force a positive step at every iteration, at the expense of violating the bounds on the variables by a small amount. Suppose that the minor feasibility tolerance is δ and the expand frequency is k. Over a period of k iterations, the tolerance actually used by SNOPT increases from 0.5δ to δ (in steps of $0.5\delta/k$).

For nonlinear models, the same procedure is used for iterations in which there is only one superbasic variable. (Cycling can occur only when the current solution is at a vertex of the feasible region.) Thus, zero steps are allowed if there is more than one superbasic variable, but otherwise positive steps are enforced.

At least every k iterations, a resetting procedure eliminates any infeasible nonbasic variables. Increasing k helps to reduce the number of these slightly infeasible nonbasic variables. However, it also diminishes the freedom to choose a large pivot element (see pivot tolerance).

Range: {1, ..., ∞}

Default: `10000`

factorization frequency *(integer)*: Number of iterations between basis factorizations ↩

At most k basis changes will occur between factorizations of the basis matrix.

- With linear programs, the basis factors are usually updated every iteration. The default k is reasonable for typical problems. Smaller values (say $k = 75$ or $k = 50$) may be more efficient on problems that are rather dense or poorly scaled.
- When the problem is nonlinear, fewer basis updates will occur as an optimum is approached. The number of iterations between basis factorizations will therefore increase. During these iterations a test is made regularly (according to the check frequency) to ensure that the general constraints are satisfied. If necessary the basis will be refactorized before the limit of k updates is reached.

By default, the frequency is set to 100 for linear models and 50 otherwise.

Range: {1, ..., ∞}

Default: `auto: 100 or 50`

feasible point *(no value)*: Ignore objective function and find a feasible point ↩

The keyword `feasible point` means "Ignore the objective function" while finding a feasible point for the linear and nonlinear constraints. It can be used to check that the nonlinear constraints are feasible.

Default: turned off.

function precision *(real)*: Relative accuracy with which the nonlinear functions are evaluated ↩

The *relative function precision* ϵ_R is intended to be a measure of the relative accuracy with which the nonlinear functions can be computed. For example, if $f(x)$ is computed as 1000.56789 for some relevant x and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_R would be `1.0e-6`.

(Ideally the functions $f_0(x)$ or $f_i(x)$ should have magnitude of order 1. If all functions are substantially less than 1 in magnitude, ϵ_R should be the absolute precision. For example, if $f(x) =$ `1.23456789e-4` at some point and if the first 6 significant digits are known to be correct, the appropriate value for ϵ_R would be `1.0e-10`.)

- The default value of ϵ_R is appropriate for simple analytic functions.
- In some cases the function values will be the result of extensive computation, possibly involving an iterative procedure that can provide rather few digits of precision at reasonable cost. Specifying an appropriate `Function precision` may lead to savings, by allowing the linesearch procedure to terminate when the difference between function values along the search direction becomes as small as the absolute error in the values.

Default: `3.00e-13`

hessian frequency *(integer)*: How often the full Hessian is reset to the identity matrix ↩

This option sets the frequency i for resetting the full memory Hessian. For example, if hessian full memory is selected and i BFGS updates have already been carried out, the Hessian approximation is reset to the identity matrix. (For certain problems, occasional resets may improve convergence, but in general they should not be necessary.)

`Hessian Full Memory` and `Hessian Frequency = 20` have a similar effect to `Hessian Limited Memory` and `Hessian Updates = 20`, except that the latter retains the current diagonal during resets.

Default: `999999`

hessian full memory *(no value)*: Approximate Hessian is treated as a dense matrix ↩

This option selects the full storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

If `Hessian Full Memory` is specified, the approximate Hessian is treated as a dense matrix and the BFGS updates are applied explicitly. This option is most efficient when the number of nonlinear variables n_1 is not too large. In this case, the storage requirement is fixed and one can expect Q-superlinear convergence to the solution.

By default, this storage method is chosen when the number of nonlinear variables $n_1 \leq 75$.

hessian limited memory *(no value)*: Limited-memory procedure is used to update a diagonal Hessian approximation ↩

This option selects the limited memory storage method for storing and updating the approximate Hessian. (SNOPT uses a quasi-Newton approximation to the Hessian of the Lagrangian. A BFGS update is applied after each major iteration.)

`Hessian Limited Memory` should be used on problems where the number of nonlinear variables n_1 is large. In this case a limited-memory procedure is used to update a diagonal Hessian approximation H_r a limited number of times. (Updates are accumulated as a list of vector pairs. They are discarded at regular intervals after H_r has been reset to their diagonal.)

By default, this storage method is chosen when the number of nonlinear variables $n_1 > 75$.

hessian updates *(integer)*: How often the limited memory Hessian is reset ↩

If hessian limited memory is selected and i BFGS updates have already been carried out, all but the diagonal elements of the accumulated updates are discarded and the updating process starts again.

Broadly speaking, the more updates stored, the better the quality of the approximate Hessian. However, the cost of each QP iteration also increases with the number of updates. The default value is likely to give a robust algorithm without significant expense, but faster convergence can sometimes be obtained with significantly fewer updates (e.g., $i = 5$).

Default: `10`

infinite bound *(real)*: Bounds larger than this number are considered Infinity ↩

If $r > 0$, r defines the "infinite" bound `infBnd` in the definition of the problem constraints. Any upper bound greater than or equal to `infBnd` will be regarded as plus infinity (and similarly for a lower bound less than or equal to `-infBnd`). If $r \leq 0$, the default value is used.

Default: `1.0e20`

iterations limit *(integer)*: Minor iteration limit ↩

The maximum number of minor iterations allowed (i.e., iterations of the simplex method or the QP algorithm), summed over all major iterations. This option, if set, overrides the GAMS iterlim specification.

Default: `GAMS iterlim`

linesearch tolerance *(real)*: Controls accuracy of steplength selected ↩

This controls the accuracy with which a steplength will be located along the direction of search at each iteration. At the start of each linesearch a target directional derivative for the merit function is identified. The `linesearch tolerance` t determines the accuracy to which this target value is approximated.

- Larger values like $t = 0.9$ request just moderate accuracy in the linesearch.
- If the nonlinear functions are cheap to evaluate, as is usually the case for GAMS models, a more accurate search may be appropriate; try $t = 0.1$, 0.01 or 0.001. The number of major iterations might decrease.
- If the nonlinear functions are expensive to evaluate, a less accurate search may be appropriate. In the case of running under GAMS where all gradients are known, try $t = 0.99$. The number of major iterations might increase, but the total number of function evaluations may decrease enough to compensate.

Range: [0, 1]

Default: 0.9

LU complete pivoting *(no value)*: LUSOL pivoting strategy ↩

See LU_partial_pivoting.

LU density tolerance *(real)*: Controls when to move to a dense factorization ↩

The density tolerance r_1 is used during LUSOL's basis factorization $B = LU$. Columns of L and rows of U are formed one at a time, and the remaining rows and columns of the basis are altered appropriately. At any stage, if the density of the remaining matrix exceeds r_1, the Markowitz strategy for choosing pivots is terminated and the remaining matrix is factored by a dense LU procedure. Raising the density tolerance towards 1.0 may give slightly sparser LU factors, with a slight increase in factorization time.

See also LU_singularity_tolerance.

Range: [0, 1]

Default: 0.6

LU factor tolerance *(real)*: Trade-off between stability and sparsity in basis factorization ↩

`LU factor tolerance` r_1

`LU update tolerance` r_2

These tolerances affect the stability and sparsity of the basis factorization $B = LU$ during refactorization and updating, respectively. They must satisfy $r_1, r_2 \geq 1.0$. The matrix L is a product of matrices of the form

$$\begin{pmatrix} 1 & \\ \mu & 1 \end{pmatrix},$$

where the multipliers μ satisfy $|\mu| \leq r_i$. Smaller values of r_i favor stability, while larger values favor sparsity.

- For large and relatively dense problems, smaller values of r_1 (e.g. $r_1 = 3.0$) may give a useful improvement in stability without impairing sparsity to a serious degree.
- For certain very regular structures (e.g., band matrices) it may be necessary to reduce r_1 and/or r_2 in order to achieve stability. For example, if the columns of A include a submatrix of the form

$$\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{pmatrix},$$

  both r_1 and r_2 should be in the range $1.0 \leq r_i < 2.0$.

For linear models, the defaults are $r_1 = 100$ and $r_2 = 10$, while for nonlinear models both tolerances default to 3.99.

See also LU_update_tolerance.

Range: [1, ∞]

Default: auto

LU partial pivoting *(no value)*: LUSOL pivoting strategy ↩

The LUSOL factorization implements a Markowitz-type search for pivots that locally minimizes fill-in subject to a threshold pivoting stability criterion. The `rook pivoting` and `complete pivoting` options are more expensive than `partial pivoting` but are more stable and better at revealing rank, as long as the LU_factor_tolerance is not too large (say $r_1 < 2.0$).

When numerical difficulties are encountered, SNOPT automatically reduces the LU tolerances toward 1.0 and switches (if necessary) to rook or complete pivoting before reverting to the default or specified options at the next refactorization. (With sysout on and system information enabled, relevant messages are output to the listing file.)

Default: `yes`

LU rook pivoting *(no value)*: LUSOL pivoting strategy ↩

See LU_partial_pivoting.

LU singularity tolerance *(real)*: Protection against ill-conditioned basis matrices ↩

The singularity tolerance r_2 helps guard against ill-conditioned basis matrices. After B is refactorized, the diagonal elements of U are tested as follows: if $|U_{jj}| \leq r_2$ or $|U_{jj}| < r_2 \max_i |U_{ij}|$, the jth column of the basis is replaced by the corresponding slack variable. (This is most likely to occur after a restart.)

See also LU_density_tolerance.

Default: `3.2e-11`

LU update tolerance *(real)*: Trade-off between stability and sparsity in basis factorization ↩

See LU_factor_tolerance for details.

Range: [1, ∞]

Default: `auto`

major feasibility tolerance *(real)*: Feasibility tolerance for nonlinear constraints ↩

This tolerance ϵ_r specifies how accurately the nonlinear constraints should be satisfied. The default value of `1.0e-6` is appropriate when the constraints are expected to have at least that accuracy.

Let `rowerr` be the maximum nonlinear constraint violation, normalized by the size of the solution. It is required to satisfy

$$\text{rowerr} = \max_i \ \text{viol}_i / \|x\| \leq \ \epsilon_r, \tag{8}$$

where viol_i is the violation of the ith nonlinear constraint.

In the GAMS/SNOPT iteration log, `rowerr` appears as the quantity labeled `"Feasibl"`. If some of the problem functions are known to be of low accuracy, a larger `Major feasibility tolerance` may be appropriate.

Default: `1.0e-6`

major iterations limit *(integer)*: Max number of major iterations ↩

This is the maximum number of major iterations allowed. It is intended to guard against an excessive number of linearizations of the constraints. By default it is set to `max(1000,m)`.

Default: `GAMS iterlim`

major optimality tolerance *(real)*: Specifies the final accuracy of the dual variables ↩

This tolerance ϵ_d specifies the final accuracy of the dual variables. On successful termination, SNOPT will have computed a solution (x, s, π) such that

$$\text{maxComp} = \max_j \ \text{Comp}_j/\|\pi\| \ \leq \ \epsilon_d, \tag{9}$$

where $Comp_j$ is an estimate of the complementarity slackness for variable j. The values $Comp_j$ are computed from the final QP solution using the reduced gradients $d_j = g_j - \pi^T a_j$ (where g_j is the jth component of the objective gradient, a_j is the associated column of the constraint matrix $(A \quad -I)$ and π is the set of QP dual variables):

$$\text{Comp}_j = \begin{cases} d_j \min\{x_j - l_j, 1\} & \text{if } d_j \geq 0; \\ -d_j \min\{u_j - x_j, 1\} & \text{if } d_j < 0. \end{cases}$$

In the GAMS/SNOPT iteration log, *maxComp* appears as the quantity labeled `"Optimal"`.

Default: `1.0e-6`

major print level *(integer)*: Amount of information printed during optimization (listing file) ↩

This controls the amount of output to the GAMS listing file at each major iteration. This output is only visible if the sysout option is turned on. `Major print level 1` gives normal output for linear and nonlinear problems, and `Major print level 11` gives additional details of the Jacobian factorization that commences each major iteration. In general, the value specified may be thought of as a binary number of the form

```
Major print level   JFDXbs
```

where each letter stands for a digit that is either `0` or `1` as follows:

- `s` single line that gives a summary of each major iteration. (This entry in `JFDXbs` is not strictly binary since the summary line is printed whenever JFDXbs ≥ 1).
- `b` BASIS statistics, i.e., information relating to the basis matrix whenever it is refactorized. (This output is always provided if JFDXbs ≥ 10).
- `X` x_k, the nonlinear variables involved in the objective function or the constraints.
- `D` π_k, the dual variables for the nonlinear constraints.
- `F` $F(x_k)$, the values of the nonlinear constraint functions.
- `J` $J(x_k)$, the Jacobian.

To obtain output of any items `JFDXbs`, set the corresponding digit to `1`, otherwise to `0`.

If `J=1`, the Jacobian will be output column-wise at the start of each major iteration. Column j will be preceded by the value of the corresponding variable x_j and a key to indicate whether the variable is basic, superbasic or nonbasic. (Hence if `J=1`, there is no reason to specify `X=1` unless the objective contains more nonlinear variables than the Jacobian.) A typical line of output is

```
3  1.250000D+01 BS      1  1.00000E+00      4  2.00000E+00
```

which would mean that x_3 is basic at value 12.5, and the third column of the Jacobian has elements of 1.0 and 2.0 in rows 1 and 4.

`Major print level 0` suppresses most output, except for error messages.

Default: `1`

major step limit *(real)*: Limits the change in x during a linesearch ↩

This parameter r limits the change in x during a linesearch. It applies to all nonlinear problems, once a "feasible solution" or "feasible subproblem" has been found.

1. A linesearch determines a step α over the range $0 < \alpha \leq \beta$, where β is 1 if there are nonlinear constraints, or the step to the nearest upper or lower bound on x if all the constraints are linear. Normally, the first steplength tried is $\alpha_1 = \min(1, \beta)$.
2. In some cases, such as $f(x) = ae^{bx}$ or $f(x) = ax^b$, even a moderate change in the components of x can lead to floating-point overflow. The parameter r is therefore used to define a limit $\bar{\beta} = r(1 + \|x\|)/\|p\|$ (where p is the search direction), and the first evaluation of $f(x)$ is at the potentially smaller steplength $\alpha_1 = \min(1, \bar{\beta}, \beta)$.
3. Wherever possible, upper and lower bounds on x should be used to prevent evaluation of nonlinear functions at meaningless points. The `Major step limit` provides an additional safeguard. The default value $r = 2.0$ should not affect progress on well behaved problems, but setting $r = 0.1$ or 0.01 may be helpful when rapidly varying functions are present. A "good" starting point may be required. An important application is to the class of nonlinear least-squares problems.
4. In cases where several local optima exist, specifying a small value for r may help locate an optimum near the starting point.

Default: `2.0`

minor feasibility tolerance *(real)*: Feasibility tolerance for all variables and linear constraints ↩

SNOPT tries to ensure that all variables eventually satisfy their upper and lower bounds to within this tolerance t. This includes slack variables, so general linear constraints should also be satisfied to within t.

Feasibility with respect to nonlinear constraints is judged by the major feasibility tolerance.

- If the bounds and linear constraints cannot be satisfied to within t, the problem is declared `infeasible`. Let `sInf` be the corresponding sum of infeasibilities. If `sInf` is quite small, it may be appropriate to raise t by a factor of 10 or 100. Otherwise, some error in the data should be suspected.
- Nonlinear functions will be evaluated only at points that satisfy the bounds and linear constraints. If there are regions where a function is undefined, every attempt should be made to eliminate these regions from the problem. For example, if $f(x) = \sqrt{x_1} + \log x_2$, it is essential to place lower bounds on both variables. If $t = 1.0e-6$, the bounds $x_1 \geq 10^{-5}$ and $x_2 \geq 10^{-4}$ might be appropriate. (The log singularity is more serious. In general, keep x as far away from singularities as possible.)
- If the model is scaled (see scale option), feasibility is defined in terms of the `scaled` problem.
- In reality, SNOPT uses t as a feasibility tolerance for satisfying the bounds on x and s in each QP subproblem. If the sum of infeasibilities cannot be reduced to zero, the QP subproblem is declared infeasible. SNOPT is then in *elastic mode* thereafter (with only the linearized nonlinear constraints defined to be elastic). See elastic weight for details.

Default: `1.0e-6`

minor iterations limit *(integer)*: Max number of minor iterations between linearizations of nonlinear constraints ↩

Minor iterations limit k. If the number of minor iterations for the optimality phase of the QP subproblem exceeds k, then all nonbasic QP variables that have not yet moved are frozen at their current values and the reduced QP is solved to optimality. Note that more than k minor iterations may be necessary to solve the reduced QP to optimality. These extra iterations are necessary to ensure that the terminated point gives a suitable direction for the linesearch. In the major iteration log, a `t` at the end of a line indicates that the corresponding QP was artificially terminated using the limit k. Note that iterations limit defines an independent absolute limit on the total number of minor iterations (summed over all QP subproblems).

Default: `500`

minor print level *(integer)*: Amount of information printed during optimization (listing file) ↩

This controls the amount of output to the GAMS listing file during solution of the QP subproblems. It is only useful if the GAMS sysout option is turned on. The value of k has the following effect:

- 0 No minor iteration output except error messages.
- ≥ 1 A single line of output each minor iteration (controlled by print frequency).
- ≥ 10 Basis factorization statistics generated during the periodic refactorization of the basis (see factorization frequency). Statistics for the *first factorization* of each major iteration are controlled by the major print level.

Default: `1`

new superbasics limit *(integer)*: Limit on new superbasics when a QP subproblem is solved ↩

This option causes early termination of the QP subproblems if the number of free variables has increased significantly since the first feasible point. If the number of new superbasics is greater than `new superbasics limit` the nonbasic variables that have not yet moved are frozen and the resulting smaller QP is solved to optimality. In the major iteration log, a `"T"` at the end of a line indicates that the QP was terminated early in this way.

Default: `99`

nonderivative linesearch *(no value)*: Linesearch method (safeguarded quadratic interpolation) without use of derivatives ↩

A `nonderivative linesearch` can be slightly less robust on difficult problems, and it is recommended that the default derivative linesearch be used if the functions and derivatives can be computed at approximately the same cost. If the gradients are very expensive relative to the functions, a nonderivative linesearch may give a significant decrease in computation time. In a GAMS environment derivative linesearch (the default) is more appropriate.

partial price *(integer)*: Number of segments in partial pricing strategy ↩

This parameter sets the number of segments k using in partial pricing and is recommended for large problems that have significantly more variables than constraints. It reduces the work required for each "pricing" operation (i.e. when a nonbasic variable is selected to become superbasic).

- When $k = 1$, all columns of the constraint matrix $(A \quad -I)$ are searched.

- Otherwise, A and I are partitioned to give k roughly equal segments A_j, I_j ($j = 1$ to k). If the previous pricing search was successful on A_j, I_j, the next search begins on the segments A_{j+1}, I_{j+1}. (All subscripts here are modulo k.)
- If a reduced gradient is found that is larger than some dynamic tolerance, the variable with the largest such reduced gradient (of appropriate sign) is selected to become superbasic. If nothing is found, the search continues on the next segments A_{j+2}, I_{j+2}, and so on.
- `Partial price` t (or $t/2$ or $t/3$) may be appropriate for time-stage models having t time periods.

The default is `10` for linear models and `1` for nonlinear models.

Range: {`1`, ..., ∞}

Default: `auto`

penalty parameter *(real)*: Initial penalty parameter ↩

After a QP subproblem has been solved, new estimates of the NLP solution are computed using a linesearch on the augmented Lagrangian merit function. This functions contains penalty parameters, which may be increased to ensure descent.

Default: `0`

pivot tolerance *(real)*: Used to keep the basis non-singular ↩

During solution of QP subproblems, the pivot tolerance r is used to prevent columns entering the basis if they would cause the basis to become almost singular.

- When x changes to $x + \alpha p$ for some search direction p, a ratio test is used to determine which component of x reaches an upper or lower bound first. The corresponding element of p is called the pivot element.
- Elements of p are ignored (and therefore cannot be pivot elements) if they are smaller than the pivot tolerance r.
- It is common for two or more variables to reach a bound at essentially the same time. In such cases, the minor feasibility tolerance (say t) provides some freedom to maximize the pivot element and thereby improve numerical stability. Excessively small values of t should therefore not be specified.
- To a lesser extent, the expand frequency (say f) also provides some freedom to maximize the pivot element. Excessively *large* values of f should therefore not be specified.

Default: `3.67e-11`

print frequency *(integer)*: Number of iterations between each log line (listing file) ↩

Synonym: log_frequency

When sysout is turned on and minor print level is positive, a line of the QP iteration log will be printed on the listing file every kth minor iteration.

Range: {`1`, ..., ∞}

Default: `100`

proximal point method *(integer)*: Controls promimal point method used for solving linear constraints ↩

Once the linear constraints are satisfied, the proximal point method chooses a linear feasible point that is closest to x_0, the initial point for the nonlinear variables. The idea is to both satisfy the linear constraints and stay close to the starting values provided for the nonlinear variables. This option is used to disable the proximal point method or to select the norm used.

Default: `1`

value	meaning
0	disable PPM Do not use the proximal point method.

| 1 | one-norm
Minimize the one-norm $||x - x_0||_1$.| | 2 | two-norm
Minimize the two-norm $\frac{1}{2}||x - x_0||_2^2$.|

qpsolver *(string)*: Controls method used for QP subproblems ↩

This specifies the method used to solve system (5) for the search directions in phase 2 of the QP subproblem.

- The Cholesky QP solver is the most robust, but may require a significant amount of computation if the number of superbasics is large.
- The quasi-Newton QP solver does not require the computation of the R at the start of each QP subproblem. It may be appropriate when the number of superbasics is large but relatively few major iterations are needed to reach a solution (e.g., if SNOPT is called with a warm start).
- The conjugate-gradient QP solver is appropriate for problems with many degrees of freedom (say, more than 2000 superbasics).

Default: `Cholesky`

value	meaning
`Cholesky`	full Cholesky factor `QPSolver Cholesky` holds the full Cholesky factor R of the reduced Hessian Z^THZ. As the minor iterations proceed, the dimension of R changes with the number of superbasic variables. If the number of superbasic variables needs to increase beyond the value of reduced hessian dimension, the reduced Hessian cannot be stored and the solver switches to `QPSolver CG`. The Cholesky solver is reactivated if the number of superbasics stabilizes at a value less than the reduced hessian dimension.
`QN`	quasi-Newton method `QPSolver QN` solves the QP using a quasi-Newton method similar to that of MINOS. In this case, R is the factor of a quasi-Newton approximate Hessian.
`CG`	conjugate-gradient method `QPSolver CG` uses an active-set method similar to `QPSolver QN`, but uses the conjugate-gradient method to solve all systems involving the reduced Hessian.

reduced hessian dimension *(integer)*: Size of Hessian matrix ↩

Synonym: hessian_dimension

This specifies that an $i \times i$ triangular matrix R is to be available for use by the `QPSolver Cholesky` option (to define the reduced Hessian according to $R^TR = Z^THZ$). The value of i affects when `QPSolver CG` is activated. The default is computed internally by SNOPT, currently as $\min\{2000, n_1 + 1\}$.

Range: {1, ..., ∞}

Default: `auto`

scale option *(integer)*: Controls problem scaling ↩

Three scale options are available. By default, option 2 is used for linear models and option 1 for nonlinear models. See also scale tolerance and scale print.

Default: `auto`

value	meaning
`0`	No scaling This is recommended if it is known that x and the constraint matrix and Jacobian never have very large elements (say, larger than 100).
`1`	Scale linear variables Linear constraints and variables are scaled by an iterative procedure that attempts to make the matrix coefficients as close as possible to 1.0 (see Fourer [69]). This will sometimes improve the performance of the solution procedures.
`2`	Scale linear + nonlinear variables All constraints and variables are scaled by the iterative procedure. Also, an additional scaling is performed that takes into account columns of $(A \quad -I)$ that are fixed or have positive lower bounds or negative upper bounds. If nonlinear constraints are present, the scales depend on the Jacobian at the first point that satisfies the linear constraints. `Scale option 2` should therefore be used only if (a) a good starting point is provided, and (b) the problem is not highly nonlinear.

scale print *(no value)*: Print scaling factors (listing file) ↩

`Scale print` causes the row-scales $r(i)$ and column-scales $c(j)$ to be printed. The scaled matrix coefficients are $\bar{a}_{ij} = a_{ij}c(j)/r(i)$, and the scaled bounds on the variables and slacks are $\bar{l}_j = l_j/c(j)$, $\bar{u}_j = u_j/c(j)$, where $c(j) \equiv r(j-n)$ if $j > n$.

The listing file will only show these values if the sysout option is turned on. See also scale option and scale tolerance.

scale tolerance *(real)*: Scale tolerance ↩

The scale tolerance t affects how many passes might be needed through the constraint matrix. On each pass, the scaling procedure computes the ratio of the largest and smallest nonzero coefficients in each column:

$$\rho_j = \max_i |a_{ij}| / \min_i |a_{ij}| \qquad (a_{ij} \neq 0).$$

If $\max_j \rho_j$ is less than r times its previous value, another scaling pass is performed to adjust the row and column scales. Raising r from 0.9 to 0.99 (say) usually increases the number of scaling passes through A. At most 10 passes are made.

See also scale option and scale print.

Range: [`0`, `1`]

Default: `0.9`

solution *(string)*: Prints SNOPT solution (listing file) ↩

This option causes the SNOPT solution file to be printed to the GAMS listing file, provided the sysout option is also turned on.

Default: turned off.

Default: `NO`

value	meaning
NO	Turn off printing of solution
YES	Turn on printing of solution

summary frequency *(integer)*: Number of iterations between each log line (log file) ↩

If minor print level is positive, a line of the QP iteration log is output every kth minor iteration.

Range: {1, ..., ∞}

Default: 100

superbasics limit *(integer)*: Maximum number of superbasics ↩

This places a limit on the storage allocated for superbasic variables. Ideally, i should be set slightly larger than the number of degrees of freedom expected at an optimal solution.

For linear programs, an optimum is normally a basic solution with no degrees of freedom. The default value of i is therefore 1. For nonlinear problems, the number of degrees of freedom is often called the "number of independent variables".

Normally, i need not be greater than $n_1 + 1$, where n_1 is the number of nonlinear variables. For many problems, i may be considerably smaller than n_1. This will save storage if n_1 is very large.

This parameter also sets the reduced hessian dimension, unless the latter is specified explicitly (and conversely). If neither parameter is specified, GAMS chooses values for both, based on problem characteristics.

Range: {1, ..., ∞}

Default: 1

suppress parameters *(no value)*: Suppress printing of parameters (listing file) ↩

Normally SNOPT prints the option file as it is being read, and then prints a complete list of the available keywords and their final values. The `suppress parameters` option tells SNOPT not to print the full list. Used in conjunction with the sysout option.

system information *(string)*: Provides additional information on the progress of the iterations (listing file) ↩

The `Yes` option provides additional information on the progress of the iterations, including basis repair details when ill-conditioned bases are encountered and the LU factorization parameters are strengthened.

Default: NO

value	meaning
NO	Turn off additional printing of information on progress of algorithm
YES	Turn on additional printing of information on progress of algorithm

timing level *(integer)*: Amount of timing information (listing file) ↩

Amount of timing information written to the listing file. Used in conjunction with the sysout option.

Range: {0, ..., 3}

Default: 3

unbounded objective value *(real)*: Determines when a problem is called unbounded ↩

This parameter sets the value $f_{\max}$ *intended* to detect unboundedness in nonlinear problems. See unbounded step size for the setting of $\alpha_{\max}$.

During a line search, f_0 is evaluated at points of the form $x + \alpha p$, where x and p are fixed and α varies. If $|f_0|$ exceeds $f_{\max}$ or α exceeds $\alpha_{\max}$, iterations are terminated with the exit message `Problem is unbounded (or badly scaled)`.

Unboundedness in x is best avoided by placing finite upper and lower bounds on the variables.

Default: 1.0e15

unbounded step size *(real)*: Determines when a problem is called unbounded ↩

This parameter sets the value $\alpha_{\max}$ used in the unboundedness test: see unbounded objective value for details.

Default: 1.0e18

verify level *(integer)*: Finite-difference checks on the derivatives ↩

This option refers to finite-difference checks on the derivatives computed by the user-provided routines. Derivatives are checked at the first point that satisfies all bounds and linear constraints.

This option has limited use in a GAMS environment.

Default: -1

value	meaning
0	Cheap test
1	Check individual gradients
2	Check individual columns of the Jacobian
3	Combines verify level 1 and 2
-1	Derivative checking is disabled

violation limit *(integer)*: Limit on maximum constraint violation after the linesearch ↩

This parameter τ is used to define an absolute limit on the magnitude of the maximum constraint violation after the line search. On completion of the line search, the new iterate x_{k+1} satisfies the condition

$$v_i(x_{k+1}) \leq \tau \max\{1, v_i(x_0)\},$$

where x_0 is the point at which the nonlinear constraints are first evaluated and $v_i(x)$ is the ith nonlinear constraint violation $v_i(x) = \max(0, l_i - F_i(x), F_i(x) - u_i)$.

The effect of this violation limit is to restrict the iterates to lie in an *expanded* feasible region whose size depends on the magnitude of τ. This makes it possible to keep the iterates within a

region where the objective is expected to be well-defined and bounded below. If the objective is bounded below for all values of the variables, then τ may be any large positive value.

Default: 10

warm start *(no value)*: Use advanced basis provided by GAMS ↩

Use an advanced basis provided by GAMS. This option takes precedence over the GAMS bratio option.

5.36.6 The SNOPT log

When GAMS/SNOPT solves a linearly constrained problem the following log is visible on the screen:

```
 ==============================
 S N O P T  7.7.6    (Jan 2021)
 ==============================
 SNMEMA EXIT 100 -- finished successfully
 SNMEMA INFO 104 -- memory requirements estimated

 Nonlinear constraints       0     Linear constraints       5
 Nonlinear variables        11     Linear variables         0
 Jacobian  variables         0     Objective variables     11
 Total constraints           5     Total variables         11
 The user has defined      11   out of      11   first  derivatives

 Major Minors     Step   nObj Feasible  Optimal      Objective     nS
     0      4                1           4.3E-02 -4.7361869E+01      4   r
     1      3  1.0E+00       2           2.1E-02 -4.7526522E+01      4   r
     2      2  1.0E+00       3           3.4E-02 -4.7586643E+01      5 s
     3      2  1.0E+00       4           1.1E-02 -4.7662724E+01      6
     4      2  1.0E+00       5           4.8E-03 -4.7678618E+01      5
     5      1  1.0E+00       6           1.3E-02 -4.7701147E+01      5
     6      2  1.0E+00       7           5.2E-03 -4.7704100E+01      6
     7      2  1.0E+00       8           3.5E-04 -4.7705775E+01      5
     8      2  1.0E+00       9           5.1E-03 -4.7706132E+01      6
     9      1  1.0E+00      10           6.1E-05 -4.7706500E+01      6

 Major Minors     Step   nObj Feasible  Optimal      Objective     nS
    10      1  1.0E+00      11           6.1E-05 -4.7706513E+01      6
    11      1  1.0E+00      12           5.0E-05 -4.7706515E+01      6
    12      1  1.0E+00      13           1.6E-05 -4.7706515E+01      6
    13      1  1.0E+00      14           2.0E-06 -4.7706515E+01      6
    14      1  1.0E+00      15          (7.1E-07)-4.7706515E+01      6
 SNOPTA EXIT   0 -- finished successfully
 SNOPTA INFO   1 -- optimality conditions satisfied

 Problem name                  mixer
 No. of iterations                30   Objective          -4.7706514831E+01
 No. of major iterations          14   Linear    obj. term  0.0000000000E+00
                                       Nonlinear obj. term -4.7706514831E+01
 User function calls (total)      15
 No. of superbasics                6   No. of basic nonlinears            4
 No. of degenerate steps           0   Percentage                      0.00
 Max x                      11 1.6E+00  Max pi                      3 1.5E+01
 Max Primal infeas           0 0.0E+00  Max Dual infeas             4 6.1E-05
```

```
 Solution not printed
 Time for MPS input                              0.00 seconds
 Time for solving problem                        0.00 seconds
 Time for solution output                        0.00 seconds
 Time for constraint functions                   0.00 seconds
 Time for objective function                     0.00 seconds
```

For a nonlinearly constrained problem, the log is somewhat different:

```
 ==============================
 S N O P T  7.7.6    (Jan 2021)
 ==============================
 SNMEMA EXIT 100 -- finished successfully
 SNMEMA INFO 104 -- memory requirements estimated

 Nonlinear constraints      23     Linear constraints      16
 Nonlinear variables        35     Linear variables         8
 Jacobian  variables        35     Objective variables      0
 Total constraints          39     Total variables         43
 The user has defined      56   out of      56   first  derivatives

 Major Minors     Step   nCon Feasible  Optimal  MeritFunction     nS Penalty
     0     22              1  1.9E+00  1.0E+00  6.0000000E+02      3           r i
     1      3  5.0E-03      2  1.9E+00  1.0E+00 -5.8415017E+04      3 7.6E-01   rli
     2      1  5.0E-03      4  1.9E+00  1.0E+00 -2.1089442E+05      3 2.8E+00   rli
     3     19  1.0E+00      6  7.9E-01  1.7E+00 -3.1110693E+04      3 2.8E+00 sM
     4      6  1.0E+00      8  4.1E-01  4.4E+00 -3.1423080E+04      4 3.2E+00  m
     5      4  1.0E+00      9  2.1E-01  3.9E+01 -1.3134062E+04      5 3.2E+00
     6      1  1.0E+00     11  1.3E-03  2.3E+02  5.1483376E+02      5 3.2E+00  m
     7      1  1.0E+00     13  1.4E-05  2.1E+00  5.1732760E+02      5 3.2E+00  m
     8      2  1.0E+00     15  1.4E-03  3.5E+01  5.3714297E+02      4 3.2E+00  m
     9      1  1.0E+00     16  1.9E-03  4.0E+01  5.5832995E+02      4 3.2E+00

 Major Minors     Step   nCon Feasible  Optimal  MeritFunction     nS Penalty
    10      4  1.0E+00     17  1.5E-04  1.4E+01  5.6904902E+02      5 3.2E+00
    11      1  3.2E-01     20  4.9E-03  7.4E+01  5.9984627E+02      5 3.2E+00
    12      1  5.7E-01     22  7.1E-03  6.5E+01  6.3671070E+02      5 3.2E+00
    13      2  1.0E+00     23  2.6E-03  1.3E+02  6.9523972E+02      4 3.2E+00
    14      2  3.6E-01     26  5.8E-03  1.5E+02  7.3043319E+02      5 3.2E+00
[...]
    85      2  1.0E+00    281  2.6E-04  6.2E+00  1.0588496E+03      1 3.2E+00
    86      2  1.0E+00    282  2.1E-04  5.1E+00  1.0588957E+03      2 3.2E+00
    87      1  1.0E+00    283  4.9E-05  1.2E+00  1.0589180E+03      2 3.2E+00
    88      1  1.0E+00    284  5.4E-06  1.3E-01  1.0589196E+03      2 3.2E+00
    89      1  1.0E+00    285 (5.0E-07) 1.2E-02  1.0589199E+03      2 3.2E+00

 Major Minors     Step   nCon Feasible  Optimal  MeritFunction     nS Penalty
    90      1  1.0E+00    286 (7.5E-10)(1.9E-07) 1.0589199E+03      2 3.2E+00
 SNOPTA EXIT   0 -- finished successfully
 SNOPTA INFO   1 -- optimality conditions satisfied

 Problem name                 chenrad
 No. of iterations                171   Objective            1.0589198561E+03
 No. of major iterations           90   Linear    obj. term  1.0589198561E+03
 Penalty parameter           3.188E+00   Nonlinear obj. term  0.0000000000E+00
 User function calls (total)      286
```

```
No. of superbasics                 2   No. of basic nonlinears           29
No. of degenerate steps            9   Percentage                      5.26
Max x                       4 6.4E+02   Max pi                     11 2.6E+03
Max Primal infeas          48 4.8E-07   Max Dual infeas            21 9.9E-04
Nonlinear constraint violn    4.8E-07

Solution not printed
Time for MPS input                            0.00 seconds
Time for solving problem                      0.02 seconds
Time for solution output                      0.00 seconds
Time for constraint functions                 0.01 seconds
Time for objective function                   0.00 seconds
```

The SNOPT log shows the following columns:

Major

The current major iteration number.

Minors

The number of iterations required by both the feasibility and optimality phases of the QP subproblem. Generally, `Minor` will be 1 in the later iterations, since theoretical analysis predicts that the correct active set will be identified near the solution (see Section Description of the method)

Step

The step length α taken along the current search direction p. The variables x have just been changed to $x + \alpha p$. On reasonably well-behaved problems, the unit step will be taken as the solution is approached.

nObj

The number of times the nonlinear objective function has been evaluated. `nObj` is printed as a guide to the amount of work required for the linesearch.

nCon

The number of times SNOPT evaluated the nonlinear constraint functions.

MeritFunction

The value of the augmented Lagrangian merit function (6). This function will decrease at each iteration unless it was necessary to increase the penalty parameters (see Section Description of the method). As the solution is approached, `Merit` will converge to the value of the objective at the solution.

In elastic mode, the merit function is a composite function involving the constraint violations weighted by the elastic weight.

If the constraints are linear, this item is labeled `Objective`, the value of the objective function. It will decrease monotonically to its optimal value.

Feasible

The value of `rowerr`, the maximum component of the scaled nonlinear constraint residual. The solution is regarded as acceptably feasible if `Feasible` is less than the Major feasibility tolerance.

If the constraints are linear, all iterates are feasible and this entry is not printed.

Optimal

The value of `maxgap`, the maximum complementarity gap. It is an estimate of the degree of nonoptimality of the reduced costs. The solution is considered to be optimal if `Optimal` is less than the major optimality tolerance.

nS

The current number of superbasic variables.

Penalty

The Euclidean norm of the vector of penalty parameters used in the augmented Lagrangian merit function (not printed if the constraints are linear).

The summary line may include additional code characters that indicate what happened during the course of the iteration.

`c` Central differences have been used to compute the unknown components of the objective and constraint gradients. This should not happen in a GAMS environment.

`d` During the linesearch it was necessary to decrease the step in order to obtain a maximum constraint violation conforming to the value of violation limit.

`l` The norm-wise change in the variables was limited by the value of the major step limit. If this output occurs repeatedly during later iterations, it may be worthwhile increasing the value of major step limit.

`i` If SNOPT is not in elastic mode, an "i" signifies that the QP subproblem is infeasible. This event triggers the start of nonlinear elastic mode, which remains in effect for all subsequent iterations. Once in elastic mode, the QP subproblems are associated with the elastic problem NP(γ).

If SNOPT is already in elastic mode, an "i" indicates that the minimizer of the elastic subproblem does not satisfy the linearized constraints. (In this case, a feasible point for the usual QP subproblem may or may not exist.)

`M` An extra evaluation of the problem functions was needed to define an acceptable positive-definite quasi-Newton update to the Lagrangian Hessian. This modification is only done when there are nonlinear constraints.

`m` This is the same as "M" except that it was also necessary to modify the update to include an augmented Lagrangian term.

`n` No positive-definite BFGS update could be found. The approximate Hessian is unchanged from the previous iteration.

`R` The approximate Hessian has been reset by discarding all but the diagonal elements. This reset will be forced periodically by the hessian frequency and hessian updates keywords. However, it may also be necessary to reset an ill-conditioned Hessian from time to time.

`r` The approximate Hessian was reset after ten consecutive major iterations in which no BFGS update could be made. The diagonals of the approximate Hessian are retained if at least one update has been done since the last reset. Otherwise, the approximate Hessian is reset to the identity matrix.

`s` A self-scaled BFGS update was performed. This update is always used when the Hessian approximation is diagonal, and hence always follows a Hessian reset.

`S` This is the same as a "s" except that it was necessary to modify the self-scaled update to maintain positive definiteness.

`t` The minor iterations were terminated at the minor iterations limit.

`T` The minor iterations were terminated at the new superbasics limit.

`u` The QP subproblem was unbounded.

`w` A weak solution of the QP subproblem was found.

`z` The superbasics limit was reached.

Finally SNOPT prints an exit message. See Section EXIT conditions.

5.36.6.1 EXIT conditions

When the solution procedure terminates, an `SNOPTA EXIT` and `SNOPTA INFO` message is printed to summarize the final result. Here we describe each message and suggest possible courses of action.

`1 -- optimality conditions satisfied`

The final point seems to be a solution. This means that x is feasible (it satisfies the constraints to the accuracy requested), the reduced gradient is negligible, the reduced costs are optimal, and R is nonsingular. In all cases, some caution should be exercised. For example, if the objective value is much better than expected, SNOPT may have obtained an optimal solution to the wrong problem! Almost any item of data could have that effect if it has the wrong value. Verifying that the problem has been defined correctly is one of the more difficult tasks for a model builder.

If nonlinearities exist, one must always ask the question: could there be more than one local optimum? When the constraints are linear and the objective is known to be convex (e.g., a sum of squares) then all will be well if we are *minimizing* the objective: a local minimum is a global minimum in the sense that no other point has a lower function value. (However, many points could have the *same* objective value, particularly if the objective is largely linear.) Conversely, if we are *maximizing* a convex function, a local maximum cannot be expected to be global, unless there are sufficient constraints to confine the feasible region.

Similar statements could be made about nonlinear constraints defining convex or concave regions. However, the functions of a problem are more likely to be neither convex nor concave. Our advice is always to specify a starting point that is as good an estimate as possible, and to include reasonable upper and lower bounds on all variables, in order to confine the solution to the specific region of interest. We expect modelers to *know something about their problem*, and to make use of that knowledge as they themselves know best.

One other caution about the "Optimal solution" message. Some of the variables or slacks may lie outside their bounds more than desired, especially if scaling was requested. If sysout is on, the listing file will contain several indicators of potential issues. `Max Primal infeas` indicates the largest bound infeasibility and which variable is involved. If it is too large, consider restarting with a smaller minor feasibility tolerance (say 10 times smaller) and perhaps a scale option of 0.

Similarly, `Max Dual infeas` indicates which variable is most likely to be at a non-optimal value. Broadly speaking, if

$$\text{Max Dual infeas/Norm of pi} = 10^{-d},$$

then the objective function would probably change in the dth significant digit if optimization could be continued. If d seems too large, consider restarting with smaller values of major feasibility tolerance and minor feasibility tolerance.

Finally, `Nonlinear constraint violn` shows the maximum infeasibility for nonlinear rows. If it seems too large, consider restarting with a smaller major feasibility tolerance.

`2 -- feasible point found`

Occurs only if feasible point is enabled.

`3 -- requested accuracy could not be achieved`

If the requested accuracy could not be achieved, a feasible solution has been found, but the requested accuracy in the dual infeasibilities could not be achieved. An abnormal termination has occurred, but SNOPT is within 10^{-2} of satisfying the major optimality tolerance. Check that the major optimality tolerance is not too small.

`11 -- infeasible linear constraints`

`12 -- infeasible linear equality constraints`

When the constraints are linear, the output messages are based on a relatively reliable indicator of infeasibility. Feasibility is measured with respect to the upper and lower bounds on the variables and slacks. Among all the points satisfying the general constraints $Ax - s = 0$, there is apparently no point that satisfies the bounds on x and s. Violations as small as the minor feasibility tolerance are ignored, but at least one component of x or s violates a bound by more than the tolerance.

`13 -- nonlinear infeasibilities minimized`

`14 -- linear infeasibilities minimized`

`15 -- infeasible linear constraints in QP subproblem`

`16 -- infeasible nonelastic constraints`

When nonlinear constraints are present, infeasibility is *much* harder to recognize correctly. Even if a feasible solution exists, the current linearization of the constraints may not contain a feasible point. In an attempt to deal with this situation, when solving each QP subproblem, SNOPT is prepared to relax the bounds on the slacks associated with nonlinear rows.

If a QP subproblem proves to be infeasible or unbounded (or if the Lagrange multiplier estimates for the nonlinear constraints become large), SNOPT enters so-called "nonlinear elastic" mode. The subproblem includes the original QP objective and the sum of the infeasibilities—suitably weighted using the elastic weight parameter. In elastic mode, the nonlinear rows are made "elastic"—i.e., they are allowed to violate their specified bounds. Variables subject to elastic bounds are known as *elastic variables*. An elastic variable is free to violate one or both of its original upper or lower bounds. If the original problem has a feasible solution and the elastic weight is sufficiently large, a feasible point eventually will be obtained for the perturbed constraints, and optimization can continue on the subproblem. If the nonlinear problem has no feasible solution, SNOPT will tend to determine a "good" infeasible point if the elastic weight is sufficiently large. (If the elastic weight were infinite, SNOPT would locally minimize the nonlinear constraint violations subject to the linear constraints and bounds.)

Unfortunately, even though SNOPT locally minimizes the nonlinear constraint violations, there may still exist other regions in which the nonlinear constraints are satisfied. Wherever possible, nonlinear constraints should be defined in such a way that feasible points are known to exist when the constraints are linearized.

```
21 -- unbounded objective
```

```
22 -- constraint violation limit reached
```

For linear problems, unboundedness is detected by the simplex method when a nonbasic variable can apparently be increased or decreased by an arbitrary amount without causing a basic variable to violate a bound. Adding a bound on the objective will allow SNOPT to find a solution, and inspection of this solution will show the variables that can become too large due to missing restrictions.

Very rarely, the scaling of the problem could be so poor that numerical error will give an erroneous indication of unboundedness. Consider using the scale option.

For nonlinear problems, SNOPT monitors both the size of the current objective function and the size of the change in the variables at each step. If either of these is very large (see unbounded step size and unbounded objective value), the problem is terminated and declared UNBOUNDED. To avoid large function values, it may be necessary to impose bounds on some of the variables in order to keep them away from singularities in the nonlinear functions.

The second message indicates an abnormal termination while enforcing the limit on the constraint violations. This exit implies that the objective is not bounded below in the feasible region defined by expanding the bounds by the value of the violation limit.

```
31 -- iteration limit reached
```

An iteration limit was reached. Most often this is cured by increasing the GAMS iterlim option. If an SNOPT option file was used, also the iterations limit may have been set too small.

Check the iteration log to be sure that progress was being made. If so, repeat the run with higher limits. If not, consider specifying new initial values for some of the nonlinear variables.

```
32 -- major iteration limit reached
```

This indicates SNOPT was running out the limit on major iterations. This can be changed using the major iterations limit.

```
33 -- the superbasics limit is too small
```

The problem appears to be more nonlinear than anticipated. The current set of basic and superbasic variables have been optimized as much as possible and a PRICE operation is necessary to continue, but there are already as many superbasics as allowed (and no room for any more).

When increasing the superbasics limit, be aware that this also increases the reduced hessian dimension unless both options are set explicitly. This may increase the amount of memory required by SNOPT dramatically. Consider also increasing the amount memory available to SNOPT via the workfactor option, or setting a more moderate value for the reduced hessian dimension option and possibly getting slower convergence.

```
34 -- time limit reached
```

A time limit was hit. Increase the GAMS reslim option.

```
41 -- current point cannot be improved
```

The algorithm could not find a better solution although optimality was not achieved within the optimality tolerance. Possibly scaling can lead to better function values and derivatives. Raising the major optimality tolerance will probably make this message go away. Try better scaling, better bounds or a better starting point.

`42 -- singular basis`

The first factorization attempt found the basis to be structurally or numerically singular. (Some diagonals of the triangular matrix U were deemed too small.) The associated variables were replaced by slacks and the modified basis refactorized, but singularity persisted. Try better scaling, better bounds or a better starting point.

`43 -- cannot satisfy the general constraints`

The basic variables x_B have been recomputed, given the present values of the superbasic and nonbasic variables. A step of "iterative refinement" has also been applied to increase the accuracy of x_B, but a row check has revealed that the resulting solution does not satisfy the QP constraints $Ax - s = b$ sufficiently well. Try better scaling, better bounds or a better starting point.

`44 -- ill-conditioned null-space basis`

During computation of the reduced Hessian $Z^T HZ$, some column(s) of Z continued to contain very large values. Try better scaling, better bounds or a better starting point.

`51 -- incorrect objective derivatives`

`52 -- incorrect constraint derivatives`

The derivatives are not deemed to be correct. This message should not occur using a GAMS model without external equations.

`53 -- irregular or badly scaled problem functions`

The problem function can be badly scaled in case of DNLPs (which SNOPT was not designed for) or because of evaluations at singular points.

`61 -- undefined function at the first feasible point`

`62 -- undefined function at the initial point`

SNOPT was unable to proceed because the functions are undefined at the initial point or the first feasible point. Try to add better bounds or linear equations such that non-linear functions can be evaluated or use a better starting point.

`63 -- unable to proceed into undefined region`

Repeated attempts to move into a region where the functions are not defined resulted in the change in variables being unacceptably small. At the final point, it appears that the only way to decrease the merit function is to move into a region where the problem functions are not defined.

Try to add better bounds or linear equations such that non-linear functions can be evaluated or use a better starting point.

`71 -- terminated during function evaluation`

`72 -- terminated from monitor routine`

These messages indicate trouble evaluating the non-linear functions or derivatives. Usually these errors show a "Function evaluation error limit" message.

`81 -- work arrays must have at least 500 elements`

`82 -- not enough character storage`

`83 -- not enough integer storage`

`84 -- not enough real storage`

Increase the memory allocation by using workfactor > 1.

5.37 SoPlex

SoPlex (**S**equential **o**bject-oriented sim**Plex**) is an optimization package for solving linear programming problems (LPs) based on an advanced implementation of the primal and dual revised simplex algorithm. SoPlex is developed at the Zuse-Institute Berlin.

For more detailed information, we refer to [74] [127] [88] [89] [75] [206] and the SoPlex web site.

5.37.1 Usage

The following statement can be used inside your GAMS program to specify using SOPLEX

```
Option LP = SOPLEX;     { or RMIP }
```

The above statement should appear before the Solve statement. If SoPlex was specified as the default solver during GAMS installation, the above statement is not necessary.

5.37.1.1 Specification of SoPlex Options

GAMS/SoPlex supports the GAMS parameters reslim and iterlim.

To allow to run without an iteration limit (SoPlex default), the GAMS/SoPlex link does not pass on a setting of GAMS option iterlim to 2147483647.

Setting the GAMS option integer1 to a nonzero value enables writing of detailed solution statistics to the log.

Setting the GAMS option integer3 to a nonzero value leads to writing the model instance to a file in LP or MPS format before starting the solution process (`integer3=1` writes an MPS file, `integer3=2` writes an LP files, `integer3=4` writes SoPlex state files (.mps, .bas, .set); sum these values to write several files). The name of the output file is chosen to be the name of the GAMS model file with the extension `.gms` replaced. Setting the GAMS option integer2 to a nonzero value makes variable and equation names available when writing the LP or MPS files. These options may be useful for debugging purposes.

Options can be specified by a SoPlex options file. A SoPlex options file consists of one option or comment per line. A pound sign (`#`) at the beginning of a line causes the entire line to be ignored. Otherwise, the line will be interpreted as an option name and value separated by an equal sign (`=`) and any amount of white space (blanks or tabs).

A small example for a soplex.opt file is:

```
bool:rowboundflips = true
int:algorithm = 0
real:feastol = 1e-5
```

It causes GAMS/SoPlex to use bound flipping also for row representations, use the primal simplex, and use a primal feasibility tolerance of 1e-5.

5.37.2 List of SoPlex Options

In the following, we give a detailed list of all SoPlex options.

Option	Description	Default
bool:acceptcycling	should cycling solutions be accepted during iterative refinement? Range: boolean	0
bool:computedegen	should the degeneracy be computed for each basis? Range: boolean	0
bool:decompositiondualsimplex	should the decomposition based dual simplex be used to solve the LP? Range: boolean	0
bool:ensureray	re-optimize the original problem to get a proof (ray) of infeasibility/unboundedness? Range: boolean	0
bool:explicitviol	Should violations of the original problem be explicitly computed in the decomposition simplex? Range: boolean	0
bool:forcebasic	try to enforce that the optimal solution is a basic solution Range: boolean	0
bool:fullperturbation	should perturbation be applied to the entire problem? Range: boolean	0
bool:lifting	should lifting be used to reduce range of nonzero matrix coefficients? Range: boolean	0
bool:persistentscaling	should persistent scaling be used? Range: boolean	1
bool:powerscaling	round scaling factors for iterative refinement to powers of two? Range: boolean	1
bool:ratfacjump	continue iterative refinement with exact basic solution if not optimal? Range: boolean	0
bool:rowboundflips	use bound flipping also for row representation? Range: boolean	0
bool:testdualinf	should dual infeasibility be tested in order to try to return a dual solution even if primal infeasible? Range: boolean	0
bool:usecompdual	should the dual of the complementary problem be used in the decomposition simplex? Range: boolean	0

Option	Description	Default
int:algorithm	type of algorithm (0 - primal, 1 - dual) Range: {0, ..., 1}	1
int:decomp_displayfreq	the frequency that the decomposition based simplex status output is displayed. Range: {1, ..., ∞}	50
int:decomp_iterlimit	the number of iterations before the decomposition simplex initialisation solve is terminated Range: {1, ..., ∞}	100
int:decomp_maxaddedrows	maximum number of rows that are added to the reduced problem when using the decomposition based simplex Range: {1, ..., ∞}	500
int:decomp_verbosity	the verbosity of decomposition based simplex (0 - error, 1 - warning, 2 - debug, 3 - normal, 4 - high, 5 - full). Range: {1, ..., 5}	0
int:displayfreq	display frequency Range: {1, ..., ∞}	200
int:factor_update_max	maximum number of LU updates without fresh factorization (0 - auto) Range: {0, ..., ∞}	0
int:factor_update_type	type of LU update (0 - eta update, 1 - Forrest-Tomlin update) Range: {0, ..., 1}	1
int:hyperpricing	mode for hyper sparse pricing (0 - off, 1 - auto, 2 - always) Range: {0, ..., 2}	1
int:iterlimit	iteration limit (-1 - no limit) Range: {-1, ..., ∞}	GAMS iterlim
int:leastsq_maxrounds	maximum number of conjugate gradient iterations in least square scaling Range: {0, ..., ∞}	50
int:pricer	pricing method (0 - auto, 1 - dantzig, 2 - parmult, 3 - devex, 4 - quicksteep, 5 - steep) Range: {0, ..., 5}	0
int:printbasismetric	print basis metric during the solve (-1 - off, 0 - condition estimate , 1 - trace, 2 - determinant, 3 - condition) Range: {-1, ..., 3}	-1
int:ratiotester	method for ratio test (0 - textbook, 1 - harris, 2 - fast, 3 - boundflipping) Range: {0, ..., 3}	3

Option	Description	Default
int:reflimit	refinement limit (-1 - no limit) Range: {-1, ..., ∞}	-1
int:representation	type of computational form (0 - auto, 1 - column representation, 2 - row representation) Range: {0, ..., 2}	0
int:scaler	scaling (0 - off, 1 - uni-equilibrium, 2 - bi-equilibrium, 3 - geometric, 4 - iterated geometric, 5 - least squares, 6 - geometric-equilibrium) Range: {0, ..., 6}	2
int:simplifier	simplifier (0 - off, 1 - auto, 2 - PaPILO, 3 - internal) Range: {0, ..., 3}	3
int:stallreflimit	stalling refinement limit (-1 - no limit) Range: {-1, ..., ∞}	-1
int:starter	crash basis generated when starting from scratch (0 - none, 1 - weight, 2 - sum, 3 - vector) Range: {0, ..., 3}	0
int:stattimer	measure for statistics, e.g. factorization time (0 - off, 1 - user time, 2 - wallclock time) Range: {0, ..., 2}	1
int:timer	type of timer (1 - cputime, aka. usertime, 2 - wallclock time, 0 - no timing) Range: {0, ..., 2}	2
int:verbosity	verbosity level (0 - error, 1 - warning, 2 - debug, 3 - normal, 4 - high, 5 - full) Range: {0, ..., 5}	3
real:epsilon_factorization	zero tolerance used in factorization Range: [0, 1]	1e-20
real:epsilon_pivot	pivot zero tolerance used in factorization Range: [0, 1]	1e-10
real:epsilon_update	zero tolerance used in update of the factorization Range: [0, 1]	1e-16
real:epsilon_zero	general zero tolerance Range: [0, 1]	1e-16
real:feastol	primal feasibility tolerance Range: [0, 1]	1e-06
real:fpfeastol	working tolerance for feasibility in floating-point solver during iterative refinement Range: [1e-12, 1]	1e-09

Option	Description	Default
real:fpopttol	working tolerance for optimality in floating-point solver during iterative refinement Range: [1e-12, 1]	1e-09
real:infty	infinity threshold Range: [1e+10, ∞]	∞
real:leastsq_acrcy	accuracy of conjugate gradient method in least squares scaling (higher value leads to more iterations) Range: [1, ∞]	1000
real:liftmaxval	lower threshold in lifting (nonzero matrix coefficients with smaller absolute value will be reformulated) Range: [10, ∞]	1024
real:liftminval	lower threshold in lifting (nonzero matrix coefficients with smaller absolute value will be reformulated) Range: [0, 0.1]	0.000976562
real:maxscaleincr	maximum increase of scaling factors between refinements Range: [1, ∞]	1e+25
real:min_markowitz	minimal Markowitz threshold in LU factorization Range: [0.0001, 0.9999]	0.01
real:minred	minimal reduction (sum of removed rows/cols) to continue simplification Range: [0, 1]	0.0001
real:objlimit_lower	lower limit on objective value Range: real	GAMS cutoff, if maximizing, else -∞
real:objlimit_upper	upper limit on objective value Range: real	GAMS cutoff, if minimizing, else +∞
real:opttol	dual feasibility tolerance Range: [0, 1]	1e-06
real:refac_basis_nnz	refactor threshold for nonzeros in last factorized basis matrix compared to updated basis matrix Range: [1, 100]	10
real:refac_mem_factor	refactor threshold for memory growth in factorization since last refactorization Range: [1, 10]	1.5
real:refac_update_fill	refactor threshold for fill-in in current factor update compared to fill-in in last factorization Range: [1, 100]	5

Option	Description	Default
real:representation_switch	threshold on number of rows vs. number of columns for switching from column to row representations in auto mode Range: [0, ∞]	1.2
real:simplifier_modifyrowfac	modify constraints when the number of nonzeros or rows is at most this factor times the number of nonzeros or rows before presolving Range: [0, 1]	1
real:sparsity_threshold	sparse pricing threshold (violations < dimension ∗ SPARSITY_THRESHOLD activates sparse pricing) Range: [0, 1]	0.6
real:timelimit	time limit in seconds Range: [0, ∞]	GAMS reslim

5.38 XPRESS

5.38.1 Introduction

The GAMS/XPRESS solver is based on the XPRESS Optimization Subroutine Library, and runs only in conjunction with the GAMS modeling system. GAMS/XPRESS (also simply referred to as XPRESS) is a versatile, high-performance optimization system. The system integrates:

- a powerful simplex-based LP solver.
- a MIP module with cut generation for integer programming problems.
- a barrier module implementing a state-of-the-art interior point algorithm for very large LP problems.
- a sequential linear programming solver (SLP) for (mixed-integer) nonlinear programs NLP, CNS and MINLP.

The GAMS/XPRESS solver is installed automatically with your GAMS system. There are different license options:

- **GAMS/XPRESS:**
  Continuous and discrete linear and convex quadratic models.
- **GAMS/XPRESS-NLP:**
  Continuous linear, quadratic and nonlinear models. In order to use GAMS Knitro, a GAMS/KNITRO license must be included.
- **GAMS/XPRESS-MINLP:**
  All model types. In order to use GAMS Knitro, a GAMS/KNITRO license must be included.
- **GAMS/XPRESS Link:**
  Users must have a separate, licensed XPRESS system. For users who wish to use XPRESS within GAMS and also in other environments. All model types, if XPRESS license enables it. In order to use GAMS Knitro, the user's Xpress license must enable it.
- **demo:**
  Like GAMS/XPRESS but with demo limits (small models only).

Attention

The free bare-bone link mode (previously GAMS/OSIXPRESS) that allowed to solve LP and MIP when the user had a separate XPRESS license installed has been removed. If you relied on using this bare-bone link option, then do not hesitate to contact sales@gams.com to arrange for a GAMS/XPRESS Link license.

5.38.2 Usage

To explicitly request that a model be solved with XPRESS, insert the statement

```
option LP = xpress;  { or MIP, RMIP, NLP, CNS, DNLP, RMINLP, MINLP, QCP, MIQCP, or RMIQCP }
```

somewhere before the solve statement. If XPRESS has been selected as the default solver (e.g. during GAMS installation) for the model type in question, the above statement is not necessary.

The standard GAMS options (e.g. iterlim, optcr) can be used to control XPRESS. For more details, see section Controlling a Solver via GAMS Options. Please note however that - apart from reslim - these are only used for linear and quadratic programs and not the nonlinear solves. Termination conditions for XPRESS SLP can be set in SLP Termination Options.

In addition, XPRESS-specific options can be specified by using a solver option file. While the content of an option file is solver-specific, the details of how to create an option file and instruct the solver to use it are not. This topic is covered in section The Solver Options File.

An example of a valid XPRESS option file is:

```
* sample XPRESS options file
algorithm simplex
presolve   0
IterLim    50000
```

In general this is enough knowledge to solve your models. In some cases you may want to use some of the XPRESS options to gain further performance improvements or for other reasons.

5.38.2.1 Linear and Quadratic Programming

The options advBasis, lpFlags, defaultAlg, basisOut, mpsOutputFile, reform, reRun, and reslim control the behavior of the GAMS/XPRESS link. The options crash, lpIterlimit, presolve, scaling, threads, and trace set XPRESS library control variables, and can be used to fine-tune XPRESS. See section General LP / MIP / QP Options for more details of XPRESS general options.

LP

See section LP Options for more details of XPRESS library control variables which can be used to fine-tune the XPRESS LP solver.

MIP

In some cases, the branch-and-bound MIP algorithm will stop with a proven optimal solution or when unboundedness or (integer) infeasibility is detected. In most cases, however, the global search is stopped through one of the generic GAMS options:

1. iterlim (on the cumulative pivot count) or reslim (in seconds of CPU time),
2. optca & optcr (stopping criteria based on gap between best integer solution found and best possible) or
3. nodlim (on the total number of nodes allowed in the B&B tree).

It is also possible to set the maxNode and maxMipSol options to stop the global search: see section MIP Options for XPRESS control variables for MIP. The options loadMipSol, mipCleanup, mipTrace, mipTraceNode, and mipTraceTime control the behavior of the GAMS/XPRESS link on MIP models. The other options in section MIP Options set XPRESS library control variables, and can be used to fine-tune the XPRESS MIP solver.

MIP Solution Pool

Typically, XPRESS finds a number of integer feasible points during its global search, but only the final solution is available. The MIP solution pool capability makes it possible to store multiple integer feasible points (aka solutions) for later processing. The MIP solution pool operates in one of two modes: by default `(solnpoolPop = 1)` the global search is not altered, but with `(solnpoolPop = 2)` a selected set (potentially all) of the integer feasible solutions are enumerated.

The MIP enumeration proceeds until all MIP solutions are enumerated or cut off, or until a user-defined limit is reached. Whenever a new solution is generated by the enumerator, it is presented to the solution pool manager. If there is room in the pool, the new solution is added. If the pool is full, a *cull round* is performed to select a number of solutions to be thrown out - these solutions can be those stored in the pool and/or the new solution. Solutions can be selected for culling based on their MIP objective value and/or the overall diversity of the solutions in the pool. If neither is chosen, a default choice is made to throw out one solution based on objective values. Whenever a solution is thrown out based on its MIP objective, the enumeration space is pruned based on the cutoff defined by this objective value.

By default, the capacity of the pool is set very large, as is the number of cull rounds to perform, so selecting only `solnpoolPop = 2` will result in full enumeration. However, many different strategies can be executed by setting the solution pool options. For example, to choose the N-best solutions, simply set the solution pool capacity to N. When the pool is full, new solutions will force a cull round, and the default is to reject one solution based on its objective and update the cutoff accordingly. To generate all solutions with an objective as good as X, leave the pool capacity set at a high level but set the cutoff to X using the `mipabscutoff` option. To return the N-first solutions, set the solution pool capacity to N and `solnpoolCullRounds = 0`: as soon as the pool is full the enumeration will stop on the cull round limit.

A number of other strategies for controlling the solution pool behavior are possible by combining different options. Several working examples are provided in the GAMS Test Library in models `xpress03.gms`, `xpress04.gms`, and `xpress05.gms`.

See section MIP Solution Pool Options for XPRESS control variables for MIP Solution Pool.

Newton-Barrier

The barrier method is invoked by default for quadratic problems, and can be selected for linear models by using one of the options

```
algorithm       barrier
defaultalg      4
```

The barrier method is likely to use more memory than the simplex method. No warm start is done, so if an advanced basis exists, you may not wish to use the barrier solver.

See section Newton-barrier Options for XPRESS control variables for the Newton-Barrier method.

5.38.2.2 Nonlinear Programming

XPRESS can solve nonlinear programs of type NLP, CNS and MINLP (and its relaxed version) using the sequential linear programming solver XPRESS SLP or the interior-point / sequential quadratic programming solver XPRESS Knitro. Convexity is not required, but for non-convex programs XPRESS will in general find local optimal solutions only. The XPRESS multistart can be used to increase the likelihood of finding a good solution by starting from many different initial points.

XPRESS SLP solves nonlinear programs by successive linearization of the nonlinearities. These linearizations, which can be controlled by the options in NLP Augmentation and Linearization Options, are solved by the LP or QCP solver. Therefore, XPRESS user options for LP or QCP are also relevant when solving nonlinear programs. Note, that the NLP Presolve is independent from the LP presolve that is executed in each XPRESS SLP iteration.

Termination

In most cases it is sufficient to control the termination of XPRESS SLP by xslp_iterLimit, reslim, xslp_validationTarget_k and xslp_ValidationTarget_r as the latter two automatically control the other XPRESS SLP convergence measures on default. More experienced users may want to modify the other convergence measures, which group into:

- Strict convergence: Describes the numerical behaviour of convergence in the formal, mathematical sense. User options: xslp_cTol, xslp_aTol_a, xslp_aTol_r.
- Extended convergence: Measures the quality of the linearization, including the effect of changes to the nonlinear terms that contribute to a variable in the linearization: User options: xslp_mTol_a, xslp_mTol_r, xslp_iTol_a, xslp_iTol_r, xslp_sTol_a, xslp_sTol_r.

When each variable has converged in one of the above cases, XPRESS SLP terminates based on the following stopping criteria:

- Baseline static objective convergence measure that compares changes in the objective over a given number of iterations relative to the average objective value. User options: xslp_vCount, xslp_vLimit, xslp_vTol_a, xslp_vTol_r.
- Static objective convergence measure that is applied when there are no unconverged variables in active constraints. User options: slpOCount, xslp_oTol_a, xslp_oTol_r.
- Static objective convergence measure that is applied when a pratical solution (all variables have converged and there are no active step bounds) has been found. User options: xslp_xCount, xslp_xLimit, xslp_xTol_a, xslp_xTol_r.
- Extended convergence continuation that is applied when a practical solution has been found. It checks if it is worth continuing. User options: xslp_wCount, xslp_wTol_a, xslp_wTol_r.

The user option slpConvergenceOps enables/disables the different convergence measures and stopping conditions.

Output

The output or logging can be controlled by the NLP Log Options. The default XPRESS SLP iteration output shows:

- `It`: Iteration number.
- `LP`: The LP status of the linearization (`O`: optimal; `I`: infeasible; `U`: unbounded; `X`: interrupted)
- `NetObj`: The net objective of the SLP iteration.
- `ValObj`: The original objective function value.
- `ErrorSum`: Sum of the error delta variables. A measure of infeasibility.
- `ErrorCost`: The value of the weighted error delta variables in the objective. A measure of the effort needed to push the model towards feasibility.
- `Validate`: Relative feasibility measure (calculated only if convergence is likely)
- `KKT`: Relative optimality measure (calculated only if convergence is likely)
- `Unconv`: The number of SLP variables that are not converged.
- `Ext`: The number of SLP variables that are converged, but only by extended criteria.
- `Action`: Special actions (`O`: failed line search; `B`: enforcing step bounds; `E`: some infeasible rows were enforced; `G`: global variables were fixed; `P`: solution needed polishing, postsolve instability; `P!`: solution polishing failed; `R`: penalty error vectors removed; `V`: feasibility validation induces further iterations; `K`: optimality validation induces further iterations)
- `T`: Time.

XPRESS Knitro

Nonlinear programs can also be solved by XPRESS Knitro using the option xslp_solver. In this case, the nonlinear program is passed to Knitro after the NLP Presolve. Setting xslp_solver to auto will enable XPRESS to choose XPRESS SLP or XPRESS Knitro automatically based on the problem instance. XPRESS Knitro options can be specified in a Knitro solver option file, which needs to be selected in knitroOptFile. Note, that XPRESS Knitro does not support all Knitro options. Specified but unsupported options trigger a warning and are then ignored.

For more information about this nonlinear programming solver, see the GAMS/Knitro documentation.

5.38.3 Summary of XPRESS Options

5.38.3.1 General LP / MIP / QP Options

Option	Description	Default
advBasis	Use advanced basis provided by GAMS	`auto`
algorithm	Choose between simplex and barrier algorithm	`simplex`
basisOut	Directs optimizer to output an MPS basis file	`none`
clamping	Allows for the adjustment of returned solution values such that they are always within bounds	`0`

Option	Description	Default
clamping_dual	Adjust primal slack values to always be within constraint bounds	0
clamping_primal	Adjust primal solution to always be within primal bounds	0
clamping_rdj	Adjust reduced costs to always be within dual bounds implied by the primal solution	0
clamping_slacks	Adjust dual solution to always be within the dual bounds implied by the slacks	0
cpuTime	How time should be measured when timings are reported in the log and when checking against time limits	0
globalBoundingBox	If a nonlinear problem cannot be solved due to appearing unbounded, it can automatically be regularized by the application of a bounding box on the variables	1e+06
inputTol	Tolerance on input values elements	0
ioTimeout	Maximum number of seconds to wait for an I/O operation before it is cancelled	30
lpIterLimit	Maximum number of iterations that will be performed by primal simplex or dual simplex before the optimization process terminates	maxint
lpRefineIterLimit	Simplex iteration limit the solution refiner can spend in attempting to increase the accuracy of an LP solution	auto
maxScaleFactor	Determines the maximum scaling factor that can be applied during scaling	64
mpsNameLength	Maximum length of MPS names in characters	0
mpsOutputFile	Name of MPS output file	none
numericalEmphasis	How much emphasis to place on numerical stability instead of solve speed	auto
outputControls	Toggles the printing of all control settings at the beginning of the search	1
outputLog	Controls the level of output produced by the Optimizer during optimization	1
outputTol	Zero tolerance on print values	1e-05
qextractalg	quadratic extraction algorithm in GAMS interface	0
randomSeed	Sets the initial seed to use for the pseudo-random number generator in the Optimizer	1
refineOps	Specifies when the solution refiner should be executed to reduce solution infeasibilities	19
refineOps_iterativeRefiner	Apply the iterative refiner to refine the solution	0
refineOps_lpOptimal	Run the solution refiner on an optimal solution of a continuous problem	1
refineOps_lpPresolve	Run the solution refiner on an optimal solution before postsolve on a continuous problem	1
refineOps_mipFixGlobals	Refine MIP solutions such that rounding them keeps the problem feasible when reoptimized	0
refineOps_mipFixGlobalsTarget	Attempt to refine MIP solutions such that rounding them keeps the problem feasible when reoptimized, but accept integers solutions even if refinement fails	0
refineOps_mipNodeLp	Run the solution refiner on each node of the MIP search	0

Option	Description	Default
refineOps_mipSolution	Run the solution refiner when a new solution is found during a tree search	1
refineOps_refinerPrecision	Use higher precision in the iterative refinement	0
refineOps_refinerUseDual	If set, the iterative refiner will use the dual simplex algorithm	0
refineOps_refinerUsePrimal	If set, the iterative refiner will use the primal simplex algorithm	0
reform	Substitute out objective var and equ when possible	1
reRun	Rerun with primal simplex when not optimal/feasible	0
reslim	Overrides GAMS reslim option	
solTimeLimit	Maximum time in seconds that the Optimizer will run a MIP solve before it terminates, given that a solution has been found	1e+20
timeLimit	Maximum time in seconds that the Optimizer will run before it terminates, including the problem setup time and solution time	1e+20
trace	Display the infeasibility diagnosis during presolve	0
writePrtSol	Directs optimizer to output a "printsol" file	0

5.38.3.2 Hardware Related Options

Option	Description	Default
barCores	If set to a positive integer it determines the number of physical CPU cores assumed to be present in the system by the barrier algorithm	auto
barOrderThreads	If set to a positive integer it determines the number of concurrent threads for the sparse matrix ordering algorithm in the Newton-barrier method	auto
barThreads	If set to a positive integer it determines the number of threads implemented to run the Newton-barrier algorithm	auto
concurrentThreads	Determines the number of threads used by the concurrent solver	auto
coresPerCpu	Used to override the detected value of the number of cores on a CPU	auto
cpuPlatform	Newton Barrier: Selects the AMD, Intel x86 or ARM vectorization instruction set that Barrier should run optimized code for	-1
crossoverThreads	Determines the maximum number of threads that parallel crossover is allowed to use	auto
dualThreads	Determines the maximum number of threads that dual simplex is allowed to use	auto
heurThreads	Branch and Bound: Number of threads to dedicate to running heuristics on the root node	0
maxMemoryHard	Sets the maximum amount of memory in megabytes the optimizer should allocate	unlimited
maxMemorySoft	When resourceStrategy is enabled, this control sets the maximum amount of memory in megabytes the optimizer targets to allocate	unlimited
mipThreads	If set to a positive integer it determines the number of threads implemented to run the parallel MIP code	auto

Option	Description	Default
relaxTreeMemoryLimit	When the memory used by the branch and bound search tree exceeds the target specified by the treeMemoryLimit control, the optimizer will try to reduce this by writing nodes to the tree file	0.1
resourceStrategy	Controls whether the optimizer is allowed to make nondeterministic decisions if memory is running low in an effort to preserve memory and finish the solve	0
threads	Default number of threads used during optimization	1
treeCompression	When writing nodes to the global file, the optimizer can try to use data-compression techniques to reduce the size of the tree file on disk	2
treeMemoryLimit	Soft limit, in megabytes, for the amount of memory to use in storing the branch and bound search tree	auto
treeMemorySavingTarget	When the memory used by the branch-and-bound search tree exceeds the limit specified by the treeMemoryLimit control, the optimizer will try to save memory by writing lower-rated sections of the tree to the tree file	0.4

5.38.3.3 Presolve Options

Option	Description	Default
barPresolveOps	Newton barrier: Controls the Newton-Barrier specific presolve operations	0
barPresolveOps_extra	Extra effort is spent in barrier specific presolve	0
barPresolveOps_full	Do full matrix eliminations (reduce matrix size)	0
barPresolveOps_standard	Use standard presolve	0
dualize	Whether presolve should form the dual of the problem	auto
dualizeOps	Bit-vector control for adjusting the behavior when a problem is dualized	1
elimFillin	Amount of fill-in allowed when performing an elimination in presolve	10
elimTol	Markowitz tolerance for the elimination phase of the presolve	0.001
indLinBigM	During presolve, indicator constraints will be linearized using a BigM coefficient whenever that BigM coefficient is small enough	100000
indPreLinBigM	During presolve, indicator constraints will be linearized using a BigM coefficient whenever that BigM coefficient is small enough	100
lpFolding	Simplex and barrier: Whether to fold an LP problem before solving it	auto
maxImpliedBound	Presolve: When tighter bounds are calculated during MIP preprocessing, only bounds whose absolute value are smaller than maxImpliedBound will be applied to the problem	1e+08
mipPresolve	Branch and Bound: Type of integer processing to be performed	-1
mipPresolve_allowChangeBounds	If node preprocessing is allowed to change bounds on continuous columns	1

Option	Description	Default
mipPresolve_allowTreeRestart	[Unused] This bit is no longer used to control restarts	1
mipPresolve_dualReductions	Dual reductions will be performed at each node	1
mipPresolve_globalCoefTightening	Allow global (non-bound) tightening of the problem during the tree search	1
mipPresolve_logicPreprocessing	Primal reductions will be performed at each node	1
mipPresolve_objBasedReductions	Objective function will be used to find reductions at each node	1
mipPresolve_reducedCostFixing	Reduced cost fixing will be performed at each node	1
mipPresolve_symmetryReductions	Allow that symmetry is used to presolve the node problem	1
preAnalyticCenter	Determines if analytic centers should be computed and used for variable fixing and the generation of alternative reduced costs (-1: Auto 0: Off, 1: Fixing, 2: Redcost, 3: Both)	auto
preBasisRed	Determines if a lattice basis reduction algorithm should be attempted as part of presolve	0
preBndRedCone	Determines if second order cone constraints should be used for inferring bound reductions on variables when solving a MIP	auto
preBndRedQuad	Determines if convex quadratic contraints should be used for inferring bound reductions on variables when solving a MIP	auto
preCliqueStrategy	Determines how much effort to spend on clique covers in presolve	-1
preCoefElim	Presolve: Specifies whether the optimizer should attempt to recombine constraints in order to reduce the number of non zero coefficients when presolving a mixed integer problem	2
preComponents	Presolve: Determines whether small independent components should be detected and solved as individual subproblems during root node processing	auto
preComponentsEffort	Presolve: Adjusts the overall effort for the independent component presolver	1
preConeDecomp	Presolve: Decompose regular and rotated cones with more than two elements and apply Outer Approximation on the resulting components	auto
preConfiguration	MIP Presolve: Determines whether binary rows with only few repeating coefficients should be reformulated	auto
preConvertSeparable	Presolve: Reformulate problem with non-diagonal quadratic objective and/or constraints as diagonal quadratic or second-order conic constraints	auto
preDomCol	Presolve: Determines the level of dominated column removal reductions to perform when presolving a mixed integer problem	auto
preDomRow	Presolve: Determines the level of dominated row removal reductions to perform when presolving a problem	auto
preDupRow	Presolve: Determines the type of duplicate rows to look for and eliminate when presolving a problem	auto

Option	Description	Default
preElimQuad	Presolve: Allows for elimination of quadratic variables via doubleton rows	auto
preFolding	Presolve: Determines if a folding procedure should be used to aggregate continuous columns in an equitable partition	auto
preImplications	Presolve: Determines whether to use implication structures to remove redundant rows	auto
preLinDep	Presolve: Determines whether to check for and remove linearly dependent equality constraints when presolving a problem	auto
preObjCutDetect	Presolve: Determines whether to check for constraints that are parallel or near parallel to a linear objective function, and which can safely be removed	1
preProbing	Presolve: Amount of probing to perform on binary variables during presolve	auto
presolve	Determines whether presolving should be performed prior to starting the main algorithm	1
presolveMaxGrow	Limit on how much the number of non-zero coefficients is allowed to grow during presolve, specified as a ratio of the number of non-zero coefficients in the original problem	0.1
presolveOps	Specifies the operations which are performed during the presolve	511
presolveOps_dualReductions	Dual reductions	1
presolveOps_duplicateColRemoval	Duplicate column removal	1
presolveOps_duplicateRowRemoval	Duplicate row removal	1
presolveOps_forcingRowRemoval	Forcing row removal	1
presolveOps_linDependRowRemoval	Linearly dependant row removal	0
presolveOps_noAdvIpReductions	No advanced IP reductions	0
presolveOps_noGlobalDomainChange	No semi-continuous variable detection	0
presolveOps_noIntVarAndSosDetect	No integer variable and SOS detection	0
presolveOps_noIntVarEliminations	No eliminations on integers	0
presolveOps_noIpReductions	No IP reductions	0
presolveOps_redundantRowRemoval	Redundant row removal	1
presolveOps_singletonColRemoval	Singleton column removal	1
presolveOps_singletonRowRemoval	Singleton row removal	1
presolveOps_strongDualReductions	Strong dual reductions	1
presolveOps_variableEliminations	Variable eliminations	1
presolvePasses	Number of reduction rounds to be performed in presolve	1

Option	Description	Default
rootPresolve	Determines if presolving should be performed on the problem after the tree search has finished with root cutting and heuristics	auto
siftPresolveOps	Determines the presolve operations for solving the subproblems during the sifting algorithm	-1

5.38.3.4 Scaling Options

Option	Description	Default
autoScaling	Whether the Optimizer should automatically select between different scaling algorithms	auto
barFreeScale	Defines how the barrier algorithm scales free variables	1e-06
barObjScale	Defines how the barrier scales the objective	auto
barRhsScale	Defines how the barrier scales the right hand side	auto
objScaleFactor	Custom objective scaling factor, expressed as a power of 2	0
scaling	Determines how the Optimizer will rescale a model internally before optimization	163
scaling_beforePresolve	Scale before presolve	0
scaling_bigM	Treat big-M rows as normal rows	0
scaling_byMaxElemNotGeoMean	0: scale by geometric mean	1
scaling_colScaling	Column scaling	1
scaling_curtisReid	Curtis-Reid	0
scaling_disableGlobalObjScaling	Do not apply automatic objective scaling	0
scaling_ignoreQuadRowPart	Exclude the quadratic part of constraint when calculating scaling factors	0
scaling_maximum	Maximum	0
scaling_noAggressiveQScaling	Disable aggressive quadratic scaling	0
scaling_noScalingColsDown	Do not scale columns down	0
scaling_noScalingRowsUp	Do not scale rows up	0
scaling_rhsScaling	RHS scaling	0
scaling_rowScaling	Row scaling	1
scaling_rowScalingAgain	Row scaling again	0
scaling_simplexObjScaling	Scale objective function for the simplex method	1
scaling_slackScaling	Enable explicit linear slack scaling	0

5.38.3.5 LP Options

Option	Description	Default
algAfterNetwork	Algorithm to be used for the clean up step after the network simplex solver	auto
autoPerturb	Simplex: Indicates whether automatic perturbation is performed	auto
bigM	Infeasibility penalty used if the "Big M" method is implemented	auto
bigMMethod	Simplex: Whether to use the "Big M" method, or the standard phase I (achieving feasibility) and phase II (achieving optimality)	1
crash	Simplex: Determines the type of crash used when the algorithm begins	2
defaultAlg	Selects the algorithm that will be used to solve the LP	auto
dualGradient	Simplex: Dual simplex pricing method	auto
dualPerturb	Factor by which the problem will be perturbed prior to optimization by dual simplex	auto
dualStrategy	Bit-vector control specifies the dual simplex strategy	1
etaTol	Tolerance on eta elements	1e-13
feasTol	Determines when a solution is treated as feasible	1e-06
feasTolPerturb	Determines how much a feasible primal basic solution is allowed to be perturbed when performing basis changes	1e-06
feasTolTarget	Target feasibility tolerance for the solution refiner	0
forceParallelDual	Dual simplex: Specifies whether the dual simplex solver should always use the parallel simplex algorithm	0
invertFreq	Simplex: Frequency with which the basis will be inverted	auto
invertMin	Simplex: Minimum number of iterations between full inversions of the basis matrix	3
lpFlags	Bit-vector control which defines the algorithm for solving an LP problem or the initial LP relaxation of a MIP problem	0
lpFlags_barrier	Use the barrier method	0
lpFlags_dual	Use the dual simplex method	0
lpFlags_network	Use the network simplex method	0
lpFlags_primal	Use the primal simplex method	0
lpLog	Simplex: Frequency at which the simplex log is printed	100
lpLogDelay	Time interval between two LP log lines	1
lpLogStyle	Simplex: Style of the simplex log	1
markowitzTol	Markowitz tolerance used for the factorization of the basis matrix	0.01
matrixTol	Zero tolerance on matrix elements	1e-09
netStallLimit	Limit the number of degenerate pivots of the network simplex algorithm, before switching to either primal or dual simplex, depending on algAfterNetwork	auto
optimalityTol	Simplex: Zero tolerance for reduced costs	1e-06
optimalityTolTarget	Target optimality tolerance for the solution refiner	0

Option	Description	Default
penalty	Minimum absolute penalty variable coefficient	auto
pivotTol	Simplex: Zero tolerance for matrix elements	1e-09
ppFactor	Partial pricing candidate list sizing parameter	1
pricingAlg	Simplex: Determines the primal simplex pricing method	auto
primalOps	Primal simplex: Allows fine tuning the variable selection in the primal simplex solver	-1
primalPerturb	Factor by which the problem will be perturbed prior to optimization by primal simplex	auto
primalUnshift	Determines whether primal is allowed to call dual to unshift	0
relPivotTol	Simplex: Minimum size of pivot element relative to largest element in column	1e-06
sifting	Determines whether to enable sifting algorithm with the dual simplex method	auto
siftPasses	Determines how quickly we allow to grow the worker problems during the sifting algorithm	4
siftSwitch	Determines which algorithm to use for solving the subproblems during sifting	-1

5.38.3.6 QP Options

Option	Description	Default
eigenvalueTol	Quadratic matrix is considered not to be positive semi-definite, if its smallest eigenvalue is smaller than the negative of this value	1e-06
ifCheckConvexity	Determines if the convexity of the problem is checked before optimization	1
qSimplexOps	Controls the behavior of the quadratic simplex solvers	0
quadraticUnshift	Determines whether an extra solution purification step is called after a solution found by the quadratic simplex (either primal or dual)	auto
repairIndefInitEq	Controls if the optimizer should make indefinite quadratic matrices positive definite when it is possible	1

5.38.3.7 Newton-barrier Options

Option	Description	Default
algAfterCrossover	Algorithm to be used for the final clean up step after the crossover	auto
barAlg	Determines which barrier algorithm is to be used to solve the problem	auto
barCrash	Newton barrier: Determines the type of crash used for the crossover	4
barDualStop	Newton barrier: Convergence parameter, representing the tolerance for dual infeasibilities	auto
barFailIterLimit	Newton barrier: Maximum number of consecutive iterations that fail to improve the solution in the barrier algorithm	auto
barGapStop	Newton barrier: Convergence parameter, representing the tolerance for the relative duality gap	auto

Option	Description	Default
barGapTarget	Newton barrier: Target tolerance for the relative duality gap	auto
barIndefLimit	Newton Barrier: Limits the number of consecutive indefinite barrier iterations that will be performed	15
barIterLimit	Newton barrier: Maximum number of iterations	500
barKernel	Newton barrier: Defines how centrality is weighted in the barrier algorithm	0
barObjPerturb	Defines how the barrier perturbs the objective	1e-06
barOrder	Newton barrier: Controls the Cholesky factorization in the Newton-Barrier	auto
barOutput	Newton barrier: Level of solution output provided	1
barPerturb	Newton barrier: In numerically challenging cases it is often advantageous to apply perturbations on the KKT system to improve its numerical properties	0
barPrimalStop	Newton barrier: Convergence parameter, indicating the tolerance for primal infeasibilities	auto
barRefIter	Newton barrier: After terminating the barrier algorithm, further refinement steps can be performed	0
barRegularize	Determines how the barrier algorithm applies regularization on the KKT system	-1
barStart	Newton barrier: Controls the computation of the starting point for the barrier algorithm	auto
barStepStop	Newton barrier: Convergence parameter, representing the minimal step size	1e-16
choleskyAlg	Newton barrier: Type of Cholesky factorization used	auto
choleskyTol	Newton barrier: Tolerance for pivot elements in the Cholesky decomposition of the normal equations coefficient matrix, computed at each iteration of the barrier algorithm	1e-15
crossover	Newton barrier: Determines whether the barrier method will cross over to the simplex method when at optimal solution has been found, to provide an end basis and advanced sensitivity analysis information	auto
crossoverAccuracyTol	Newton barrier: Determines how crossover adjusts the default relative pivot tolerance	1e-06
crossoverIterLimit	Newton barrier: Maximum number of iterations that will be performed in the crossover procedure before the optimization process terminates	2147483647
crossoverOps	Newton barrier: Bit vector for adjusting the behavior of the crossover procedure	0
denseColLimit	Newton barrier: Controls trigger point for special treatment of dense columns in Cholesky factorization	auto

5.38.3.8 MIP Options

Option	Description	Default
backTrack	Branch and Bound: Specifies how to select the next node to work on when a full backtrack is performed	3
backtrackTie	Branch and Bound: Specifies how to break ties when selecting the next node to work on when a full backtrack is performed	-1

Option	Description	Default
branchChoice	Once a MIP entity has been selected for branching, this control determines which of the branches is solved first	0
branchDisj	Branch and Bound: Determines whether the optimizer should attempt to branch on general split disjunctions during the branch and bound search	auto
branchStructural	Branch and Bound: Determines whether the optimizer should search for special structure in the problem to branch on during the branch and bound search	auto
breadthFirst	Number of nodes to include in the best-first search before switching to the local first search (nodeSelection = 4)	11
deterministic	Selects whether to use a deterministic or opportunistic mode when solving a problem using multiple threads	1
feasibilityPump	Branch and Bound: Decides if the Feasibility Pump heuristic should be run at the top node	auto
fixoptfile	name of option file which is read just before solving the fixed problem	
genConsAbsTransformation	Specifies the reformulation method for absolute value general constraints at the beginning of the search	auto
genConsDualReductions	Parameter specifies whether dual reductions should be applied to reduce the number of columns and rows added when transforming general constraints to MIP structs	1
historyCosts	Branch and Bound: How to update the pseudo cost for a MIP entity when a strong branch or a regular branch is applied	auto
loadMipSol	Loads a MIP solution (the initial point)	0
localChoice	Controls when to perform a local backtrack between the two child nodes during a dive in the branch and bound tree	auto
maxLocalBacktrack	Branch-and-Bound: How far back up the current dive path the optimizer is allowed to look for a local backtrack candidate node	auto
maxMipSol	Branch and Bound: Limit on the number of integer solutions to be found by the Optimizer	0
maxMipTasks	Branch-and-Bound: The maximum number of tasks to run in parallel during a MIP solve	auto
maxNode	Branch and Bound: Maximum number of nodes that will be explored	maxint
maxStallTime	Maximum time in seconds that the Optimizer will continue to search for improving solution after finding a new incumbent	0
mipAbsCutoff	Branch and Bound: If the user knows that they are interested only in values of the objective function which are better than some value, this can be assigned to mipAbsCutoff	auto
mipAbsStop	Branch and Bound: Absolute tolerance determining whether the tree search will continue or not	0
mipAddCutoff	Branch and Bound: Amount to add to the objective function of the best integer solution found to give the new CURRMIPCUTOFF	0
mipCleanup	Clean up the MIP solution (round-fix-solve) to get duals	1
mipComponents	Determines whether disconnected components in a MIP should be solved as separate MIPs	auto
mipConcurrentNodes	Sets the node limit for when a winning solve is selected when concurrent MIP solves are enabled	auto

Option	Description	Default
mipConcurrentSolves	Selects the number of concurrent solves to start for a MIP	0
mipDualReductions	Branch and Bound: Limits operations that can reduce the MIP solution space	1
mipFracReduce	Branch and Bound: Specifies how often the optimizer should run a heuristic to reduce the number of fractional integer variables in the node LP solutions	auto
mipKappaFreq	Branch and Bound: Specifies how frequently the basis condition number (also known as kappa) should be calculated during the branch-and-bound search	0
mipLog	MIP log print control	-100
mipRampUp	Controls the strategy used by the parallel MIP solver during the ramp-up phase of a branch-and-bound tree search	auto
mipRefineIterLimit	Defines an effort limit expressed as simplex iterations for the MIP solution refiner	auto
mipRelCutoff	Branch and Bound: Percentage of the LP solution value to be added to the value of the objective function when an integer solution is found, to give the new value of CURRMIPCUTOFF	0
mipRelStop	Branch and Bound: Determines when the branch and bound tree search will terminate	0.0001
mipRestart	Branch and Bound: Controls strategy for in-tree restarts	auto
mipRestartFactor	Branch and Bound: Fine tune initial conditions to trigger an in-tree restart	1
mipRestartGapThreshold	Branch and Bound: Initial gap threshold to delay in-tree restart	0.02
mipstopexpr	Stopping expression for branch and bound	
mipTol	Branch and Bound: Tolerance within which a decision variable's value is considered to be integral	5e-06
mipToltarget	Target mipTol value used by the automatic MIP solution refiner as defined by refineOps	0
mipTrace	Name of MIP trace file	none
mipTraceNode	Node interval between MIP trace file entries	100
mipTraceTime	Time interval, in seconds, between MIP trace file entries	5
miqcpAlg	Determines which algorithm is to be used to solve mixed integer quadratic constrained and mixed integer second order cone problems	auto
nodeProbingEffort	Adjusts the overall level of node probing	1
nodeSelection	Branch and Bound: Determines which nodes will be considered for solution once the current node has been solved	auto
objGoodEnough	Stop once an objective this good is found	none
pseudoCost	Branch and Bound: Default pseudo cost used in estimation of the degradation associated with an unexplored node in the tree search	0.01
qcRootAlg	Determines which algorithm is to be used to solve the root of a mixed integer quadratic constrained or mixed integer second order cone problem, when outer approximation is used	auto

Option	Description	Default
sbBest	Number of infeasible MIP entities to initialize pseudo costs for on each node	auto
sbEffort	Adjusts the overall amount of effort when using strong branching to select an infeasible MIP entity to branch on	1
sbEstimate	Branch and Bound: How to calculate pseudo costs from the local node when selecting an infeasible MIP entity to branch on	auto
sbIterLimit	Number of dual iterations to perform the strong branching for each entity	auto
sbSelect	Size of the candidate list of MIP entities for strong branching	-2
sosRefTol	Minimum relative gap between the ordering values of elements in a special ordered set	1e-06
symmetry	Adjusts the overall amount of effort for symmetry detection	1
symSelect	Adjusts the overall amount of effort for symmetry detection	-1
varSelection	Branch and Bound: Determines the formula used to calculate the estimate of each integer variable, and thus which integer variable is selected to be branched on at a given node	auto

5.38.3.9 MIP Cuts Options

Option	Description	Default
autoCutting	Automatically decide whether to generate cutting planes at local nodes in the tree	auto
conflictCuts	Branch and Bound: Specifies how cautious or aggressive the optimizer should be when searching for and applying conflict cuts	auto
coverCuts	Branch and Bound: Number of rounds of lifted cover inequalities at the top node	auto
cutDepth	Branch and Bound: Sets the maximum depth in the tree search at which cuts will be generated	auto
cutFactor	Limit on the number of cuts and cut coefficients the optimizer is allowed to add to the matrix during tree search	auto
cutFreq	Branch and Bound: Frequency at which cuts are generated in the tree search	auto
cutSelect	Bit vector providing detailed control of the cuts created for the root node of a MIP solve	-1
cutSelect_clique	Clique cuts	1
cutSelect_cover	Lifted cover cuts	1
cutSelect_disableCutRows	Disable cutting from cut rows	1
cutSelect_farkas	Farkas cuts	1
cutSelect_flowpath	Flow path cuts	1
cutSelect_gomory	Strong Chvatal-Gomory cuts	1
cutSelect_gubCover	Lifted GUB cover cuts	1
cutSelect_implication	Implication cuts	1

Option	Description	Default
cutSelect_indicator	Indicator constraint cuts	1
cutSelect_liftAndProject	Turn on automatic Lift-and-Project cutting strategy	1
cutSelect_mir	Mixed Integer Rounding (MIR) cuts	1
cutSelect_mirRowAggregation	Turn on row aggregation for MIR cuts	1
cutSelect_zeroHalf	Zero-half cuts	1
cutStrategy	Branch and Bound: Cut strategy	auto
gomCuts	Branch and Bound: Number of rounds of Gomory or lift-and-project cuts at the top node	auto
lnpBest	Number of infeasible MIP entities to create lift-and-project cuts for during each round of Gomory cuts at the top node (see gomCuts)	50
lnpIterLimit	Number of iterations to perform in improving each lift-and-project cut	auto
maxCutTime	Maximum amount of time allowed for generation of cutting planes and reoptimization	0
netCuts	Determines the addition of multi-commodity network cuts to a problem	0
qcCuts	Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for the root node, when solving a mixed integer quadratic constrained or mixed integer second order conic problem with outer approximation	auto
treeCoverCuts	Branch and Bound: Number of rounds of lifted cover inequalities generated at nodes other than the top node in the tree	auto
treeCutSelect	Bit vector providing detailed control of the cuts created during the tree search of a MIP solve	-1
treeCutSelect_clique	Clique cuts	1
treeCutSelect_cover	Lifted cover cuts	1
treeCutSelect_disableCutRows	Disable cutting from cut rows	1
treeCutSelect_farkas	Farkas cuts	1
treeCutSelect_flowpath	Flow path cuts	1
treeCutSelect_gomory	Strong Chvatal-Gomory cuts	1
treeCutSelect_gubCover	Lifted GUB cover cuts	1
treeCutSelect_implication	Implication cuts	1
treeCutSelect_indicator	Indicator constraint cuts	1
treeCutSelect_liftAndProject	Turn on automatic Lift and Project cutting strategy	1
treeCutSelect_mir	Mixed Integer Rounding (MIR) cuts	1
treeCutSelect_mirRowAggregation	Turn on row aggregation for MIR cuts	1
treeCutSelect_zeroHalf	Zero-half cuts	1

Option	Description	Default
treeGomCuts	Branch and Bound: Number of rounds of Gomory cuts generated at nodes other than the first node in the tree	auto
treeQCCuts	Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for nodes other than the root node, when solving a mixed integer quadratic constrained or mixed integer second order conic problem with outer approximation	auto

5.38.3.10 MIP Heuristics Options

Option	Description	Default
feasibilityJump	MIP: Decides if the Feasibility Jump heuristic should be run	1
heurBeforeLp	Branch and Bound: Determines whether primal heuristics should be run before the initial LP relaxation has been solved	auto
heurDiveIterLimit	Branch and Bound: Simplex iteration limit for reoptimizing during the diving heuristic	auto
heurDiveRandomize	Level of randomization to apply in the diving heuristic	0
heurDiveSoftRounding	Branch and Bound: Enables a more cautious strategy for the diving heuristic, where it tries to push binaries and integer variables to their bounds using the objective, instead of directly fixing them	auto
heurDiveSpeedUp	Branch and Bound: Changes the emphasis of the diving heuristic from solution quality to diving speed	-1
heurDiveStrategy	Branch and Bound: Chooses the strategy for the diving heuristic	auto
heurEmphasis	Branch and Bound: Specifies an emphasis for the search w.r.t. primal heuristics and other procedures that affect the speed of convergence of the primal-dual gap	-1
heurForceSpecialObj	Branch and Bound: Whether local search heuristics without objective or with an auxiliary objective should always be used, despite the automatic selection of the Optimizer	0
heurFreq	Branch and Bound: Frequency at which heuristics are used in the tree search	-1
heurSearchEffort	Adjusts the overall level of the local search heuristics	1
heurSearchFreq	Branch and Bound: How often the local search heuristic should be run in the tree	auto
heurSearchRootCutFreq	How frequently to run the local search heuristic during root cutting	auto
heurSearchRootSelect	Bit vector control for selecting which local search heuristics to apply on the root node of a MIP solve	117
heurSearchTreeSelect	Bit vector control for selecting which local search heuristics to apply during the tree search of a MIP solve	17

5.38.3.11 MIP Solution Pool Options

Option	Description	Default
solnpool	Solution pool file name	none

Option	Description	Default
solnpoolCapacity	Limit on number of solutions to store	999999999
solnpoolCullDiversity	Cull N solutions based on solution diversity	-1
solnpoolCullObj	Cull N solutions based on objective values	-1
solnpoolCullRounds	Terminate solution generation after N culling rounds	999999999
solnpoolDupPolicy	Policy to use when handling storage of duplicate solutions	0
solnpoolmerge	Solution pool file name for merged solutions	none
solnpoolnumsym	Maximum number of variable symbols when writing merged solutions	10
solnpoolPop	Controls method used to populate the solution pool	1
solnpoolPrefix	File name prefix for GDX solution files	soln
solnpoolVerbosity	Controls verbosity of solution pool routines	0

5.38.3.12 General NLP / MINLP Options

Option	Description	Default
knitroOptFile	Option file for NLP solver KNITRO	
xslp_algorithm	Bit map describing the SLP algorithm(s) to be used	166
xslp_algorithm_cascadeBounds	Step bounds are updated to accomodate cascaded values (otherwise cascaded values are pushed to respect step bounds)	0
xslp_algorithm_clampExtendedActiveSB	Apply clamping when converged on extended criteria only with some variables having active step bounds	0
xslp_algorithm_clampExtendedAll	Apply clamping when converged on extended criteria only	0
xslp_algorithm_dynamicDamping	Use dynamic damping	0
xslp_algorithm_escalatePenalties	Escalate penalties	0
xslp_algorithm_estimateStepBounds	Estimate step bounds from early SLP iterations	1
xslp_algorithm_holdValues	Do not update values which are converged within strict tolerance	0
xslp_algorithm_maxCostOption	Continue optimizing after penalty cost reaches maximum	0
xslp_algorithm_noLPPolishing	Skip the solution polishing step if the LP postsolve returns a slightly infeasible, but claimed optimal solution	0
xslp_algorithm_noStepBounds	Do not apply step bounds	0
xslp_algorithm_quickConvergenceCheck	Quick convergence check	1
xslp_algorithm_resetDeltaZ	Reset xslp_delta_z to zero when converged and continue SLP	0
xslp_algorithm_residualErrors	Accept a solution which has converged even if there are still significant active penalty error vectors	0

Option	Description	Default
xslp_algorithm_retainPreviousValue	Retain previous value when cascading if determining row is zero	1
xslp_algorithm_stepBoundsAsRequired	Apply step bounds to SLP delta vectors only when required	1
xslp_algorithm_switchToPrimal	Use the primal simplex algorithm when all error vectors become inactive	0
xslp_calcThreads	Number of threads used for formula and derivatives evaluations	auto
xslp_filter	Bit map for controlling solution updates	3
xslp_filterKeepBest	Retrain solution best according to the merit function	1
xslp_filterZeroLineSearch	Force minimum step sizes in line search	0
xslp_filterZeroLineSearchTR	Accept the trust region step is the line search returns a zero step size	0
xslp_findIV	Option for running a heuristic to find a feasible initial point	auto
xslp_infinity	Value returned by a divide-by-zero in a formula	1e+10
xslp_primalIntegralRef	Reference solution value to take into account when calculating the primal integral	1e+20
xslp_scale	When to re-scale the SLP problem	1
xslp_scaleCount	Iteration limit used in determining when to re-scale the SLP matrix	0
xslp_solver	Selects the library to use for local solves	auto
xslp_threads	Default number of threads to be used	auto
xslp_zero	Absolute tolerance	1e-15

5.38.3.13 NLP Presolve Options

Option	Description	Default
xslp_linQuadBR	Use linear and quadratic constraints and objective function to further reduce bounds on all variables	auto
xslp_postsolve	Determines whether postsolving should be performed automatically	-1
xslp_presolve	Determines whether presolving should be performed prior to starting the main algorithm	1
xslp_presolveLevel	Determines the level of changes presolve may carry out on the problem	4
xslp_presolveOps	Bitmap indicating the SLP presolve actions to be taken	2104
xslp_presolveOps_domain	Bound tightening based on function domains	1
xslp_presolveOps_eliminations	Allow eliminations on determined variables	1
xslp_presolveOps_fixAll	Explicitly fix all columns identified as fixed	0
xslp_presolveOps_fixZero	Explicitly fix columns identified as fixed to zero	0
xslp_presolveOps_general	Generic SLP presolve	0

Option	Description	Default
xslp_presolveOps_intBounds	MISLP bound tightening	1
xslp_presolveOps_noCoefficients	Do not presolve coefficients	0
xslp_presolveOps_noDeltas	Do not remove delta variables	0
xslp_presolveOps_noDualSide	Avoid reductions that can not be dual postsolved	0
xslp_presolveOps_noLinear	Avoid performing linear reductions at the nlp level	0
xslp_presolveOps_noSimplifier	Avoid simplifying nonlinear expressions	0
xslp_presolveOps_setBounds	SLP bound tightening	1
xslp_presolveZero	Minimum absolute value for a variable which is identified as nonzero during SLP presolve	1e-09
xslp_probing	Determines whether probing on a subset of variables should be performed prior to starting the main algorithm	auto

5.38.3.14 NLP Augmentation and Linearization Options

Option	Description	Default
xslp_augmentation	Bit map describing the SLP augmentation method(s) to be used	12
xslp_augmentation_allErrorVectors	Penalty error vectors on all non-linear inequality constraints	1
xslp_augmentation_allRowErrorVectors	Penalty error vectors on all constraints	0
xslp_augmentation_aMeanWeight	Use arithmetic means to estimate penalty weights	0
xslp_augmentation_equalityErrorVectors	Penalty error vectors on all non-linear equality constraints	1
xslp_augmentation_evenHanded	Even handed augmentation	0
xslp_augmentation_minimum	Minimum augmentation	0
xslp_augmentation_noUpdateIfOnlyIV	Intial values do not imply an SLP variable	0
xslp_augmentation_penaltyDeltaVectors	Penalty vectors to exceed step bounds	0
xslp_augmentation_sbFromAbsValues	Estimate step bounds from absolute values of row coefficients	0
xslp_augmentation_sbFromValues	Estimate step bounds from values of row coefficients	0
xslp_augmentation_stepBoundRows	Row-based step bounds	0
xslp_delta_x	Minimum absolute value of delta coefficients to be retained	1e-06
xslp_feasTolTarget	When set, this defines a target feasibility tolerance to which the linearizations are solved to	not set
xslp_optimalityTolTarget	When set, this defines a target optimality tolerance to which the linearizations are solved to	not set
xslp_unfinishedLimit	Number of consecutive SLP iterations that may have an unfinished status before the solve is terminated	3

Option	Description	Default
xslp_zeroCriterion	Bitmap determining the behavior of the placeholder deletion procedure	0
xslp_zeroCriterionCount	Number of consecutive times a placeholder entry is zero before being considered for deletion	0
xslp_zeroCriterionStart	SLP iteration at which criteria for deletion of placeholder entries are first activated	0
xslp_zeroCriterion_deltaNBDRRow	Remove placeholders in a basic delta variable if the determining row for the corresponding SLP variable is nonbasic	0
xslp_zeroCriterion_deltaNBUpdateRow	Remove placeholders in a basic delta variable if its update row is nonbasic and the corresponding SLP variable is nonbasic	0
xslp_zeroCriterion_nbDelta	Remove placeholders in nonbasic delta variables	0
xslp_zeroCriterion_nbSLPVar	Remove placeholders in nonbasic SLP variables	0
xslp_zeroCriterion_print	Print information about zero placeholders	0
xslp_zeroCriterion_slpVarNBUpdateRow	Remove placeholders in a basic SLP variable if its update row is nonbasic	0

5.38.3.15 NLP Barrier Options

Option	Description	Default
xslp_barCrossOverStart	Default crossover activation behaviour for barrier start	0
xslp_barLimit	Number of initial SLP iterations using the barrier method	0
xslp_barStallingLimit	Number of iterations to allow numerical failures in barrier before switching to dual	3
xslp_barStallingObjLimit	Number of iterations over which to measure the objective change for barrier iterations with no crossover	3
xslp_barStallingTol	Required change in the objective when progress is measured in barrier iterations without crossover	0.05
xslp_barStartOps	Controls behaviour when the barrier is used to solve the linearizations	-1
xslp_barStartOps_allowInteriorSol	If a non-vertex converged solution found by barrier without crossover can be returned as a final solution	1
xslp_barStartOps_stallingNumerical	Fall back to dual simplex if too many numerical problems are reported by the barrier	1
xslp_barStartOps_stallingObjective	Check objective progress when no crossover is applied	1

5.38.3.16 NLP Penalty Options

Option	Description	Default
xslp_deltaCost	Initial penalty cost multiplier for penalty delta vectors	200
xslp_deltaCostFactor	Factor for increasing cost multiplier on total penalty delta vectors	1.3

Option	Description	Default
xslp_deltaMaxCost	Maximum penalty cost multiplier for penalty delta vectors	1e+20
xslp_enforceCostShrink	Factor by which to decrease the current penalty multiplier when enforcing rows	1e-05
xslp_enforceMaxCost	Maximum penalty cost in the objective before enforcing most violating rows	1e+11
xslp_errorCost	Initial penalty cost multiplier for penalty error vectors	200
xslp_errorCostFactor	Factor for increasing cost multiplier on total penalty error vectors	1.3
xslp_errorMaxCost	Maximum penalty cost multiplier for penalty error vectors	1e+20
xslp_errorTol_a	Absolute tolerance for error vectors	1e-05
xslp_errorTol_p	Absolute tolerance for printing error vectors	0.0001
xslp_escalation	Factor for increasing cost multiplier on individual penalty error vectors	1.25
xslp_eTol_a	Absolute tolerance on penalty vectors	0.0001
xslp_eTol_r	Relative tolerance on penalty vectors	0.0001
xslp_evTol_a	Absolute tolerance on total penalty costs	-1
xslp_evTol_r	Relative tolerance on total penalty costs	-1
xslp_granularity	Base for calculating penalty costs	4
xslp_maxWeight	Maximum penalty weight for delta or error vectors	100
xslp_minWeight	Minimum penalty weight for delta or error vectors	0.01
xslp_objToPenaltyCost	Factor to estimate initial penalty costs from objective function	0
xslp_penaltyInfoStart	Iteration from which to record row penalty information	3

5.38.3.17 NLP Step Bounds Options

Option	Description	Default
xslp_clampShrink	Shrink ratio used to impose strict convergence on variables converged in extended criteria only	0.3
xslp_clampValidationTol_a	Absolute validation tolerance for applying xslp_clampShrink	not set
xslp_clampValidationTol_r	Relative validation tolerance for applying xslp_clampShrink	not set
xslp_defaultStepBound	Minimum initial value for the step bound of an SLP variable if none is explicitly given	16
xslp_djTol	Tolerance on DJ value for determining if a variable is at its step bound	1e-06
xslp_expand	Multiplier to increase a step bound	2
xslp_minSBFactor	Factor by which step bounds can be decreased beneath xslp_aTol_a	1
xslp_sameCount	Number of steps reaching the step bound in the same direction before step bounds are increased	3
xslp_sbStart	SLP iteration after which step bounds are first applied	8

Option	Description	Default
xslp_shrink	Multiplier to reduce a step bound	0.5
xslp_shrinkBias	Defines an overwrite / adjustment of step bounds for improving iterations	not set

5.38.3.18 NLP Variable Update Options

Option	Description	Default
xslp_damp	Damping factor for updating values of variables	1
xslp_dampExpand	Multiplier to increase damping factor during dynamic damping	1
xslp_dampMax	Maximum value for the damping factor of a variable during dynamic damping	1
xslp_dampMin	Minimum value for the damping factor of a variable during dynamic damping	1
xslp_dampShrink	Multiplier to decrease damping factor during dynamic damping	1
xslp_dampStart	SLP iteration at which damping is activated	0
xslp_lsIterLimit	Number of iterations in the line search	0
xslp_lsPatternLimit	Number of iterations in the pattern search preceding the line search	0
xslp_lsStart	Iteration in which to active the line search	8
xslp_lsZeroLimit	Maximum number of zero length line search steps before line search is deactivated	5
xslp_meritLambda	Factor by which the net objective is taken into account in the merit function	0
xslp_sameDamp	Number of steps in same direction before damping factor is increased	3

5.38.3.19 NLP Termination Options

Option	Description	Default
xslp_aTol_a	Absolute delta convergence tolerance	auto
xslp_aTol_r	Relative delta convergence tolerance	auto
xslp_convergenceOps	Bit map describing which convergence tests should be carried out	7167
xslp_convergenceOps_aTol	Execute the delta tolerance checks	1
xslp_convergenceOps_cTol	Execute the closure tolerance checks	1
xslp_convergenceOps_extendedScaling	Take scaling of individual variables / rows into account	0
xslp_convergenceOps_iTol	Execute the impact tolerance checks	1
xslp_convergenceOps_mTol	Execute the matrix tolerance checks	1
xslp_convergenceOps_oTol	Execute the objective range + active step bound check	1
xslp_convergenceOps_sTol	Execute the slack impact tolerance checks	1

Option	Description	Default
xslp_convergenceOps_validation	Execute the validation target convergence checks	1
xslp_convergenceOps_validationK	Execute the first order optimality target convergence checks	1
xslp_convergenceOps_vTol	Execute the objective range checks	1
xslp_convergenceOps_wTol	Execute the convergence continuation check	1
xslp_convergenceOps_xTol	Execute the objective range + constraint activity check	1
xslp_cTol	Closure convergence tolerance	auto
xslp_ecfCheck	Check feasibility at the point of linearization for extended convergence criteria	1
xslp_ecfTol_a	Absolute tolerance on testing feasibility at the point of linearization	auto
xslp_ecfTol_r	Relative tolerance on testing feasibility at the point of linearization	auto
xslp_infeasLimit	Maximum number of consecutive infeasible SLP iterations which can occur before Xpress-SLP terminates	3
xslp_iterLimit	Maximum number of SLP iterations	auto
xslp_iTol_a	Absolute impact convergence tolerance	auto
xslp_iTol_r	Relative impact convergence tolerance	auto
xslp_mTol_a	Absolute effective matrix element convergence tolerance	auto
xslp_mTol_r	Relative effective matrix element convergence tolerance	auto
xslp_mvTol	Marginal value tolerance for determining if a constraint is slack	auto
xslp_oCount	Number of SLP iterations over which to measure objective function variation for static objective (2) convergence criterion	5
xslp_oTol_a	Absolute static objective (2) convergence tolerance	auto
xslp_oTol_r	Relative static objective (2) convergence tolerance	auto
xslp_sTol_a	Absolute slack convergence tolerance	auto
xslp_sTol_r	Relative slack convergence tolerance	auto
xslp_stopOutOfRange	Stop optimization and return error code if internal function argument is out of range	0
xslp_validationTarget_k	Optimality target tolerance	1e-06
xslp_validationTarget_r	Feasiblity target tolerance	1e-06
xslp_vCount	Number of SLP iterations over which to measure static objective (3) convergence	0
xslp_vLimit	Number of SLP iterations after which static objective (3) convergence testing starts	0
xslp_vTol_a	Absolute static objective (3) convergence tolerance	auto

Option	Description	Default
xslp_vTol_r	Relative static objective (3) convergence tolerance	auto
xslp_wCount	Number of SLP iterations over which to measure the objective for the extended convergence continuation criterion	0
xslp_wTol_a	Absolute extended convergence continuation tolerance	auto
xslp_wTol_r	Relative extended convergence continuation tolerance	auto
xslp_xCount	Number of SLP iterations over which to measure static objective (1) convergence	5
xslp_xLimit	Number of SLP iterations up to which static objective (1) convergence testing starts	100
xslp_xTol_a	Absolute static objective function (1) tolerance	auto
xslp_xTol_r	Relative static objective function (1) tolerance	auto

5.38.3.20 NLP Multistart Options

Option	Description	Default
xslp_msMaxBoundRange	Defines the maximum range inside which initial points are generated by multistart presets	1000
xslp_multistartPreset	Enable multistart	0
xslp_multistart_maxSolves	Maximum number of jobs to create during the multistart search	unlimited
xslp_multistart_maxTime	Maximum total time to be spent in the mutlistart search	unlimited
xslp_multistart_poolsize	Maximum number of problem objects allowed to pool up before synchronization in the deterministic multistart	2
xslp_multistart_seed	Random seed used for the automatic generation of initial point when loading multistart presets	0
xslp_multistart_threads	Maximum number of threads to be used in multistart	auto

5.38.3.21 NLP Derivative Options

Option	Description	Default
xslp_cdTol_a	Absolute tolerance for deducing constant derivatives	1e-08
xslp_cdTol_r	Relative tolerance for deducing constant derivatives	1e-08
xslp_deltaZLimit	Number of SLP iterations during which to apply xslp_delta_z	0
xslp_delta_a	Absolute perturbation of values for calculating numerical derivatives	0.001
xslp_delta_r	Relative perturbation of values for calculating numerical derivatives	0.001
xslp_delta_z	Tolerance used when calculating derivatives	1e-05
xslp_delta_zero	Absolute zero acceptance tolerance used when calculating derivatives	-1
xslp_derivatives	Bitmap describing the method of calculating derivatives	1

Option	Description	Default
xslp_hessian	Second order differentiation mode when using analytical derivatives	-1
xslp_jacobian	First order differentiation mode when using analytical derivatives	-1

5.38.3.22 NLP Log Options

Option	Description	Default
xslp_analyze	Bit map activating additional options supporting model / solution path analyzis	0
xslp_analyze_extendedFinalSummary	Include an extended iteration summary	0
xslp_analyze_infeasibleIteration	Run infeasibility analysis on infeasible iterations	0
xslp_analyze_saveFile	Create an Xpress SLP save file at every xslp_autosave iterations	0
xslp_analyze_saveIterBasis	Write the initial basis of the linearizations to disk at every xslp_autosave iterations	0
xslp_analyze_saveLinearizations	Write the linearizations to disk at every xslp_autosave iterations	0
xslp_autosave	Frequency with which to save the model	0
xslp_log	Level of printing during SLP iterations	0
xslp_slpLog	Frequency with which SLP status is printed	1

5.38.3.23 MINLP Options

Option	Description	Default
xslp_cutStrategy	Determines whihc cuts to apply in the MISLP search when the default SLP-in-MIP strategy is used	0
xslp_heurStrategy	Branch and Bound: MINLP heuristic strategy	auto
xslp_mipAlgorithm	Bitmap describing the MISLP algorithms to be used	17
xslp_mipAlgorithm_finalFixSLP	Fix step bounds according to xslp_mipFixStepBounds after MIP solution is found	0
xslp_mipAlgorithm_finalRelaxSLP	Relax step bounds according to xslp_mipRelaxStepBounds after MIP solution is found	0
xslp_mipAlgorithm_initialFixSLP	Fix step bounds according to xslp_mipFixStepBounds after initial node	0
xslp_mipAlgorithm_initialRelaxSLP	Relax step bounds according to xslp_mipRelaxStepBounds after initial node	0
xslp_mipAlgorithm_initialSLP	Solve initial SLP to convergence	1
xslp_mipAlgorithm_nodeFixSLP	Fix step bounds according to xslp_mipFixStepBounds at each node	0

Option	Description	Default
xslp_mipAlgorithm_nodeLimitSLP	Limit iterations at each node to xslp_mipIterLimit	0
xslp_mipAlgorithm_nodeRelaxSLP	Relax step bounds according to xslp_mipRelaxStepBounds at each node	1
xslp_mipAlgorithm_slpThenMIP	Use MIP on converged SLP solution and then SLP on the resulting MIP solution	0
xslp_mipAlgorithm_withinSLP	Use MIP at each SLP iteration instead of SLP at each node	0
xslp_mipCutOffCount	Number of SLP iterations to check when considering a node for cutting off	5
xslp_mipCutOffLimit	Number of SLP iterations to check when considering a node for cutting off	10
xslp_mipCutOff_a	Absolute objective function cutoff for MIP termination	1e-05
xslp_mipCutOff_r	Absolute objective function cutoff for MIP termination	1e-05
xslp_mipDefaultAlgorithm	Default algorithm to be used during the tree search in MISLP	auto
xslp_mipErrorTol_a	Absolute penalty error cost tolerance for MIP cut-off	0
xslp_mipErrorTol_r	Relative penalty error cost tolerance for MIP cut-off	0
xslp_mipFixStepBounds	Bitmap describing the step-bound fixing strategy during MISLP	0
xslp_mipFixStepBounds_coef	Fix step bounds on SLP variables appearing in coefficients	0
xslp_mipFixStepBounds_coefOnly	Fix step bounds on SLP variables appearing only in coefficients	0
xslp_mipFixStepBounds_structAll	Fix step bounds on all structural SLP variables	0
xslp_mipFixStepBounds_structNotCoef	Fix step bounds on structural SLP variables which are not in coefficients	0
xslp_mipIterLimit	Maximum number of SLP iterations at each node	0
xslp_mipLog	Frequency with which MIP status is printed	0
xslp_mipOCount	Number of SLP iterations at each node over which to measure objective function variation	5
xslp_mipOTol_a	Absolute objective function tolerance for MIP termination	1e-05
xslp_mipOTol_r	Relative objective function tolerance for MIP termination	1e-05
xslp_mipRelaxStepBounds	Bitmap describing the step-bound relaxation strategy during MISLP	15
xslp_mipRelaxStepBounds_coef	Relax step bounds on SLP variables appearing in coefficients	1
xslp_mipRelaxStepBounds_coefOnly	Relax step bounds on SLP variables appearing only in coefficients	1
xslp_mipRelaxStepBounds_structAll	Relax step bounds on all structural SLP variables	1
xslp_mipRelaxStepBounds_structNotCoef	Relax step bounds on structural SLP variables which are not in coefficients	1

5.38.4 Detailed Descriptions of XPRESS Options

advBasis *(boolean)*: Use advanced basis provided by GAMS ↩

Default: `auto`

algAfterCrossover *(integer)*: Algorithm to be used for the final clean up step after the crossover ↩

Default: `auto`

value	meaning
1	Automatically determined.
2	Dual simplex.
3	Primal simplex.
4	Concurrent.

algAfterNetwork *(integer)*: Algorithm to be used for the clean up step after the network simplex solver ↩

Default: `auto`

value	meaning
-1	Automatically determined.
2	Dual simplex.
3	Primal simplex.

algorithm *(string)*: Choose between simplex and barrier algorithm ↩

This option is used to select the barrier method to solve LPs. By default the barrier method will do a crossover to find a basic solution.

Default: `simplex`

value	meaning
barrier	Use the barrier algorithm
simplex	Use the simplex algorithm

autoCutting *(integer)*: Automatically decide whether to generate cutting planes at local nodes in the tree ↩

If the cutFreq control is set, no automatic selection will be made and local cutting will be enabled.

Default: `auto`

value	meaning
-1	Automatic.
0	Disabled.
1	Enabled.

autoPerturb *(boolean)*: Simplex: Indicates whether automatic perturbation is performed ↩

If this is set to 1, the problem will be perturbed whenever the simplex method encounters an excessive number of degenerate pivot steps, thus preventing the Optimizer being hindered by degeneracies.

Default: `auto`

value	meaning
`0`	No perturbation performed.
`1`	Automatic perturbation is performed.

autoScaling *(integer)*: Whether the Optimizer should automatically select between different scaling algorithms ↩

If the scaling control is set, no automatic scaling will be applied.

Default: `auto`

value	meaning
`-1`	Automatic.
`0`	Disabled.
`1`	Cautious strategy. Non-standard scaling will only be selected if it appears to be clearly superior.
`2`	Moderate strategy.
`3`	Aggressive strategy. Standard scaling will only be selected if it appears to be clearly superior.

backTrack *(integer)*: Branch and Bound: Specifies how to select the next node to work on when a full backtrack is performed ↩

Note When two nodes are rated the same according to the backTrack selection, a secondary rating is performed using the method set by backtrackTie.

Default: `3`

value	meaning
`-1`	Automatically determined.
`1`	Unused.
`2`	Select the node with the best estimated solution.
`3`	Select the node with the best bound on the solution.
`4`	Select the deepest node in the search tree (equivalent to depth-first search).
`5`	Select the highest node in the search tree (equivalent to breadth-first search).
`6`	Select the earliest node created.
`7`	Select the latest node created.
`8`	Select a node randomly.
`9`	Select the node whose LP relaxation contains the fewest number of infeasible MIP entities.
`10`	Combination of 2 and 9.
`11`	Combination of 2 and 4.
`12`	Combination of 3 and 4.

backtrackTie *(integer)*: Branch and Bound: Specifies how to break ties when selecting the next node to work on when a full backtrack is performed ↩

The options are the same as for the backTrack control.

Default: `-1`

value	meaning
`-1`	Default selection.
`1`	Unused.
`2`	Select the node with the best estimated solution.
`3`	Select the node with the best bound on the solution.
`4`	Select the deepest node in the search tree (equivalent to depth-first search).
`5`	Select the highest node in the search tree (equivalent to breadth-first search).
`6`	Select the earliest node created.
`7`	Select the latest node created.
`8`	Select a node randomly.
`9`	Select the node whose LP relaxation contains the fewest number of infeasible MIP entities.
`10`	Combination of 2 and 9.
`11`	Combination of 2 and 4.
`12`	Combination of 3 and 4.

barAlg *(integer)*: Determines which barrier algorithm is to be used to solve the problem ↩

The automatic setting uses 1 for LP and QP problems and 3 for QCQP problems. Usually the detection of primal or dual infeasibility is more robust with settings 2 or 3, therefore, it is advantageous to use one of these values if the model is presumably infeasible.

Default: `auto`

value	meaning
`-1`	Determined automatically.
`0`	Unused.
`1`	Use the infeasible-start barrier algorithm.
`2`	Use the homogeneous self-dual barrier algorithm.
`3`	Start with 2 and optionally switch to 1 during the execution.

barCores *(integer)*: If set to a positive integer it determines the number of physical CPU cores assumed to be present in the system by the barrier algorithm ↩

If the value is set to the default value (-1), Xpress will automatically detect the number of cores.

The control is provided for cross-hardware reproducibility purposes. The count does not include logical cores created by Hyper-Threading.

Range: {`-1`, ..., ∞}

Default: `auto`

barCrash *(integer)*: Newton barrier: Determines the type of crash used for the crossover ↩

During the crash procedure, an initial basis is determined which attempts to speed up the crossover. A good choice at this stage will significantly reduce the number of iterations required to crossover to an optimal solution. The possible values increase proportionally to their time-consumption.

Default: `4`

value	meaning
`0`	Turns off all crash procedures.
`1-6`	Available strategies with 1 being conservative and 6 being aggressive.

barDualStop *(real)*: Newton barrier: Convergence parameter, representing the tolerance for dual infeasibilities ↩

If the difference between the constraints and their bounds in the dual problem falls below this tolerance in absolute value, optimization will stop and the current solution will be returned.

Range: [`0`, ∞]

Default: `auto`

barFailIterLimit *(integer)*: Newton barrier: Maximum number of consecutive iterations that fail to improve the solution in the barrier algorithm ↩

Default: `auto`

value	meaning
`0`	Determined automatically
`>0`	Maximum number of consecutive barrier iterations allowed without progress.

barFreeScale *(real)*: Defines how the barrier algorithm scales free variables ↩

When using smaller values the barrier algorithm scales free variables more aggressively which can improve performance but may impact numerical stability.

Range: [`0`, ∞]

Default: `1e-06`

barGapStop *(real)*: Newton barrier: Convergence parameter, representing the tolerance for the relative duality gap ↩

When the difference between the primal and dual objective function values falls below this tolerance, the Optimizer determines that the optimal solution has been found.

Range: [`0`, ∞]

Default: `auto`

barGapTarget *(real)*: Newton barrier: Target tolerance for the relative duality gap ↩

The barrier algorithm will keep iterating until either barGapTarget is satisfied or until no further improvements are possible. In the latter case, if barGapStop is satisfied, it will declare the problem optimal.

When a solution returned by the barrier algorithm has not converged tightly enough for an application, for example if the dual solution is not accurate enough or crossover is taking too long, setter barGapTarget to a small value often resolves the problem, without the risk of the solve failing due to a complementarity level not being numerically achievable. Typical suggested values can be between 1-10 and 1-18.

Range: [0, ∞]

Default: auto

barIndefLimit *(integer)*: Newton Barrier: Limits the number of consecutive indefinite barrier iterations that will be performed ↩

The optimizer will try to minimize (resp. maximize) a QP problem even if the Q matrix is not positive (resp. negative) semi-definite. However, the optimizer may detect that the Q matrix is indefinite and this can result in the optimizer not converging. This control specifies how many indefinite iterations may occur before the optimizer stops and reports that the problem is indefinite. It is usual to specify a value greater than one, and only stop after a series of indefinite matrices, as the problem may be found to be indefinite incorrectly on a few iterations for numerical reasons.

Range: {1, ..., ∞}

Default: 15

barIterLimit *(integer)*: Newton barrier: Maximum number of iterations ↩

While the simplex method usually performs a number of iterations which is proportional to the number of constraints (rows) in a problem, the barrier method standardly finds the optimal solution to a given accuracy after a number of iterations which is independent of the problem size. The penalty is rather that the time for each iteration increases with the size of the problem. barIterLimit specifies the maximum number of iterations which will be carried out by the barrier.

Range: {0, ..., ∞}

Default: 500

barKernel *(real)*: Newton barrier: Defines how centrality is weighted in the barrier algorithm ↩

Increasing this parameter may increase the number of iterations, therefore the recommended range is [1,2] and [-2,-1].

Default: 0

value	meaning
>=+1.0	Increases the emphasis on centrality when larger value is set.
<=-1.0	Selects a value adaptively in every iteration from [+1, -barKernel].

barObjPerturb *(real)*: Defines how the barrier perturbs the objective ↩

The perturbation scale should be set carefully with consideration to the optimality tolerance. The parameter affects only the barrier solve.

Default: `1e-06`

value	meaning
`0`	Turn off objective perturbation.
`>0`	Let the optimizer decide if the objective is perturbed or not and use the parameter value as the scale of the perturbation.
`<0`	Always perturb the objective by the absolute value of the parameter.

barObjScale *(real)*: Defines how the barrier scales the objective ↩

The scaling perfromed by the barrier is applied on top of any other scaling in the problem and only affects the barrier solve.

Default: `auto`

value	meaning
`-1`	Let the optimizer decide.
`0`	Scale by geometric mean.
`>=0`	Scale such that the largest objective coefficient's largest element does not exceed this number. In quadratic problems, the quadratic diagonal is used as reference valuses instead of the linear objective.

barOrder *(integer)*: Newton barrier: Controls the Cholesky factorization in the Newton-Barrier ↩

Default: `auto`

value	meaning
`0`	Choose automatically.
`1`	Minimum degree method. This selects diagonal elements with the smallest number of nonzeros in their rows or columns.
`2`	Minimum local fill method. This considers the adjacency graph of nonzeros in the matrix and seeks to eliminate nodes that minimize the creation of new edges.
`3`	Nested dissection method. This considers the adjacency graph and recursively seeks to separate it into non-adjacent pieces.

barOrderThreads *(integer)*: If set to a positive integer it determines the number of concurrent threads for the sparse matrix ordering algorithm in the Newton-barrier method ↩

Larger values than barCores will be automatically reduced to the value of barCores.

Range: {`0`, ..., ∞}

Default: `auto`

barOutput *(boolean)*: Newton barrier: Level of solution output provided ↩

Output is provided either after each iteration of the algorithm, or else can be turned off completely by this parameter.

Default: `1`

value	meaning
0	No output.
1	At each iteration.

barPerturb *(real)*: Newton barrier: In numerically challenging cases it is often advantageous to apply perturbations on the KKT system to improve its numerical properties ↩

barPerturb controlls how much perturbation is allowed during the barrier iterations. By default no perturbation is allowed. Set this parameter with care as larger perturbations may lead to less efficient iterates and the best settings are problem-dependent.

Range: [0, ∞]

Default: 0

barPresolveOps *(integer)*: Newton barrier: Controls the Newton-Barrier specific presolve operations ↩

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0

value	meaning
bit 0 = 1	Equivalent to barPresolveOps_standard.
bit 1 = 2	Equivalent to barPresolveOps_extra.
bit 2 = 4	Equivalent to barPresolveOps_full.

barPresolveOps_extra *(boolean)*: Extra effort is spent in barrier specific presolve ↩

See also barPresolveOps.

Default: 0

barPresolveOps_full *(boolean)*: Do full matrix eliminations (reduce matrix size) ↩

See also barPresolveOps.

Default: 0

barPresolveOps_standard *(boolean)*: Use standard presolve ↩

See also barPresolveOps.

Default: 0

barPrimalStop *(real)*: Newton barrier: Convergence parameter, indicating the tolerance for primal infeasibilities ↩

If the difference between the constraints and their bounds in the primal problem falls below this tolerance in absolute value, the Optimizer will terminate and return the current solution.

Range: [0, ∞]

Default: auto

barRefIter *(integer)*: Newton barrier: After terminating the barrier algorithm, further refinement steps can be performed ↩

Such refinement steps are especially helpful if the solution is near to the optimum and can improve primal feasibility and decrease the complementarity gap. It is also often advantageous for the crossover algorithm. barRefIter specifies the maximum number of such refinement iterations.

Range: {0, ..., ∞}

Default: 0

barRegularize *(integer)*: Determines how the barrier algorithm applies regularization on the KKT system ↩

The parameter is a bit set but value -1 (the default value) is treated in a special way: if the parameter is set to -1 then the solver will automatically select the bits it deems most useful.

Default: -1

value	meaning
bit 0 = 1	Standard regularization is turned on/off.
bit 1 = 2	Reduced regularization is turned on/off. This option reduces the perturbation effect of the standard regularization.
bit 2 = 4	Forces to keep dependent rows in the KKT system.
bit 3 = 8	Forces to preserve degenerate rows in the KKT system.
bit 4 = 16	Restrict regularization to infeasible iterates.
bit 5 = 32	Disable iterative regularizations.
bit 6 = 64	Apply iterative regularization more often.

barRhsScale *(real)*: Defines how the barrier scales the right hand side ↩

The scaling perfromed by the barrier is applied on top of any other scaling in the problem and only affects the barrier solve.

Default: auto

value	meaning
-1	Let the optimizer decide.
0	Scale by geometric mean.
>=0	Scale such that the largest right hand side coefficient's largest element does not exceed this number.

barStart *(integer)*: Newton barrier: Controls the computation of the starting point for the barrier algorithm ↩

Default: auto

value	meaning
-1	Uses the available solution for warm-start.
0	Determine automatically.
1	Uses simple heuristics to compute the starting point based on the magnitudes of the matrix entries.

value	meaning
2	Uses the pseudoinverse of the constraint matrix to determine primal and dual initial solutions. Less sensitive to scaling and numerically more robust, but in several case less efficient than 1.
3	Uses the unit starting point for the homogeneous self-dual barrier algorithm.

barStepStop *(real)*: Newton barrier: Convergence parameter, representing the minimal step size ↩

On each iteration of the barrier algorithm, a step is taken along a computed search direction. If that step size is smaller than barStepStop, the Optimizer will terminate and return the current solution.

If the barrier method is making small improvements on barGapStop on later iterations, it may be better to set this value higher, to return a solution after a close approximation to the optimum has been found.

Range: [0, ∞]

Default: `1e-16`

barThreads *(integer)*: If set to a positive integer it determines the number of threads implemented to run the Newton-barrier algorithm ↩

If the value is set to the default value (-1), the threads control will determine the number of threads used.

There is a practical upper limit of 50 on the number of parallel threads the optimizer will create.

Range: {`-1`, ..., ∞}

Default: `auto`

basisOut *(string)*: Directs optimizer to output an MPS basis file ↩

In general this option is not used in a GAMS environment, as GAMS maintains basis information for you automatically.

Default: `none`

bigM *(real)*: Infeasibility penalty used if the "Big M" method is implemented ↩

Range: [0, ∞]

Default: `auto`

bigMMethod *(boolean)*: Simplex: Whether to use the "Big M" method, or the standard phase I (achieving feasibility) and phase II (achieving optimality) ↩

In the "Big M" method, the objective coefficients of the variables are considered during the feasibility phase, possibly leading to an initial feasible basis which is closer to optimal. The side-effects involve possible round-off errors due to the presence of the "Big M" factor in the problem.

Default: `1`

value	meaning
0	For phase I / phase II.
1	If "Big M" method to be used.

branchChoice *(integer)*: Once a MIP entity has been selected for branching, this control determines which of the branches is solved first ↩

Default: 0

value	meaning
0	Minimum estimate branch first.
1	Maximum estimate branch first.
2	If an incumbent solution exists, solve the branch satisfied by that solution first. Otherwise solve the minimum estimate branch first (option 0).
3	Solve first the branch that forces the value of the branching variable to move farther away from the value it had at the root node. If the branching entity is not a simple variable, solve the minimum estimate branch first (option 0).

branchDisj *(integer)*: Branch and Bound: Determines whether the optimizer should attempt to branch on general split disjunctions during the branch and bound search ↩

Note Split disjunctions are a special form of disjunctions that can be written as $\sum$j mj xj $\leq$ m0 v $\sum$j mj xj $\geq$ m0+1 The split disjunctions created by the optimizer will use a combination of binary or integer variables xj, with integer coefficients mj. Split disjunctions for branching will always be created with a default priority value of 400 instead of the default value of 500 for regular entity branches.

Default: auto

value	meaning
-1	Automatic selection of the strategy.
0	Disabled.
1	Cautious strategy. Disjunctive branches will be created only for general integers with a wide range.
2	Moderate strategy.
3	Aggressive strategy. Disjunctive branches will be created for both binaries and integers.

branchStructural *(integer)*: Branch and Bound: Determines whether the optimizer should search for special structure in the problem to branch on during the branch and bound search ↩

Structural branches will often involve branching on more than a single MIP entity at a time. As a result of a structural branch, a parent node could therefore end up with more than two child nodes, unlike the standard single entity branches. Structural branches will always be created with a default priority value of 400 instead of the default value of 500 for regular entity branches.

Default: auto

value	meaning
-1	Automatically determined.
0	Disabled.
1	Enabled.

breadthFirst *(integer)*: Number of nodes to include in the best-first search before switching to the local first search (nodeSelection = 4) ↩

Range: {0, ..., ∞}

Default: 11

choleskyAlg *(integer)*: Newton barrier: Type of Cholesky factorization used ↩

Default: auto

value	meaning
bit 0 = 1	Matrix blocking: 0: automatic setting; 1: manual setting.
bit 1 = 2	If manual selection of matrix blocking: 0: multi-pass; 1: single-pass.
bit 2 = 4	Nonseparable QP relaxation: 0: off; 1: on.
bit 3 = 8	Corrector weight: 0: automatic setting; 1: manual setting.
bit 4 = 16	If manual selection of corrector weight: 0: off; 1: on.
bit 5 = 32	Refinement: 0: automatic setting; 1: manual setting.
bit 6 = 64	Preconditioned conjugate gradient method (PCGM): 0: PCGM off; 1: PCGM on.
bit 7 = 128	Preconditioned quasi minimal residual (QMR) to refine solution: 0: QMR off; 1: QMR on.
bit 8 = 256	Perform refinement on the augmented system 0: off; 1: on.
bit 9 = 512	Force highest accuracy in refinement 0: off; 1: on.

choleskyTol *(real)*: Newton barrier: Tolerance for pivot elements in the Cholesky decomposition of the normal equations coefficient matrix, computed at each iteration of the barrier algorithm ↩

If the absolute value of the pivot element is less than or equal to choleskyTol, it merits special treatment in the Cholesky decomposition process.

Range: [0, ∞]

Default: 1e-15

clamping *(integer)*: Allows for the adjustment of returned solution values such that they are always within bounds ↩

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0

value	meaning
-1	Determined automatically.
0	Equivalent to clamping_primal.
1	Equivalent to clamping_dual.
2	Equivalent to clamping_slacks.
3	Equivalent to clamping_rdj.

clamping_dual *(boolean)*: Adjust primal slack values to always be within constraint bounds ↩

See also clamping.

Default: `0`

clamping_primal *(boolean)*: Adjust primal solution to always be within primal bounds ↩

Slacks if provided will be adjusted accordingly.

See also clamping.

Default: `0`

clamping_rdj *(boolean)*: Adjust reduced costs to always be within dual bounds implied by the primal solution ↩

See also clamping.

Default: `0`

clamping_slacks *(boolean)*: Adjust dual solution to always be within the dual bounds implied by the slacks ↩

Reduced costs, if provided, will be adjusted accordingly.

See also clamping.

Default: `0`

concurrentThreads *(integer)*: Determines the number of threads used by the concurrent solver ↩

Default: `auto`

value	meaning
`-1`	Determined automatically
`>0`	Number of threads to use.

conflictCuts *(integer)*: Branch and Bound: Specifies how cautious or aggressive the optimizer should be when searching for and applying conflict cuts ↩

Conflict cuts are in-tree cuts derived from nodes found to be infeasible or cut off, which can be used to cut off other branches of the search tree.

Default: `auto`

value	meaning
`-1`	Automatic.
`0`	Disable conflict cuts.
`1`	Cautious application of conflict cuts.
`2`	Medium application of conflict cuts.
`3`	Aggressive application of conflict cuts.

coresPerCpu *(integer)*: Used to override the detected value of the number of cores on a CPU ↩

The cache size (either detected or specified via the CACHESIZE control) used in Barrier methods will be divided by this amount, and this scaled-down value will be the amount of cache allocated to each Barrier thread.

Range: {-1, ..., ∞}

Default: auto

coverCuts *(integer)*: Branch and Bound: Number of rounds of lifted cover inequalities at the top node ↩

A lifted cover inequality is an additional constraint that can be particularly effective at reducing the size of the feasible region without removing potential integral solutions. The process of generating these can be carried out a number of times, further reducing the feasible region, albeit incurring a time penalty. There is usually a good payoff from generating these at the top node, since these inequalities then apply to every subsequent node in the tree search.

Range: {-1, ..., ∞}

Default: auto

cpuPlatform *(integer)*: Newton Barrier: Selects the AMD, Intel x86 or ARM vectorization instruction set that Barrier should run optimized code for ↩

On AMD / Intel x86 platforms the SSE2, AVX and AVX2 instruction sets are supported while on ARM platforms the NEON architecture extension can be activated.

Generic code, SSE2 and AVX optimized code will all result in the same solution path. Using AVX2 or NEON might result in a different solution path.

Default: -1

value	meaning
-2	Highest supported [Generic, SSE2, AVX or AVX2].
-1	Highest supported solve path consistent code [Generic, SSE2 or AVX].
0	Use generic code compatible with all CPUs.
1	Use SSE2 / NEON optimized code.
2	Use AVX optimized code.
3	Use AVX2 optimized code.

cpuTime *(integer)*: How time should be measured when timings are reported in the log and when checking against time limits ↩

Default: 0

value	meaning
-1	Disable the timer.
0	Use elapsed time.
1	Use process time.

crash *(integer)*: Simplex: Determines the type of crash used when the algorithm begins ↩

During the crash procedure, an initial basis is determined which is as close to feasibility and triangularity as possible. A good choice at this stage will significantly reduce the number of iterations required to find an optimal solution. The possible values increase proportionally to their time-consumption.

For primal simplex the non-bit-vector choices are relevant.

For dual simplex the bit-vector choices are relevant.

Default: 2

value	meaning
0	Turns off all crash procedures.
1	For singletons only (one pass).
2	For singletons only (multi pass).
3	Multiple passes through the matrix considering slacks.
4	Multiple (≤ 10) passes through the matrix but only doing slacks at the very end.
bit 0 = 1	Perform standard crash.
bit 1 = 2	Perform additional numerical checks during crash.
bit 2 = 4	Extend the set of column candidates for crash.
bit 3 = 8	Extend the set of row candidates for crash.
bit 4 = 16	Force crash, i.e., consider all suitable columns/rows as candidates for crash.
n>10	As for value 4 but performing at most n - 10 passes.

crossover *(integer)*: Newton barrier: Determines whether the barrier method will cross over to the simplex method when at optimal solution has been found, to provide an end basis and advanced sensitivity analysis information ↩

The full primal and dual solution is available whether or not crossover is used. The crossover must not be disabled if the barrier is used to reoptimize nodes of a MIP. By default crossover will not be performed on QP and MIQP problems.

Default: auto

value	meaning
-1	Determined automatically.
0	No crossover.
1	Primal crossover first.
2	Dual crossover first.

crossoverAccuracyTol *(real)*: Newton barrier: Determines how crossover adjusts the default relative pivot tolerance ↩

When re-inversion is necessary, crossover will compare the recalculated working basic solution with the assumed ones just before re-inversion took place. If the error is above this threshold, crossover will adjust the relative pivot tolerance to address the build-up of numerical inaccuracies.

The full primal and dual solution is available whether or not crossover is used. The crossover must not be disabled if the barrier is used to reoptimize nodes of a MIP. By default crossover will not be performed on QP and MIQP problems.

Range: [0, ∞]

Default: 1e-06

crossoverIterLimit *(integer)*: Newton barrier: Maximum number of iterations that will be performed in the crossover procedure before the optimization process terminates ↩

Range: {0, ..., ∞}

Default: 2147483647

crossoverOps *(integer)*: Newton barrier: Bit vector for adjusting the behavior of the crossover procedure ↩

Default: 0

value	meaning
bit 0 = 1	Returned solution when the crossover terminates prematurely: 0: Return the last basis from the crossover; 1: Return the barrier solution.
bit 1 = 2	Select the crossover stages to be performed: 0: Perform both crossover stages; 1: Skip second crossover stage.
bit 2 = 4	Set crossover behaviour: 0: Force to perform all pivots; 1: Skip pivots that are numerically less reliable.
bit 3 = 8	Set crossover behaviour: 0: Perform standard crossover; 1: Perform a slower, but numerically more careful crossover.

crossoverThreads *(integer)*: Determines the maximum number of threads that parallel crossover is allowed to use ↩

If crossoverThreads is set to the default value (-1), the barThreads control will determine the number of threads used.

Range: {-1, ..., ∞}

Default: auto

cutDepth *(integer)*: Branch and Bound: Sets the maximum depth in the tree search at which cuts will be generated ↩

Generating cuts can take a lot of time, and is often less important at deeper levels of the tree since tighter bounds on the variables have already reduced the feasible region. A value of 0 signifies that no cuts will be generated.

Does not affect cutting on the root node.

Range: {-1, ..., ∞}

Default: auto

cutFactor *(real)*: Limit on the number of cuts and cut coefficients the optimizer is allowed to add to the matrix during tree search ↩

The cuts and cut coefficients are limited by cutFactor times the number of rows and coefficients in the initial matrix.

A value of 0.0 prevents cuts from being added, and a value of e.g. 1.0 will allow the problem to grow to twice the initial number of rows and coefficients.

Default: auto

value	meaning
-1	Let the optimizer decide on the maximum amount of cuts based on cutStrategy.
>=0	Multiple of number of rows and coefficients to use.

cutFreq *(integer)*: Branch and Bound: Frequency at which cuts are generated in the tree search ↩

If the depth of the node modulo cutFreq is zero, then cuts will be generated.

Range: {-1, ..., ∞}

Default: auto

cutSelect *(integer)*: Bit vector providing detailed control of the cuts created for the root node of a MIP solve ↩

Use treeCutSelect to control cuts during the tree search.

The default value is -1 which enables all bits. Any bits not listed in the above table should be left in their default 'on' state, since the interpretation of such bits might change in future versions of the optimizer.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: -1

value	meaning
bit 5 = 32	Equivalent to cutSelect_clique.
bit 6 = 64	Equivalent to cutSelect_mir.
bit 7 = 128	Equivalent to cutSelect_cover.
bit 8 = 256	Equivalent to cutSelect_mirRowAggregation.
bit 11 = 2048	Equivalent to cutSelect_flowpath.
bit 12 = 4096	Equivalent to cutSelect_implication.
bit 13 = 8192	Equivalent to cutSelect_liftAndProject.
bit 14 = 16384	Equivalent to cutSelect_disableCutRows.
bit 15 = 32768	Equivalent to cutSelect_gubCover.
bit 16 = 65536	Equivalent to cutSelect_zeroHalf.
bit 17 = 131072	Equivalent to cutSelect_indicator.
bit 18 = 262144	Equivalent to cutSelect_gomory.
bit 20 = 1048576	Equivalent to cutSelect_farkas.

cutSelect_clique *(boolean)*: Clique cuts ↩

See also cutSelect.

Default: 1

cutSelect_cover *(boolean)*: Lifted cover cuts ↩

See also cutSelect.

Default: 1

cutSelect_disableCutRows *(boolean)*: Disable cutting from cut rows ↩

See also cutSelect.

Default: 1

cutSelect_farkas *(boolean)*: Farkas cuts ↩

See also cutSelect.

Default: `1`

cutSelect_flowpath *(boolean)*: Flow path cuts ↩

See also cutSelect.

Default: `1`

cutSelect_gomory *(boolean)*: Strong Chvatal-Gomory cuts ↩

See also cutSelect.

Default: `1`

cutSelect_gubCover *(boolean)*: Lifted GUB cover cuts ↩

See also cutSelect.

Default: `1`

cutSelect_implication *(boolean)*: Implication cuts ↩

See also cutSelect.

Default: `1`

cutSelect_indicator *(boolean)*: Indicator constraint cuts ↩

See also cutSelect.

Default: `1`

cutSelect_liftAndProject *(boolean)*: Turn on automatic Lift-and-Project cutting strategy ↩

See also cutSelect.

Default: `1`

cutSelect_mir *(boolean)*: Mixed Integer Rounding (MIR) cuts ↩

See also cutSelect.

Default: `1`

cutSelect_mirRowAggregation *(boolean)*: Turn on row aggregation for MIR cuts ↩

See also cutSelect.

Default: `1`

cutSelect_zeroHalf *(boolean)*: Zero-half cuts ↩

See also cutSelect.

Default: `1`

cutStrategy *(integer)*: Branch and Bound: Cut strategy ↩

A more aggressive cut strategy, generating a greater number of cuts, will result in fewer nodes to be explored, but with an associated time cost in generating the cuts. The fewer cuts generated, the less time taken, but the greater subsequent number of nodes to be explored.

Default: `auto`

value	meaning
-1	Automatic selection of the cut strategy.
0	No cuts.
1	Conservative cut strategy.
2	Moderate cut strategy.
3	Aggressive cut strategy.

defaultAlg *(integer)*: Selects the algorithm that will be used to solve the LP ↩

Please note that this will affect how the MIP node LP problems are solved during the global search.

Default: auto

value	meaning
1	Automatically determined.
2	Dual simplex.
3	Primal simplex.
4	Newton barrier.

denseColLimit *(integer)*: Newton barrier: Controls trigger point for special treatment of dense columns in Cholesky factorization ↩

Columns with more than denseColLimit elements are considered to be dense. Such columns will be handled specially in the Cholesky factorization of this matrix.

Range: {0, ..., ∞}

Default: auto

deterministic *(integer)*: Selects whether to use a deterministic or opportunistic mode when solving a problem using multiple threads ↩

In deterministic mode thread synchronization is performed in deterministic order, which guarantees that repeated solves of the same problems under the same starting conditions will always produce the same outcome. This assumes that there are no non-deterministic events affecting the solve, such as interruption due to time limits or non-deterministic interaction through callback functions. In opportunistic mode the solver will always schedule work on any available thread. This can produce higher CPU utilization, but will sacrifice reproducability.

Default: 1

value	meaning
0	Use opportunistic mode.
1	Use deterministic mode.
2	Use deterministic mode, except allow the initial concurrent continuous solve of a MIP to be opportunistic.

dualGradient *(integer)*: Simplex: Dual simplex pricing method ↩

Default: auto

value	meaning
-1	Determine automatically.
0	Devex.
1	Steepest edge.
2	Direct steepest edge.
3	Sparse Devex.

dualize *(integer)*: Whether presolve should form the dual of the problem ↩

Default: auto

value	meaning
-1	Determine automatically.
0	Solve the primal problem.
1	Solve the dual problem.

dualizeOps *(boolean)*: Bit-vector control for adjusting the behavior when a problem is dualized ↩

Default: 1

value	meaning
bit 0 = 1	Swap the simplex algorithm to run. If dual simplex is selected for the original problem then primal simplex will be run on the dualized problem, and simiarly if primal simplex is selected.

dualPerturb *(real)*: Factor by which the problem will be perturbed prior to optimization by dual simplex ↩

A value of 0.0 results in no perturbation prior to optimization. Note the interconnection to the autoPerturb control. If autoPerturb is set to 1, the decision whether to perturb or not is left to the Optimizer. When the problem is automatically perturbed in dual simplex, however, the value of dualPerturb will be used for perturbation.

Range: [-∞, ∞]

Default: auto

dualStrategy *(integer)*: Bit-vector control specifies the dual simplex strategy ↩

Default: 1

value	meaning
bit 0 = 1	Switch to primal when re-optimization goes dual infeasible and numerically unstable.
bit 1 = 2	When dual intend to switch to primal, stop the solve instead of switching to primal.
bit 2 = 4	Use more aggressive cut-off in MIP search.
bit 3 = 8	Use dual simplex to remove cost perturbations.
bit 4 = 16	Enable more aggressive dual pivoting strategy.
bit 5 = 32	Keep using dual simplex even when it's numerically unstable.

dualThreads *(integer)*: Determines the maximum number of threads that dual simplex is allowed to use ↩

If dualThreads is set to the default value (-1), the threads control will determine the number of threads used.

When solving a linear MIP, the dual simplex algorithm will use multiple threads only when solving the initial LP relaxation or when reoptimizing between rounds of cuts on the root node. The parallel dual simplex algorithm differs from the sequential dual simplex algorithm and might follow a different solve path. For dualThreads > 1 the solve path is independent of the number of threads used, although the practical limit for observing performance benefits is around dualThreads $= 8$.

Range: {`-1`, ..., ∞}

Default: `auto`

eigenvalueTol *(real)*: Quadratic matrix is considered not to be positive semi-definite, if its smallest eigenvalue is smaller than the negative of this value ↩

Range: [`0`, ∞]

Default: `1e-06`

elimFillin *(integer)*: Amount of fill-in allowed when performing an elimination in presolve ↩

Range: {`0`, ..., ∞}

Default: `10`

elimTol *(real)*: Markowitz tolerance for the elimination phase of the presolve ↩

Range: [`0`, ∞]

Default: `0.001`

etaTol *(real)*: Tolerance on eta elements ↩

During each iteration, the basis inverse is premultiplied by an elementary matrix, which is the identity except for one column - the eta vector. Elements of eta vectors whose absolute value is smaller than etaTol are taken to be zero in this step.

Range: [`0`, ∞]

Default: `1e-13`

feasibilityJump *(boolean)*: MIP: Decides if the Feasibility Jump heuristic should be run ↩

Default: `1`

value	meaning
`0`	Turned off.
`1`	Run the heuristic.

feasibilityPump *(integer)*: Branch and Bound: Decides if the Feasibility Pump heuristic should be run at the top node ↩

Default: `auto`

value	meaning
`-1`	Automatic.
`0`	Turned off.
`1`	Always try the Feasibility Pump.
`2`	Try the Feasibility Pump only if other heuristics have failed to find an integer solution.

feasTol *(real)*: Determines when a solution is treated as feasible ↩

If the amount by which a constraint's activity violates its right-hand side or ranged bound is less in absolute magnitude than feasTol, then the constraint is treated as satisfied. Similarly, if the amount by which a column violates its bounds is less in absolute magnitude than feasTol, those bounds are also treated as satisfied.

Range: [0, ∞]

Default: `1e-06`

feasTolPerturb *(real)*: Determines how much a feasible primal basic solution is allowed to be perturbed when performing basis changes ↩

The tolerance feasTol is always considered as an upper limit for the perturbations, but in some cases smaller value can be more desirable.

Range: [0, ∞]

Default: `1e-06`

feasTolTarget *(real)*: Target feasibility tolerance for the solution refiner ↩

Zero and negative values are ignored, and the value of feasTol is used.

Use very small values like 1e-100 to state the refinement should continue as long as an improvement is made. Use very large values like 1e+100 to disable only this aspect of the refiner.

Refining solutions to match the feastoltarget can influence and worsen their objective value in case the previous objective could only be achieved through slight infeasibilities.

Range: [-∞, ∞]

Default: `0`

fixoptfile *(string)*: name of option file which is read just before solving the fixed problem ↩

forceParallelDual *(boolean)*: Dual simplex: Specifies whether the dual simplex solver should always use the parallel simplex algorithm ↩

By default, when using a single thread, the dual simplex solver will execute a dedicated sequential simplex algorithm.

Default: `0`

value	meaning
0	Disabled.
1	Enabled. Force the dual simplex solver to use the parallel algorithm.

genConsAbsTransformation *(integer)*: Specifies the reformulation method for absolute value general constraints at the beginning of the search ↩

Default: auto

value	meaning
-1	Automatic.
0	Use a formulation based on indicator constraints.
1	Use a formulation based on SOS1-contraints.

genConsDualReductions *(boolean)*: Parameter specifies whether dual reductions should be applied to reduce the number of columns and rows added when transforming general constraints to MIP structs ↩

Default: 1

value	meaning
0	Disabled. No dual reductions, add columns and rows.
1	Enabled. Only add neccessary columns and rows, drop those implied by the objective sense.

globalBoundingBox *(real)*: If a nonlinear problem cannot be solved due to appearing unbounded, it can automatically be regularized by the application of a bounding box on the variables ↩

If this control is set to a negative value, in a second solving attempt all original variables will be bounded by the absolute value of this control. If set to a positive value, there will be a third solving attempt afterwards, if necessary, in which also all auxiliary variables are bounded by this value.

Default: 1e+06

value	meaning
0	Disabled. Problem will return unbounded.
n<0	Enabled. Apply lower and upper bounds of this magnitude to all original variables if initial LP is unbounded.
n>0	Enabled. Apply lower and upper bounds of this magnitude to all original and auxiliary variables if initial LP and first regularization are unbounded.

gomCuts *(integer)*: Branch and Bound: Number of rounds of Gomory or lift-and-project cuts at the top node ↩

Range: {-1, ..., ∞}

Default: auto

heurBeforeLp *(integer)*: Branch and Bound: Determines whether primal heuristics should be run before the initial LP relaxation has been solved ↩

It is possible that a heuristic will find an optimal integer solution that will result in the LP relaxation solution being cut off. If dedicated heuristic threads are enabled through the heurThreads control, then the initial heuristics will be run in parallel with the LP solve, instead of before.

Default: `auto`

value	meaning
`-1`	Automatic - let the optimizer decide if heuristics should be run.
`0`	Disabled.
`1`	Enabled.

heurDiveIterLimit *(real)*: Branch and Bound: Simplex iteration limit for reoptimizing during the diving heuristic ↩

Default: `auto`

value	meaning
`0`	No iteration limit.
`>=1`	Fixed iteration limit.
`<0`	Automatic selection of the iteration limit based on the problem size. The absolute value is used as a multiplier on the automatic selection.

heurDiveRandomize *(real)*: Level of randomization to apply in the diving heuristic ↩

The diving heuristic uses priority weights on rows and columns to determine the order in which to e.g. round fractional columns, or the direction in which to round them. This control determines by how large a random factor these weights should be changed.

Default: `0`

value	meaning
`0.0-1.0`	Amount of randomization (0.0=none, 1.0=full)

heurDiveSoftRounding *(integer)*: Branch and Bound: Enables a more cautious strategy for the diving heuristic, where it tries to push binaries and integer variables to their bounds using the objective, instead of directly fixing them ↩

This can be useful when the default diving heuristics fail to find any feasible solutions.

Default: `auto`

value	meaning
`-1`	Automatic selection.
`0`	Do not use soft rounding.
`1`	Cautious use of the soft rounding strategy.
`2`	More aggressive use of the soft rounding strategy.

heurDiveSpeedUp *(integer)*: Branch and Bound: Changes the emphasis of the diving heuristic from solution quality to diving speed ↩

Default: `-1`

value	meaning
-2	Automatic selection biased towards quality
-1	Automatic selection biased towards speed.
0-4	Manual emphasis bias from emphasis on quality (0) to emphasis on speed (4).

heurDiveStrategy *(integer)*: Branch and Bound: Chooses the strategy for the diving heuristic ↩

Default: auto

value	meaning
-1	Automatic selection of strategy.
0	Disables the diving heuristic.
1-18	Available pre-set strategies for rounding infeasible MIP entities and reoptimizing during the heuristic dive.

heurEmphasis *(integer)*: Branch and Bound: Specifies an emphasis for the search w.r.t. primal heuristics and other procedures that affect the speed of convergence of the primal-dual gap ↩

For problems where the goal is to achieve a small gap but not neccessarily solving them to optimality, it is recommended to set heurEmphasis to 1. This setting triggers many additional heuristic calls, aiming for reducing the gap at the beginning of the search, typically at the expense of an increased time for proving optimality.

Default: -1

value	meaning
-1	Optimizer default strategy.
0	Disables all heuristics.
1	Focus on reducing the primal-dual gap in the early part of the search.
2	Extremely aggressive search heuristics.

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

Deactivated by default.

Default: 0

value	meaning
0	Disabled.
1	Enabled. Run special objective heuristics on large problems and even if incumbent exists.

heurFreq *(integer)*: Branch and Bound: Frequency at which heuristics are used in the tree search ↩

Heuristics will only be used at a node if the depth of the node is a multiple of heurFreq.

Range: {-1, ..., ∞}

Default: -1

heurSearchEffort *(real)*: Adjusts the overall level of the local search heuristics ↩

heurSearchEffort is used as a multiplier on the default amount of work the local search heuristics should do. A higher value means the local search heuristics will be run more often and that they are allowed to search larger neighborhoods.

Range: [0, ∞]

Default: 1

heurSearchFreq *(integer)*: Branch and Bound: How often the local search heuristic should be run in the tree ↩

Default: auto

value	meaning
-1	Automatic.
0	Disabled in the tree.
n>0	Number of nodes between each run.

heurSearchRootCutFreq *(integer)*: How frequently to run the local search heuristic during root cutting ↩

This is given as how many cut rounds to perform between runs of the heuristic. Set to zero to avoid applying the heuristic during root cutting. Branch and Bound: This specifies how often the local search heuristic should be run in the tree.

Default: auto

value	meaning
-1	Automatic.
0	Disabled heuristic during cutting.
n>0	Number of cutting rounds between each run.

heurSearchRootSelect *(integer)*: Bit vector control for selecting which local search heuristics to apply on the root node of a MIP solve ↩

Use heurSearchTreeSelect to control local search heuristics during the tree search.

Some of the local search heuristics will benefit from having an existing incumbent solution, but it is not required.

Default: 117

value	meaning
bit 0 = 1	Local search with a large neighborhood. Potentially slow but is good for finding solutions that differs significantly from the incumbent.
bit 1 = 2	Local search with a small neighborhood centered around a node LP solution.
bit 2 = 4	Local search with a small neighborhood centered around an integer solution. This heuristic will often provide smaller, incremental improvements to an incumbent solution.
bit 3 = 8	Local search with a neighborhood set up through the combination of multiple integer solutions.

value	meaning
`bit 4 = 16`	Unused
`bit 5 = 32`	Local search without an objective function. Called seldom and only when no feasible solution is available.
`bit 6 = 64`	Local search with an auxiliary objective function. Called seldom and only when no feasible solution is available.

heurSearchTreeSelect *(integer)*: Bit vector control for selecting which local search heuristics to apply during the tree search of a MIP solve ↩

Use heurSearchRootSelect to control local search heuristics on the root node.

Some of the local search heuristics will benefit from having an existing incumbent solution, but it is not required.

Default: `17`

value	meaning
`bit 0 = 1`	Local search with a large neighborhood. Potentially slow but is good for finding solutions that differs significantly from the incumbent.
`bit 1 = 2`	Local search with a small neighborhood centered around a node LP solution.
`bit 2 = 4`	Local search with a small neighborhood centered around an integer solution. This heuristic will often provide smaller, incremental improvements to an incumbent solution.
`bit 3 = 8`	Local search with a neighborhood set up through the combination of multiple integer solutions.
`bit 4 = 16`	Unused
`bit 5 = 32`	Local search without an objective function. Called seldom and only when no feasible solution is available.
`bit 6 = 64`	Local search with an auxiliary objective function. Called seldom and only when no feasible solution is available.

heurThreads *(integer)*: Branch and Bound: Number of threads to dedicate to running heuristics on the root node ↩

When heuristic threads are enable, the heuristics will be run in parallel with the initial LP solve, if possible, and in parallel with the root cutting.

Default: `0`

value	meaning
`-1`	Automatically determined from the threads control.
`0`	Disabled. Heuristics will be run sequentially with the root LP solve and cutting.
`>=1`	Number of root threads to dedicate to parallel heuristics.

historyCosts *(integer)*: Branch and Bound: How to update the pseudo cost for a MIP entity when a strong branch or a regular branch is applied ↩

Default: `auto`

value	meaning
`-1`	Automatically determined.

value	meaning
0	No update.
1	Update using only regular branches from the root to the current node.
2	Same as 1, but update with strong branching results as well.
3	Update using any regular branching or strong branching information from all nodes solves before the current node.

ifCheckConvexity *(boolean)*: Determines if the convexity of the problem is checked before optimization ↩

Applies to quadratic, mixed integer quadratic and quadratically constrained problems. Checking convexity takes some time, thus for problems that are known to be convex it might be reasonable to switch the checking off.

Default: 1

value	meaning
0	Turn off convexity checking.
1	Turn on convexity checking.

indLinBigM *(real)*: During presolve, indicator constraints will be linearized using a BigM coefficient whenever that BigM coefficient is small enough ↩

This control defines the largest BigM for which such a linearized version will be added to the problem in addition to the original constraint. If the BigM is even smaller than indPreLinBigM, then the original indicator constraint will additionally be dropped from the problem.

indLinBigM should always be at least as large as indPreLinBigM. For any value less or equal to indPreLinBigM, indicator constraints will never be duplicated and only indPreLinBigM is taken into account for linearization.

Range: [0, ∞]

Default: 100000

indPreLinBigM *(real)*: During presolve, indicator constraints will be linearized using a BigM coefficient whenever that BigM coefficient is small enough ↩

This control defines the largest BigM for which the original constraint will be replaced by the linearized version. If the BigM is larger than indPreLinBigM but smaller than indLinBigM, the linearized row will be added but the original indicator constraint is kept as a numerically stable way to check feasibility.

Replacing an indicator constraint with a BigM row has a side effect on tolerances. In the indicator constraint form, the constraint part is satisfied with feasTol tolerance; while after changing it to BigM form, the constraint also includes the binary indicator variable (with a coefficient up to indPreLinBigM and an integrality tolerance of mipTol), therefore the constraint part of the indicator contraint is satisfied with tolerance feasTol+mipTol∗indPreLinBigM.

Range: [0, ∞]

Default: 100

inputTol *(real)*: Tolerance on input values elements ↩

If any value is less than or equal to inputTol in absolute value, it is treated as zero. For the internal zero tolerance see matrixTol.

This control needs to be set before inputting the problem, as it has no effect afterwards.

Range: [0, ∞]

Default: 0

invertFreq *(integer)*: Simplex: Frequency with which the basis will be inverted ↩

The basis is maintained in a factorized form and on most simplex iterations it is incrementally updated to reflect the step just taken. This is considerably faster than computing the full inverted matrix at each iteration, although after a number of iterations the basis becomes less well-conditioned and it becomes necessary to compute the full inverted matrix. The value of invertFreq specifies the maximum number of iterations between full inversions.

Range: {-1, ..., ∞}

Default: auto

invertMin *(integer)*: Simplex: Minimum number of iterations between full inversions of the basis matrix ↩

See the description of invertFreq for details.

Range: {0, ..., ∞}

Default: 3

ioTimeout *(integer)*: Maximum number of seconds to wait for an I/O operation before it is cancelled ↩

Range: {0, ..., ∞}

Default: 30

knitroOptFile *(string)*: Option file for NLP solver KNITRO ↩

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

Range: {0, ..., ∞}

Default: 50

lnpIterLimit *(integer)*: Number of iterations to perform in improving each lift-and-project cut ↩

By setting the number to zero a Gomory cut will be created instead.

Range: {-1, ..., ∞}

Default: auto

loadMipSol *(boolean)*: Loads a MIP solution (the initial point) ↩

If true, the initial point provided by GAMS will be passed to the optimizer to be treated as an integer feasible point. The optimizer uses the values for the discrete variables only: the level values for the continuous variables are ignored and are calculated by fixing the integer variables and reoptimizing. In some cases, loading an initial MIP solution can improve performance. In addition, there will always be a feasible solution to return.

Default: 0

localChoice *(integer)*: Controls when to perform a local backtrack between the two child nodes during a dive in the branch and bound tree ↩

Default: auto

value	meaning
1	Never backtrack from the first child, unless it is dropped (infeasible or cut off).
2	Always solve both child nodes before deciding which child to continue with.
3	Automatically determined.

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

When more than one bit is set, then the LP problem will be solved with the concurrent solver. When this control and `algorithm` is set at the same time, `algorithm` will overrule the value of this control.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: `0`

value	meaning
`bit 0 = 1`	Equivalent to lpFlags_dual.
`bit 1 = 2`	Equivalent to lpFlags_primal.
`bit 2 = 4`	Equivalent to lpFlags_barrier.
`bit 3 = 8`	Equivalent to lpFlags_network.

lpFlags_barrier *(boolean)*: Use the barrier method ↩

See also lpFlags.

Default: `0`

lpFlags_dual *(boolean)*: Use the dual simplex method ↩

See also lpFlags.

Default: `0`

lpFlags_network *(boolean)*: Use the network simplex method ↩

See also lpFlags.

Default: `0`

lpFlags_primal *(boolean)*: Use the primal simplex method ↩

See also lpFlags.

Default: `0`

lpFolding *(integer)*: Simplex and barrier: Whether to fold an LP problem before solving it ↩

Default: `auto`

value	meaning
-1	Automatic.
0	Disable LP folding.
1	Enable LP folding. Attempt to fold all LP problems and MIP initial relaxations.

lpIterLimit *(integer)*: Maximum number of iterations that will be performed by primal simplex or dual simplex before the optimization process terminates ↩

Synonym: iterlim

For MIP problems, this is the maximum total number of iterations over all nodes explored by the Branch and Bound method.

Range: {0, ..., ∞}

Default: `maxint`

lpLog *(integer)*: Simplex: Frequency at which the simplex log is printed ↩

This control only has an effect if lpLogStyle is set to 0.

Default: `100`

value	meaning
0	Log displayed at the end of the optimization only.
n<0	Detailed output every - n iterations.
n>0	Summary output every n iterations.

lpLogDelay *(real)*: Time interval between two LP log lines ↩

This control only has an effect if lpLogStyle is set to 1.

Range: [0, ∞]

Default: `1`

lpLogStyle *(boolean)*: Simplex: Style of the simplex log ↩

Default: `1`

value	meaning
0	Simplex log is printed based on simplex iteration count, at a fixed frequency as specified by the lpLog control.
1	Simplex Log is printed based on an estimation of elapsed time, determined by an internal deterministic timer.

lpRefineIterLimit *(integer)*: Simplex iteration limit the solution refiner can spend in attempting to increase the accuracy of an LP solution ↩

The solution refiner iteratively attempts to increase the accuracy of the solution until either both feasTolTarget and optimalityTolTarget is satisfied, or accuracy cannot further be increased, or the effort limit determined by lpRefineIterLimit is exhausted.

Range: {-1, ..., ∞}

Default: `auto`

markowitzTol *(real)*: Markowitz tolerance used for the factorization of the basis matrix ↩

Range: [0, ∞]

Default: 0.01

matrixTol *(real)*: Zero tolerance on matrix elements ↩

If the value of a matrix element is less than or equal to matrixTol in absolute value, it is treated as zero. The control applies when solving a problem, for an input tolerance see inputTol.

Range: [0, ∞]

Default: 1e-09

maxCutTime *(real)*: Maximum amount of time allowed for generation of cutting planes and reoptimization ↩

The limit is checked during generation and no further cuts are added once this limit has been exceeded.

Default: 0

value	meaning
0	No time limit.
>0	Stop cut generation after the given number of seconds.

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

For numerically challenging MIP problems, it can sometimes help make the solve more stable by reducing the value of maxImpliedBound to something smaller - e.g. 1.0E+06. It is not recommended to increase this parameter beyond the default of 1.0E+08.

Range: [0, ∞]

Default: 1e+08

maxLocalBacktrack *(integer)*: Branch-and-Bound: How far back up the current dive path the optimizer is allowed to look for a local backtrack candidate node ↩

If this control is set to k, then the candidate set of nodes for a local backtrack will consist of all active nodes in the subtree rooted at height k above the current node. For example, a setting of 1 will result in only sibling nodes of the current node being considered.

Default: auto

value	meaning
-1	Automatic.
n>0	Local backtrack limit.

maxMemoryHard *(integer)*: Sets the maximum amount of memory in megabytes the optimizer should allocate ↩

If this limit is exceeded, the solve will terminate. This control is designed to make the optimizer stop in a controlled manner, so that the problem object is valid once termination occurs.

The solve state will be set to incomplete. This is different to an out of memory condition in which case the optimizer returns an error. The optimizer may still allocate memory once the limit is exceeded to be able to finsish the operations and stop in a controlled manner. When resourceStrategy is enabled, the control also has the same effect as maxMemorySoft and will cause the optimizer to try preserving memory when possible.

Range: {0, ..., ∞}

Default: `unlimited`

maxMemorySoft *(integer)*: When resourceStrategy is enabled, this control sets the maximum amount of memory in megabytes the optimizer targets to allocate ↩

This may change the solving path, but will not cause the solve to terminate early. To set a hard version of the same, please set maxMemoryHard.

Range: {0, ..., ∞}

Default: `unlimited`

maxMipSol *(integer)*: Branch and Bound: Limit on the number of integer solutions to be found by the Optimizer ↩

It is possible that during optimization the Optimizer will find the same objective solution from different nodes. However, maxMipSol refers to the total number of integer solutions found, and not necessarily the number of distinct solutions.

Setting maxMipSol=1 can alter the solution path as this will put the emphasis on finding any feasible solution by triggering additional heuristics.

Range: {0, ..., ∞}

Default: `0`

maxMipTasks *(integer)*: Branch-and-Bound: The maximum number of tasks to run in parallel during a MIP solve ↩

The MIP solver will create smaller tasks from individual active nodes or based on local search heuristics. These are tasks that will be executed in parallel by the number of threads set by mipThreads.

If maxMipTasks is set to a fixed, positive value, the branch-and-bound tree nodes will always be solved in the same deterministic way, independent of the actual number of executing threads implied by mipThreads. How a MIP is solved will still depend on the number of threads used for solving the continuous relaxation and therefore on the settings for the controls barThreads, barCores, dualThreads and concurrentThreads). To obtain a MIP solve that is completely independent of the number of threads, it is sufficient to set maxMipTasks, forceParallelDual, barThreads and barCores. The concurrent LP solver should be avoided in this case.

While you can set this control to large value, the implementation will limit the number of tasks for performance reasons. This limit is currently 32 on 32bit platforms and 256 on 64 bit platforms.

Default: `auto`

value	meaning
`-1`	Task limit determined automatically from mipThreads.
`>0`	Fixed task limit.

maxNode *(integer)*: Branch and Bound: Maximum number of nodes that will be explored ↩

Range: {0, ..., ∞}

Default: `maxint`

maxScaleFactor *(integer)*: Determines the maximum scaling factor that can be applied during scaling ↩

The maximum is provided as an exponent of a power of 2.

Default: `64`

value	meaning
0-64	The maximum is provided an exponent of a power of 2.

maxStallTime *(real)*: Maximum time in seconds that the Optimizer will continue to search for improving solution after finding a new incumbent ↩

Default: `0`

value	meaning
0	No stall time limit.
>0	If an integer solution has been found, stop MIP search after the given number of seconds without a new incumbent. No effect as long as no solution was found.

mipAbsCutoff *(real)*: Branch and Bound: If the user knows that they are interested only in values of the objective function which are better than some value, this can be assigned to mipAbsCutoff ↩

This allows the Optimizer to ignore solving any nodes which may yield worse objective values, saving solution time.

Range: [-∞, ∞]

Default: `auto`

mipAbsStop *(real)*: Branch and Bound: Absolute tolerance determining whether the tree search will continue or not ↩

It will terminate if | MIPOBJVAL - BESTBOUND| ≤ mipAbsStop where MIPOBJVAL is the value of the best solution's objective function, and BESTBOUND is the current best solution bound. For example, to stop the tree search when a MIP solution has been found and the Optimizer can guarantee it is within 100 of the optimal solution, set mipAbsStop to 100.

Range: [0, ∞]

Default: `0`

mipAddCutoff *(real)*: Branch and Bound: Amount to add to the objective function of the best integer solution found to give the new CURRMIPCUTOFF ↩

Once an integer solution has been found whose objective function is equal to or better than CURRMIPCUTOFF, improvements on this value may not be interesting unless they are better by at least a certain amount. If mipAddCutoff is nonzero, it will be added to CURRMIPCUTOFF each time an integer solution is found which is better than this new value. This cuts off sections of the tree whose solutions would not represent substantial improvements in the objective function, saving processor time. The control mipAbsStop provides a similar function but works in a different way.

Range: [-∞, ∞]

Default: `0`

mipCleanup *(boolean)*: Clean up the MIP solution (round-fix-solve) to get duals ↩

If nonzero, clean up the integer solution obtained, i.e. round and fix the discrete variables and re-solve as an LP to get some marginal values for the discrete vars.

Default: `1`

mipComponents *(integer)*: Determines whether disconnected components in a MIP should be solved as separate MIPs ↩

There can be significant performence benefits from solving disconnected components individual instead of being part of the main branch-and-bound search.

If there are no constraints linking two variables, either directly or indirectly through other variables, they are said to belong to two separate disconnected components. When a problem contains disconnected components of signficant size, it can be advantageous to solve each component as a separate MIP. When significant disconnected components are detected in the problem, the solver will switch to a different solve mode where each component is solved separately. This switch will happen after the root node processing has completed and when the solve is about to enter the branch-and-bound search.

Solving disconnected components separately is not compatible with concurrent MIP solves. If concurrent MIP solves has been turned off, disconnected components will be solved as part of the standard branch-and-bound search in each concurrent solve.

Disabling MIP dual reductions through mipDualReductions will also disable the separate solve of disconnected components.

Default: `auto`

value	meaning
`-1`	Automatic - let the solver decide.
`0`	Disable solving disconnected components separately.
`1`	Solve disconnected components separately.

mipConcurrentNodes *(integer)*: Sets the node limit for when a winning solve is selected when concurrent MIP solves are enabled ↩

When multiple MIP solves are started, they each run up to the mipConcurrentNodes node limit and only one winning solve is selected for contuinuing the search with.

Default: `auto`

value	meaning
-1	Automatic - let the solver decide on a node limit.
>0	Number of nodes each concurrent solve should complete before a winner is selected.

mipConcurrentSolves *(integer)*: Selects the number of concurrent solves to start for a MIP ↩

Each solve will use a unique random seed for its random number generator, but will otherwise apply the same user controls. The first concurrent solve to complete will have solved the MIP and all the concurrent solves will be terminated at this point. Using concurrent solves can be advantageous when a MIP displays a high level of performance variability.

A node limit is imposed on each concurrent solve, through mipConcurrentNodes. When a concurrent solve reaches this node limit, it will be suspended until all concurrent solves have reached the limit. At this point a winner will be declared, based on which solve made the most progress towards optimality and only the winning solve will continue, using all threading resources. If a concurrent solve completes its MIP search before reaching the node limit, all solves will be stopped.

Default: 0

value	meaning
-1	Enabled. The number of concurrent solves depends on mipThreads.
n>1	Enabled. The number of concurrent solves to start is given by n.
0, 1	Disabled

mipDualReductions *(integer)*: Branch and Bound: Limits operations that can reduce the MIP solution space ↩

The mipDualReductions control, when set to a value different from 1 will adjust the values of other controls in order to prevent MIP solver operations that can result in the removal of dominated solutions. For example, dual reductions during preprocessing attempts to remove dominated solutions based on objective arguments, assuming that all constraints are known to the Optimizer. If a problem is detected to have symmetries, the solver might also remove some symmetrical solutions from the search space. In both cases, the set of feasible MIP solutions might be reduced. With default settings, it is only guaranteed that at least one optimal solution remains.

When attempting to collect the n-best solutions, it is recommended to set mipDualReductions=2. This will ensure that the only solutions missed by the enumeration are those that only differ from an existing solution in the values of the continuous variables.

Default: 1

value	meaning
0	Prevent all dual reductions.
1	Allow all dual reductions.
2	Allow dual reductions on continuous variables only.

mipFracReduce *(integer)*: Branch and Bound: Specifies how often the optimizer should run a heuristic to reduce the number of fractional integer variables in the node LP solutions ↩

This heuristic is only applicable to problems that are dual degenerate. These are problems that contain multiple solutions with identical objective function value. The more dual degenerate a problem is, the more likely it will be for this heuristic to have an improving effect.

Default: `auto`

value	meaning
`-1`	Automatic.
`0`	Disabled.
`1`	Run before and after cutting on the root node.
`2`	Run also during root cutting.
`3`	Run also during the tree search.

mipKappaFreq *(integer)*: Branch and Bound: Specifies how frequently the basis condition number (also known as kappa) should be calculated during the branch-and-bound search ↩

The condition number is calculated as the norm of the basis matrix multiplied by the norm of its inverse. This uses the Froebenius norm.

A summary will be printed at the end of the solve, summarizing the collected condition numbers collected:

- Nodes kappa stable: No. of stable sampled nodes (kappa < `10^7`)
- Nodes kappa suspicius: No. of suspicious sampled nodes (`10^7` ≤ kappa < `10^10`)
- Nodes kappa unstable: No. of unstable sampled nodes (`10^10` ≤ kappa < `10^13`)
- Nodes kappa ill-posed: No. of ill-posed sampled nodes (`10^13` ≤ kappa)
- Largest kappa seen: The largest condition number calculated through all sampled nodes.
- Kappa attention level: A measure of how ill-posed the problem is (between 0 and 1).

Default: `0`

value	meaning
`0`	Do not calculate condition numbers.
`1`	Calculate conditions numbers on every node, including after each round of root cutting.
`n>1`	Calculate a condition number once per node of every n'th level of the branch-and-bound tree.

mipLog *(integer)*: MIP log print control ↩

Default: `-100`

value	meaning
`0`	No printout during MIP tree search.
`1`	Only print out summary statement at the end.
`2`	Print out detailed log at all solutions found.
`3`	Print out detailed log at each node.
`-n`	Print out summary log at each n th node.

mipPresolve *(integer)*: Branch and Bound: Type of integer processing to be performed ↩

If set to 0, no processing will be performed.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: `-1`

value	meaning
`bit 0 = 1`	Equivalent to mipPresolve_reducedCostFixing.
`bit 1 = 2`	Equivalent to mipPresolve_logicPreprocessing.
`bit 2 = 4`	[Unused] This bit is no longer used to control probing. Refer to the integer control preProbing for setting probing level during presolve.
`bit 3 = 8`	Equivalent to mipPresolve_allowChangeBounds.
`bit 4 = 16`	Equivalent to mipPresolve_dualReductions.
`bit 5 = 32`	Equivalent to mipPresolve_globalCoefTightening.
`bit 6 = 64`	Equivalent to mipPresolve_objBasedReductions.
`bit 7 = 128`	Equivalent to mipPresolve_allowTreeRestart.
`bit 8 = 256`	Equivalent to mipPresolve_symmetryReductions.

mipPresolve_allowChangeBounds *(boolean)*: If node preprocessing is allowed to change bounds on continuous columns ↩

See also mipPresolve.

Default: `1`

mipPresolve_allowTreeRestart *(boolean)*: [Unused] This bit is no longer used to control restarts ↩

Refer to the integer control mipRestart for disabling tree restarts.

See also mipPresolve.

Default: `1`

mipPresolve_dualReductions *(boolean)*: Dual reductions will be performed at each node ↩

See also mipPresolve.

Default: `1`

mipPresolve_globalCoefTightening *(boolean)*: Allow global (non-bound) tightening of the problem during the tree search ↩

See also mipPresolve.

Default: `1`

mipPresolve_logicPreprocessing *(boolean)*: Primal reductions will be performed at each node ↩

Uses constraints of the node to tighten the range of variables, often resulting in fixing their values. This greatly simplifies the problem and may even determine optimality or infeasibility of the node.

See also mipPresolve.

Default: `1`

mipPresolve_objBasedReductions *(boolean)*: Objective function will be used to find reductions at each node ↩

See also mipPresolve.

Default: `1`

mipPresolve_reducedCostFixing *(boolean)*: Reduced cost fixing will be performed at each node ↩

This can simplify the node before it is solved, by deducing that certain variables' values can be fixed based on additional bounds imposed on other variables at this node.

See also mipPresolve.

Default: `1`

mipPresolve_symmetryReductions *(boolean)*: Allow that symmetry is used to presolve the node problem ↩

See also mipPresolve.

Default: `1`

mipRampUp *(integer)*: Controls the strategy used by the parallel MIP solver during the ramp-up phase of a branch-and-bound tree search ↩

The branch-and-bound tree search starts from the single root node, and only through branching on this root node and the resulting child nodes, are enough active nodes created to produce sufficient tasks to keep all MIP workers busy. This is referred to as the ramp-up phase of a parallel MIP. In a typical MIP solve, the solutions found during the initial dives will typically provide a significant improvement over the root heuristic solutions. It can therefore be advantageous to let these initial dives run as fast as possible, by limiting resource contention. This can be accomplished by restricting the number of parallel tasks and thereby reducing the memory bus contention. The mipRampUp control can be used to turn this initial task restriction of a parallel MIP solve on or off.

Default: `auto`

value	meaning
`-1`	Automatically determined.
`0`	No special treatment during the ramp-up phase. Always run with the maximal number of tasks.
`1`	Limit the number of tasks until the initial dives have completed.

mipRefineIterLimit *(integer)*: Defines an effort limit expressed as simplex iterations for the MIP solution refiner ↩

The limit is per reoptimizations in the MIP refiner.

Range: {`-1`, ..., ∞}

Default: `auto`

mipRelCutoff *(real)*: Branch and Bound: Percentage of the LP solution value to be added to the value of the objective function when an integer solution is found, to give the new value of CURRMIPCUTOFF ↩

The effect is to cut off the search in parts of the tree whose best possible objective function would not be substantially better than the current solution. The control mipRelStop provides a similar functionality but works in a different way.

Range: [`0`, ∞]

Default: `0`

mipRelStop *(real)*: Branch and Bound: Determines when the branch and bound tree search will terminate ↩

Branch and bound tree search will stop if: | MIPOBJVAL - BESTBOUND| ≤ mipRelStop x max(| BESTBOUND|,| MIPOBJVAL|) where MIPOBJVAL is the value of the best solution's objective function and BESTBOUND is the current best solution bound. For example, to stop the tree search when a MIP solution has been found and the Optimizer can guarantee it is within 5% of the optimal solution, set mipRelStop to 0.05.

This control is a stopping criteria only and different values of the control will not affect the solution path before termination. Unlike other stopping criteria, like time and node count, termination on mipRelStop will cause the final solution to be declared optimal and the problem to be returned to its original state.

Tolerances, such as mipRelCutoff and mipAbsCutoff, determine how much the objective value of a new MIP solution has to differ from the incumbent for it to be accepted. These controls therefore also influence the final gap at the end of a MIP solve.

Range: [0, ∞]

Default: `0.0001`

mipRestart *(integer)*: Branch and Bound: Controls strategy for in-tree restarts ↩

Default: `auto`

value	meaning
`-1`	Determined automatically (XPRS_MIPRESTART_DEFAULT).
`0`	Disable in-tree restarts (XPRS_MIPRESTART_OFF).
`1`	Allow in-tree restarts at normal aggressiveness (XPRS_MIPRESTART_MODERATE).
`2`	Allow in-tree restarts at higher aggressiveness (more likely to trigger a restart) (XPRS_MIPRESTART_AGGRESSIVE).

mipRestartFactor *(real)*: Branch and Bound: Fine tune initial conditions to trigger an in-tree restart ↩

Use a value > 1 to increase the aggressiveness with which the Optimizer restarts. Use a value < 1 to relax the aggressiveness with which the Optimizer restarts. Note that this control does not affect the initial condition on the gap, which must be set separately.

Range: [0, ∞]

Default: `1`

mipRestartGapThreshold *(real)*: Branch and Bound: Initial gap threshold to delay in-tree restart ↩

The restart is delayed initially if the gap, given as a fraction between 0 and 1, is below this threshold. The optimizer adjusts the threshold every time a restart is delayed. Note that there are other criteria that can delay or prevent a restart.

Range: [0, 1]

Default: `0.02`

mipstopexpr *(string)*: Stopping expression for branch and bound ↩

If the provided logical expression is true, the branch-and-bound is aborted. Supported values are: resusd, nodusd, objest, objval. Supported operators are: +, -, ∗, /, ^, %, !=, ==, <, <=, >, >=, !, &&, ||, (,), abs, ceil, exp, floor, log, log10, pow, sqrt . Example:
```
nodusd >= 1000 && abs(objest - objval) / abs(objval) < 0.1
```

If multiple stop expressions are given in an option file, the algorithm stops if any of them is true (|| concatenation).

mipThreads *(integer)*: If set to a positive integer it determines the number of threads implemented to run the parallel MIP code ↩

If mipThreads is set to the default value (-1), the threads control will determine the number of threads used.

Range: {`-1`, ..., ∞}

Default: `auto`

mipTol *(real)*: Branch and Bound: Tolerance within which a decision variable's value is considered to be integral ↩

Range: [`0`, ∞]

Default: `5e-06`

mipToltarget *(real)*: Target mipTol value used by the automatic MIP solution refiner as defined by refineOps ↩

Negative and zero values are ignored.

Refining solutions to match the miptoltarget can influence and worsen their objective value in case the previous objective could only be achieved through slight infeasibilities.

Range: [-∞, ∞]

Default: `0`

mipTrace *(string)*: Name of MIP trace file ↩

A miptrace file with the specified name will be created. This file records the best integer and best bound values every mipTraceNode nodes and at mipTraceTime-second intervals.

Default: `none`

mipTraceNode *(integer)*: Node interval between MIP trace file entries ↩

Default: `100`

mipTraceTime *(real)*: Time interval, in seconds, between MIP trace file entries ↩

Default: `5`

miqcpAlg *(integer)*: Determines which algorithm is to be used to solve mixed integer quadratic constrained and mixed integer second order cone problems ↩

Default: `auto`

value	meaning
-1	Determined automatically.
0	Use the barrier algorithm in the branch and bound algorithm.
1	Use outer approximations in the branch and bound algorithm.

mpsNameLength *(integer)*: Maximum length of MPS names in characters ↩

Internally it is rounded up to the smallest multiple of 8. MPS names are right padded with blanks. Maximum value is 64.

Range: {0, ..., ∞}

Default: 0

mpsOutputFile *(string)*: Name of MPS output file ↩

If specified XPRESS-MP will generate an MPS file corresponding to the GAMS model: the argument is the file name to be used. You can prefix the file name with an absolute or relative path.

Default: none

netCuts *(integer)*: Determines the addition of multi-commodity network cuts to a problem ↩

The parameter is defined as a bit string, and values 1, 2, 4 can be summed up if the user wants more classes of cuts to be added.

If the user wants to add both cut-set inequalities and lifted flow-cover inequalities but not node cut-set inequalities, the value of the control should be set to 1+4=5.

Default: 0

value	meaning
-1	Automatically determined.
0	Do not add these cuts.
1	Add cut-set inequalities.
2	Add node cut-set inequalities, i.e., cut-set inequalities that are based on a network cut defined on a single network node.
4	Add lifted flow-cover inequalities.

netStallLimit *(integer)*: Limit the number of degenerate pivots of the network simplex algorithm, before switching to either primal or dual simplex, depending on algAfterNetwork ↩

Default: auto

value	meaning
-1	Automatically determined limit
0	No limit.
n>0	Limit to n network simplex iterations.

nodeProbingEffort *(real)*: Adjusts the overall level of node probing ↩

nodeProbingEffort is used as a multiplier on the default amount of work node probing should do. Setting the control to zero disables node probing.

Range: [0, ∞]

Default: 1

nodeSelection *(integer)*: Branch and Bound: Determines which nodes will be considered for solution once the current node has been solved ↩

Default: auto

value	meaning
1	Local first: Choose between descendant and sibling nodes if available; choose from all outstanding nodes otherwise.
2	Best first: Choose from all outstanding nodes.
3	Local depth first: Choose between descendant and sibling nodes if available; choose from the deepest nodes otherwise.
4	Best first, then local first: Best first is used for the first breadthFirst nodes, after which local first is used.
5	Pure depth first: Choose from the deepest outstanding nodes.

numericalEmphasis *(integer)*: How much emphasis to place on numerical stability instead of solve speed ↩

Default: auto

value	meaning
-1	Automatic. The emphasis might be influenced by the setting of other controls.
0	Emphasize speed.
1	Mild emphasis on numerical stability.
2	Medium emphasis on numerical stability.
3	Strong emphasis on numerical stability.

objGoodEnough *(real)*: Stop once an objective this good is found ↩

Default: none

objScaleFactor *(integer)*: Custom objective scaling factor, expressed as a power of 2 ↩

When set, it overwrites the automatic objective scaling factor. A value of 0 means no objective scaling. This control is applied for the full solve, and is independent of any extra scaling that may occur specifically for the barrier or simplex solvers. As it is a power of 2, to scale by 16, set the value of the control to 4.

Range: {0, ..., ∞}

Default: 0

optimalityTol *(real)*: Simplex: Zero tolerance for reduced costs ↩

On each iteration, the simplex method searches for a variable to enter the basis which has a negative reduced cost. The candidates are only those variables which have reduced costs less than the negative value of optimalityTol.

Range: [0, ∞]

Default: 1e-06

optimalityTolTarget *(real)*: Target optimality tolerance for the solution refiner ↩

Zero and negative values are ignored, and the value of optimalityTol is used.

Use very small values like 1e-100 to state the refinement should continue as long as an improvement is made. Use very large values like 1e+100 to disable only this aspect of the refiner.

Refining solutions to match the optimalitytoltarget can influence and increase their infeasibility in case the previous feasibility could only be achieved through slight dual violations.

Range: [-∞, ∞]

Default: 0

outputControls *(boolean)*: Toggles the printing of all control settings at the beginning of the search ↩

This includes the printing of controls that have been explicitly assigned to their default value. All unset controls are omitted as they keep their default value.

Setting outputControls to 0 has no effect on the function XPRSdumpcontrols

Default: 1

value	meaning
0	Turn off printing of user-specified control settings.
1	Print controls.

outputLog *(integer)*: Controls the level of output produced by the Optimizer during optimization ↩

Default: 1

value	meaning
0	Turn all output off.
1	Print all messages.
3	Print error and warning messages.
4	Print error messages only.

outputTol *(real)*: Zero tolerance on print values ↩

Range: [0, ∞]

Default: 1e-05

penalty *(real)*: Minimum absolute penalty variable coefficient ↩

Range: [0, ∞]

Default: auto

pivotTol *(real)*: Simplex: Zero tolerance for matrix elements ↩

On each iteration, the simplex method seeks a nonzero matrix element to pivot on. Any element with absolute value less than pivotTol is treated as zero for this purpose.

Range: [0, ∞]

Default: 1e-09

ppFactor *(real)*: Partial pricing candidate list sizing parameter ↩

Range: [0, ∞]

Default: 1

preAnalyticCenter *(integer)*: Determines if analytic centers should be computed and used for variable fixing and the generation of alternative reduced costs (-1: Auto 0: Off, 1: Fixing, 2: Redcost, 3: Both) ↩

Default: auto

value	meaning
-1	Automatic.
0	Disable analytic center presolving.
1	Use analytic center for variable fixing only.
2	Use analytic center for reduced cost computation only.
3	Use analytic centers for both, variable fixing and reduced cost computation.

preBasisRed *(integer)*: Determines if a lattice basis reduction algorithm should be attempted as part of presolve ↩

Default: 0

value	meaning
-1	Automatic.
0	Disable basis reduction.
1	Enable basis reduction.

preBndRedCone *(integer)*: Determines if second order cone constraints should be used for inferring bound reductions on variables when solving a MIP ↩

Default: auto

value	meaning
-1	Automatic.
0	Disable bound reductions from second order cone constraints.
1	Enable bound reductions from second order cone constraints.

preBndRedQuad *(integer)*: Determines if convex quadratic contraints should be used for inferring

bound reductions on variables when solving a MIP ↩

Default: `auto`

value	meaning
`-1`	Automatic.
`0`	Disable bound reductions from quadratic constraints.
`1`	Enable bound reductions from quadratic constraints.

preCliqueStrategy *(integer)*: Determines how much effort to spend on clique covers in presolve ↩

Range: {`-1`, ..., ∞}

Default: `-1`

preCoefElim *(integer)*: Presolve: Specifies whether the optimizer should attempt to recombine constraints in order to reduce the number of non zero coefficients when presolving a mixed integer problem ↩

Default: `2`

value	meaning
`0`	Disabled.
`1`	Remove as many coefficients as possible.
`2`	Cautious eliminations. Will not perform a reduction if it might destroy problem structure useful to e.g. heuristics or cutting.

preComponents *(integer)*: Presolve: Determines whether small independent components should be detected and solved as individual subproblems during root node processing ↩

Default: `auto`

value	meaning
`-1`	Automatically determined.
`0`	Disable detection of independent components.
`1`	Enable detection of independent components.

preComponentsEffort *(real)*: Presolve: Adjusts the overall effort for the independent component presolver ↩

This control affects working limits for the subproblem solving as well as thresholds when it is called. Increase to put more emphasis on component presolving.

Range: [`0`, ∞]

Default: `1`

preConeDecomp *(integer)*: Presolve: Decompose regular and rotated cones with more than two elements and apply Outer Approximation on the resulting components ↩

Default: `auto`

value	meaning
-1	Automatically determined.
0	Disable cone decomposition.
1	Enable cone decomposition by replacing large cones with small ones in the presolved problem.
2	Similar to 1, plus decomposition is enabled even if the cone variable is fixed.
3	Cones are decomposed within the Outer Approximation domain only, i.e., the problem maintains the original cones.

preConfiguration *(integer)*: MIP Presolve: Determines whether binary rows with only few repeating coefficients should be reformulated ↩

The reformulation enumerates the extremal feasible configurations of a row and introduces new columns and rows to model the choice between these extremal configurations. This presolve operation can be disabled as part of the (advanced) IP reductions presolveOps.

Default: `auto`

value	meaning
-1	Automatically determined.
0	Disable configuration presolving.

preConvertSeparable *(integer)*: Presolve: Reformulate problem with non-diagonal quadratic objective and/or constraints as diagonal quadratic or second-order conic constraints ↩

This control is only used in MIQPs and MIQCQPs, and has no effect when used on continuous quadratic problems.

Default: `auto`

value	meaning
-1	Automatically determined.
0	Disable reformulation.
1	Enable reformulation to diagonal quadratic constraints.
2	Similar to 1, plus reduction to second-order cones.
3	Similar to 2, plus the objective function is converted to a constraint and treated as a quadratic constraint.

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

Only binary columns will be checked.

Default: `auto`

value	meaning
-1	Automatically determined.
0	Disabled.
1	Cautious strategy.
2	All candidate binaries will be checked for domination.

preDomRow *(integer)*: Presolve: Determines the level of dominated row removal reductions to perform when presolving a problem ↩

Default: `auto`

value	meaning
`-1`	Automatically determined.
`0`	Disabled.
`1`	Cautious strategy.
`2`	Medium strategy.
`3`	Aggressive strategy. All candidate row combinations will be considered.

preDupRow *(integer)*: Presolve: Determines the type of duplicate rows to look for and eliminate when presolving a problem ↩

Duplicate rows can also be disabled by clearing the corresponding bit of the presolveOps integer control.

Default: `auto`

value	meaning
`-1`	Automatically determined.
`0`	Do not eliminate duplicate rows.
`1`	Eliminate only rows that are identical in all variables.
`2`	Same as option 1 plus eliminate duplicate rows with simple penalty variable expressions. (MIP only).
`3`	Same as option 2 plus eliminate duplicate rows with more complex penalty variable expressions. (MIP only).

preElimQuad *(integer)*: Presolve: Allows for elimination of quadratic variables via doubleton rows ↩

Default: `auto`

value	meaning
`-1`	Automatically determined.
`0`	Do not eliminate duplicate rows.
`1`	Eliminate at least one quadratic variable for each doubleton row.

preFolding *(integer)*: Presolve: Determines if a folding procedure should be used to aggregate continuous columns in an equitable partition ↩

Default: `auto`

value	meaning
`-1`	Automatically determined.
`0`	Disabled.
`1`	Enabled.

preImplications *(integer)*: Presolve: Determines whether to use implication structures to remove redundant rows ↩

If implication sequences are detected, they might also be used in probing.

Default: `auto`

value	meaning
`-1`	Automatically determined.
`0`	Do not use implications for sparsification.
`1`	Use implications to remove reduandant rows.

preLinDep *(integer)*: Presolve: Determines whether to check for and remove linearly dependent equality constraints when presolving a problem ↩

Default: `auto`

value	meaning
`-1`	Automatically determined.
`0`	Do not check for linearly dependent equality constraints.
`1`	Check for and remove linearly dependent equality constraints.

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

This reduction applies to MIPs only.

Default: `1`

value	meaning
`0`	Disable check and reductions.
`1`	Enable check and reductions.

preProbing *(integer)*: Presolve: Amount of probing to perform on binary variables during presolve ↩

This is done by fixing a binary to each of its values in turn and analyzing the implications.

Default: `auto`

value	meaning
`-1`	Let the optimizer decide on the amount of probing.
`0`	Disabled.
`1`	Light probing — only few implications will be examined.
`2`	Full probing — all implications for all binaries will be examined.
`3`	Full probing and repeat as long as the problem is significantly reduced.

presolve *(integer)*: Determines whether presolving should be performed prior to starting the main algorithm ↩

Presolve attempts to simplify the problem by detecting and removing redundant constraints, tightening variable bounds, etc. In some cases, infeasibility may even be determined at this stage, or the optimal solution found.

Memory for presolve is dynamically resized. If the Optimizer runs out of memory for presolve, an error message (245) is produced. Presolve settings 2 and 3 can sometimes make the barrier solves more efficient.

Default: `1`

value	meaning
`-1`	Presolve applied, but a problem will not be declared infeasible if primal infeasibilities are detected. The problem will be solved by the LP optimization algorithm, returning an infeasible solution, which can sometimes be helpful.
`0`	Presolve not applied.
`1`	Presolve applied.
`2`	Presolve applied, but redundant bounds are not removed. This can sometimes increase the efficiency of the barrier algorithm.
`3`	Presolve is applied, and bounds detected to be redundant are always removed.

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

Range: [`0`, ∞]

Default: `0.1`

presolveOps *(integer)*: Specifies the operations which are performed during the presolve ↩

Setting single boolean options will overwrite the single bits of this bit map option.

Default: `511`

value	meaning
`bit 0 = 1`	Equivalent to presolveOps_singletonColRemoval.
`bit 1 = 2`	Equivalent to presolveOps_singletonRowRemoval.
`bit 2 = 4`	Equivalent to presolveOps_forcingRowRemoval.
`bit 3 = 8`	Equivalent to presolveOps_dualReductions.
`bit 4 = 16`	Equivalent to presolveOps_redundantRowRemoval.
`bit 5 = 32`	Equivalent to presolveOps_duplicateColRemoval.
`bit 6 = 64`	Equivalent to presolveOps_duplicateRowRemoval.
`bit 7 = 128`	Equivalent to presolveOps_strongDualReductions.
`bit 8 = 256`	Equivalent to presolveOps_variableEliminations.
`bit 9 = 512`	Equivalent to presolveOps_noIpReductions.
`bit 10 = 1024`	Equivalent to presolveOps_noGlobalDomainChange.
`bit 11 = 2048`	Equivalent to presolveOps_noAdvIpReductions.
`bit 12 = 4096`	Equivalent to presolveOps_noIntVarEliminations.
`bit 14 = 16384`	Equivalent to presolveOps_linDependRowRemoval.
`bit 15 = 32768`	Equivalent to presolveOps_noIntVarAndSosDetect.

presolveOps_dualReductions *(boolean)*: Dual reductions ↩

See also presolveOps.

Default: `1`

presolveOps_duplicateColRemoval *(boolean)*: Duplicate column removal ↩

See also presolveOps.

Default: `1`

presolveOps_duplicateRowRemoval *(boolean)*: Duplicate row removal ↩

See also presolveOps.

Default: `1`

presolveOps_forcingRowRemoval *(boolean)*: Forcing row removal ↩

See also presolveOps.

Default: `1`

presolveOps_linDependRowRemoval *(boolean)*: Linearly dependant row removal ↩

See also presolveOps.

Default: `0`

presolveOps_noAdvIpReductions *(boolean)*: No advanced IP reductions ↩

See also presolveOps.

Default: `0`

presolveOps_noGlobalDomainChange *(boolean)*: No semi-continuous variable detection ↩

See also presolveOps.

Default: `0`

presolveOps_noIntVarAndSosDetect *(boolean)*: No integer variable and SOS detection ↩

See also presolveOps.

Default: `0`

presolveOps_noIntVarEliminations *(boolean)*: No eliminations on integers ↩

See also presolveOps.

Default: `0`

presolveOps_noIpReductions *(boolean)*: No IP reductions ↩

See also presolveOps.

Default: `0`

presolveOps_redundantRowRemoval *(boolean)*: Redundant row removal ↩

See also presolveOps.

Default: `1`

presolveOps_singletonColRemoval *(boolean)*: Singleton column removal ↩

See also presolveOps.

Default: `1`

presolveOps_singletonRowRemoval *(boolean)*: Singleton row removal ↩

See also presolveOps.

Default: `1`

presolveOps_strongDualReductions *(boolean)*: Strong dual reductions ↩

See also presolveOps.

Default: `1`

presolveOps_variableEliminations *(boolean)*: Variable eliminations ↩

See also presolveOps.

Default: `1`

presolvePasses *(integer)*: Number of reduction rounds to be performed in presolve ↩

Range: {`0`, ..., ∞}

Default: `1`

pricingAlg *(integer)*: Simplex: Determines the primal simplex pricing method ↩

It is used to select which variable enters the basis on each iteration. In general Devex pricing requires more time on each iteration, but may reduce the total number of iterations, whereas partial pricing saves time on each iteration, but may result in more iterations.

Default: `auto`

value	meaning
`-1`	Partial pricing.
`0`	Determined automatically.
`1`	Devex pricing.
`2`	Steepest edge.
`3`	Steepest edge with unit initial weights.

primalOps *(integer)*: Primal simplex: Allows fine tuning the variable selection in the primal simplex solver ↩

If both bits 0 and 1 are both set or unset then the dj scaling strategy is determined automatically.

Default: `-1`

value	meaning
`bit 0 = 1`	Use aggressive dj scaling.

value	meaning
bit 1 = 2	Conventional dj scaling.
bit 2 = 4	Use reluctant switching back to partial pricing.
bit 3 = 8	Use dynamic switching between cheap and expensive pricing strategies.
bit 4 = 16	Keep solving even after potential cycling is detected.

primalPerturb *(real)*: Factor by which the problem will be perturbed prior to optimization by primal simplex ↩

A value of 0.0 results in no perturbation prior to optimization. Note the interconnection to the autoPerturb control. If autoPerturb is set to 1, the decision whether to perturb or not is left to the Optimizer. When the problem is automatically perturbed in primal simplex, however, the value of primalPerturb will be used for perturbation.

Range: [-∞, ∞]

Default: auto

primalUnshift *(boolean)*: Determines whether primal is allowed to call dual to unshift ↩

Default: 0

value	meaning
0	Allow the dual algorithm to be used to unshift.
1	Don't allow the dual algorithm to be used to unshift.

pseudoCost *(real)*: Branch and Bound: Default pseudo cost used in estimation of the degradation associated with an unexplored node in the tree search ↩

A pseudo cost is associated with each integer decision variable and is an estimate of the amount by which the objective function will be worse if that variable is forced to an integral value.

Range: [0, ∞]

Default: 0.01

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

This control only has an effect for problems with quadratic or second order cone constraints, and only if outer approximation has not been disabled by setting miqcpAlg to 0.

Range: {-1, ..., ∞}

Default: auto

qcRootAlg *(integer)*: Determines which algorithm is to be used to solve the root of a mixed integer quadratic constrained or mixed integer second order cone problem, when outer approximation is used ↩

This control only has an effect if miqcpAlg is set to 1.

Default: auto

value	meaning
-1	Determined automatically.
0	Use the barrier algorithm.
1	Use the dual simplex on a relaxation of the problem constructed using outer approximation.

qextractalg *(integer)*: quadratic extraction algorithm in GAMS interface ↩

Default: 0

value	meaning
0	Automatic
1	ThreePass: Uses a three-pass forward / backward / forward AD technique to compute function / gradient / Hessian values and a hybrid scheme for storage.
2	DoubleForward: Uses forward-mode AD to compute and store function, gradient, and Hessian values at each node or stack level as required. The gradients and Hessians are stored in linked lists.
3	Concurrent: Uses ThreePass and DoubleForward in parallel. As soon as one finishes, the other one stops.

qSimplexOps *(integer)*: Controls the behavior of the quadratic simplex solvers ↩

Default: 0

value	meaning
bit 0 = 1	Force traditional primal first phase.
bit 1 = 2	Force BigM primal first phase.
bit 2 = 4	Force traditional dual first phase.
bit 3 = 8	Force BigM dual first phase.
bit 4 = 16	Always use artificial bounds in dual.
bit 5 = 32	Use original problem basis only when warmstarting the KKT.
bit 6 = 64	Skip the primal bound flips for ranged primals (might cause more trouble than good if the bounds are very large).
bit 7 = 128	Also do the single pivot crash.
bit 8 = 256	Do not apply aggressive perturbation in dual.

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

Default: auto

value	meaning
-1	Determined automatically.
0	No purification step.
1	Always do the purification step.

randomSeed *(integer)*: Sets the initial seed to use for the pseudo-random number generator in the Optimizer ↩

The sequence of random numbers is always reset using the seed when starting a new optimization run.

Range: {-∞, ..., ∞}

Default: 1

refineOps *(integer)*: Specifies when the solution refiner should be executed to reduce solution infeasibilities ↩

The refiner will attempt to satisfy the target tolerances for all original linear constraints before presolve or scaling has been applied.

If neither the 7th nor 8th bit is set, the refiner will use the primal simplex if the primal violations are larger than the dual violations, otherwise it will use the dual simplex. If both the 7th and 8th bit are set then the refiner will split the problem into a primal feasible and dual feasible part, and solve the first with primal simplex and the second with dual simplex.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 19

value	meaning
bit 0 = 1	Equivalent to refineOps_lpOptimal.
bit 1 = 2	Equivalent to refineOps_mipSolution.
bit 3 = 8	Equivalent to refineOps_mipNodeLp.
bit 4 = 16	Equivalent to refineOps_lpPresolve.
bit 5 = 32	Equivalent to refineOps_iterativeRefiner.
bit 6 = 64	Equivalent to refineOps_refinerPrecision.
bit 7 = 128	Equivalent to refineOps_refinerUsePrimal.
bit 8 = 256	Equivalent to refineOps_refinerUseDual.
bit 9 = 512	Equivalent to refineOps_mipFixGlobals.
bit 10 = 1024	Equivalent to refineOps_mipFixGlobalsTarget.

refineOps_iterativeRefiner *(boolean)*: Apply the iterative refiner to refine the solution ↩

See also refineOps.

Default: 0

refineOps_lpOptimal *(boolean)*: Run the solution refiner on an optimal solution of a continuous problem ↩

See also refineOps.

Default: 1

refineOps_lpPresolve *(boolean)*: Run the solution refiner on an optimal solution before postsolve on a continuous problem ↩

See also refineOps.

Default: 1

refineOps_mipFixGlobals *(boolean)*: Refine MIP solutions such that rounding them keeps the problem feasible when reoptimized ↩

See also refineOps.

Default: `0`

refineOps_mipFixGlobalsTarget *(boolean)*: Attempt to refine MIP solutions such that rounding them keeps the problem feasible when reoptimized, but accept integers solutions even if refinement fails ↩

See also refineOps.

Default: `0`

refineOps_mipNodeLp *(boolean)*: Run the solution refiner on each node of the MIP search ↩

See also refineOps.

Default: `0`

refineOps_mipSolution *(boolean)*: Run the solution refiner when a new solution is found during a tree search ↩

The refiner will be applied to the presolved solution before any post-solve operations are applied.

See also refineOps.

Default: `1`

refineOps_refinerPrecision *(boolean)*: Use higher precision in the iterative refinement ↩

See also refineOps.

Default: `0`

refineOps_refinerUseDual *(boolean)*: If set, the iterative refiner will use the dual simplex algorithm ↩

See also refineOps.

Default: `0`

refineOps_refinerUsePrimal *(boolean)*: If set, the iterative refiner will use the primal simplex algorithm ↩

See also refineOps.

Default: `0`

reform *(boolean)*: Substitute out objective var and equ when possible ↩

Default: `1`

relaxTreeMemoryLimit *(real)*: When the memory used by the branch and bound search tree exceeds the target specified by the treeMemoryLimit control, the optimizer will try to reduce this by writing nodes to the tree file ↩

In rare cases, usually where the solve has many millions of very small nodes, the tree structural data (which cannot be written to the tree file) will grow large enough to approach or exceed the tree's memory target. When this happens, optimizer performance can degrade greatly as the solver makes heavy use of the tree file in preference to memory. To prevent this, the solver will automatically relax the tree memory limit when it detects this case; the relaxTreeMemoryLimit control specifies the proportion of the previous memory limit by which to relax it. Set relaxTreeMemoryLimit to 0.0 to force the Xpress Optimizer to never relax the tree memory limit in this way.

While setting higher values of relaxTreeMemoryLimit can improve performance significantly for a small number of models in low memory situations, the user is advised to use the treeMemoryLimit control to tune the memory usage of the branch and bound tree, according to the solve characteristics of their problem, rather than increasing relaxTreeMemoryLimit.

Range: [0, ∞]

Default: 0.1

relPivotTol *(real)*: Simplex: Minimum size of pivot element relative to largest element in column ↩

At each iteration a pivot element is chosen within a given column of the matrix. The relative pivot tolerance, relPivotTol, is the size of the element chosen relative to the largest possible pivot element in the same column.

Range: [0, ∞]

Default: 1e-06

repairIndefInitEq *(boolean)*: Controls if the optimizer should make indefinite quadratic matrices positive definite when it is possible ↩

Default: 1

value	meaning
0	Repair if possible.
1	Do not repair.

reRun *(boolean)*: Rerun with primal simplex when not optimal/feasible ↩

Applies only in cases where presolve is turned on and the model is diagnosed as infeasible or unbounded. If rerun is nonzero, we rerun the model using primal simplex with presolve turned off in hopes of getting better diagnostic information. If rerun is zero, no good diagnostic information exists, so we return no solution, only an indication of unboundedness/infeasibility.

Default: 0

reslim *(real)*: Overrides GAMS reslim option ↩

Sets the resource limit. When the solver has used more than this amount of CPU time (in seconds) the system will stop the search and report the best solution found so far. timeLimit and solTimeLimit overwrite this option.

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

Available memory (or container limits) are automatically detected but can also be changed by maxMemorySoft and maxMemoryHard.

Default: 0

value	meaning
1	Allow the optimizer to change the solve path if necessary to preserve memory when getting close to one of the memory limits.

rootPresolve *(integer)*: Determines if presolving should be performed on the problem after the tree search has finished with root cutting and heuristics ↩

Default: auto

value	meaning
-1	Let the optimizer decide if the problem should be presolved again.
0	Disabled.
1	Always presolve the root problem.

sbBest *(integer)*: Number of infeasible MIP entities to initialize pseudo costs for on each node ↩

If sbBest is set to zero, the control historyCosts will also be treated as zero and no past branching or strong branching information will be used in the MIP entity selection.

Default: auto

value	meaning
-1	Determined automatically.
0	Disable strong branching.
n>0	Perform strong branching on up to n entities at each node.

sbEffort *(real)*: Adjusts the overall amount of effort when using strong branching to select an infeasible MIP entity to branch on ↩

sbEffort is used as a multiplier on other strong branching related controls, and affects the values used for sbBest, sbSelect and sbIterLimit when those are set to automatic.

Range: [0, ∞]

Default: 1

sbEstimate *(integer)*: Branch and Bound: How to calculate pseudo costs from the local node when selecting an infeasible MIP entity to branch on ↩

These pseudo costs are used in combination with local strong branching and history costs to select the branch candidate.

Default: auto

value	meaning
-1	Automatically determined.
1-6	Different variants of local pseudo costs.

sbIterLimit *(integer)*: Number of dual iterations to perform the strong branching for each entity ↩

This control can be useful to increase or decrease the amount of effort (and thus time) spent performing strong branching at each node. Setting sbIterLimit=0 will disable dual strong

branch iterations. Instead, the entity at the head of the candidate list will be selected for branching.

Range: {-1, ..., ∞}

Default: auto

sbSelect *(integer)*: Size of the candidate list of MIP entities for strong branching ↩

Before strong branching is applied on a node of the branch and bound tree, a list of candidates is selected among the infeasible MIP entities. These entities are then evaluated based on the local LP solution and prioritized. Strong branching will then be applied to the sbBest candidates. The evaluation is potentially expensive and for some problems it might improve performance if the size of the candidate list is reduced.

Default: -2

value	meaning
-2	Automatic (low effort).
-1	Automatic (high effort).
n>=0	Include n entities in the candidate list (but always at least sbBest candidates).

scaling *(integer)*: Determines how the Optimizer will rescale a model internally before optimization ↩

If set to 0, no scaling will take place.

Setting scaling to 0 will preserve the current scaling of the problem. Note that the Optimizer might automatically select a different scaling strategy, when the control autoScaling is not disabled. However, if scaling is set to any value by the user, autoScaling will be ignored.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 163

value	meaning
bit 0 = 1	Equivalent to scaling_rowScaling.
bit 1 = 2	Equivalent to scaling_colScaling.
bit 2 = 4	Equivalent to scaling_rowScalingAgain.
bit 3 = 8	Equivalent to scaling_maximum.
bit 4 = 16	Equivalent to scaling_curtisReid.
bit 5 = 32	Equivalent to scaling_byMaxElemNotGeoMean.
bit 6 = 64	Equivalent to scaling_bigM.
bit 7 = 128	Equivalent to scaling_simplexObjScaling.
bit 8 = 256	Equivalent to scaling_ignoreQuadRowPart.
bit 9 = 512	Equivalent to scaling_beforePresolve.
bit 10 = 1024	Equivalent to scaling_noScalingRowsUp.
bit 11 = 2048	Equivalent to scaling_noScalingColsDown.
bit 12 = 4096	Equivalent to scaling_disableGlobalObjScaling.
bit 13 = 8192	Equivalent to scaling_rhsScaling.
bit 14 = 16384	Equivalent to scaling_noAggressiveQScaling.
bit 15 = 32768	Equivalent to scaling_slackScaling.

scaling_beforePresolve *(boolean)*: Scale before presolve ↩

See also scaling.

Default: `0`

scaling_bigM *(boolean)*: Treat big-M rows as normal rows ↩

See also scaling.

Default: `0`

scaling_byMaxElemNotGeoMean *(boolean)*: 0: scale by geometric mean ↩

1: scale by maximum element.

See also scaling.

Default: `1`

scaling_colScaling *(boolean)*: Column scaling ↩

See also scaling.

Default: `1`

scaling_curtisReid *(boolean)*: Curtis-Reid ↩

See also scaling.

Default: `0`

scaling_disableGlobalObjScaling *(boolean)*: Do not apply automatic objective scaling ↩

See also scaling.

Default: `0`

scaling_ignoreQuadRowPart *(boolean)*: Exclude the quadratic part of constraint when calculating scaling factors ↩

See also scaling.

Default: `0`

scaling_maximum *(boolean)*: Maximum ↩

See also scaling.

Default: `0`

scaling_noAggressiveQScaling *(boolean)*: Disable aggressive quadratic scaling ↩

See also scaling.

Default: `0`

scaling_noScalingColsDown *(boolean)*: Do not scale columns down ↩

See also scaling.

Default: 0

scaling_noScalingRowsUp *(boolean)*: Do not scale rows up ↩

See also scaling.

Default: 0

scaling_rhsScaling *(boolean)*: RHS scaling ↩

See also scaling.

Default: 0

scaling_rowScaling *(boolean)*: Row scaling ↩

See also scaling.

Default: 1

scaling_rowScalingAgain *(boolean)*: Row scaling again ↩

See also scaling.

Default: 0

scaling_simplexObjScaling *(boolean)*: Scale objective function for the simplex method ↩

See also scaling.

Default: 1

scaling_slackScaling *(boolean)*: Enable explicit linear slack scaling ↩

See also scaling.

Default: 0

sifting *(integer)*: Determines whether to enable sifting algorithm with the dual simplex method ↩

Default: auto

value	meaning
-1	Automatically determined.
0	Disable sifting.
1	Enable sifting.

siftPasses *(integer)*: Determines how quickly we allow to grow the worker problems during the sifting algorithm ↩

Using larger values can increase the number of columns added to the worker problem which often results in increased solve times for the worker problems but the number of necessary sifting iterations may be reduced.

Range: {0, ..., ∞}

Default: 4

siftPresolveOps *(integer)*: Determines the presolve operations for solving the subproblems during the sifting algorithm ↩

Default: -1

value	meaning
-1	Use the presolveOps setting specified for the original problem.
>=0	Use the value for the presolveOps parameter for solving the subproblems during the sifting algorithm.

siftSwitch *(integer)*: Determines which algorithm to use for solving the subproblems during sifting ↩

Default: -1

value	meaning
-1	Dual simplex.
0	Barrier.
>0	Use the barrier algorithm while the number of dual infeasibilities is larger than this value, otherwise use dual simplex.

solnpool *(string)*: Solution pool file name ↩

If set, the integer feasible solutions generated during the global search will be saved to a solution pool. A GDX file whose name is given by this option will be created and will contain an index to separate GDX files containing the individual solutions in the solution pool.

Default: none

solnpoolCapacity *(integer)*: Limit on number of solutions to store ↩

Range: {1, ..., ∞}

Default: 999999999

solnpoolCullDiversity *(integer)*: Cull N solutions based on solution diversity ↩

When performing a round of culls due to a full solution pool, this control sets the maximum number to cull based on the diversity of the solutions in the pool.

Range: {-1, ..., ∞}

Default: -1

solnpoolCullObj *(integer)*: Cull N solutions based on objective values ↩

When performing a round of culls due to a full solution pool, this control sets the maximum number to cull based on the MIP objective function.

Range: {-1, ..., ∞}

Default: -1

solnpoolCullRounds *(integer)*: Terminate solution generation after N culling rounds ↩

Limits the rounds of culls performed due to a full solution pool.

Default: 999999999

solnpoolDupPolicy *(integer)*: Policy to use when handling storage of duplicate solutions ↩

Default: 0

value	meaning
0	Keep all: All solutions are kept including duplicates.
1	Continuous: All variables are compared with an exact match. Duplicate solutions are discarded.
2	Discrete and continuous separate: Both the discrete component of a solution pair and the continuous solution variables are compared. The continuous variables are compared with an exact match. Duplicate solutions are discarded.
3	Discrete only: Only the discrete component of a solution pair is compared. Duplicate solutions are discarded.

solnpoolmerge *(string)*: Solution pool file name for merged solutions ↩

Default: `none`

solnpoolnumsym *(integer)*: Maximum number of variable symbols when writing merged solutions ↩

Range: {1, ..., ∞}

Default: `10`

solnpoolPop *(integer)*: Controls method used to populate the solution pool ↩

By default the MIP solution pool merely stores the incumbent solutions that are found during the global search, without changing the behavior of the search itself. In constrast, the MIP solution enumerator makes it possible to enumerate all or many of the feasible solutions for the MIP, instead of searching for the best solution.

Default: `1`

value	meaning
1	generate solutions using the normal search algorithm
2	invoke the solution enumerator to generate solutions

solnpoolPrefix *(string)*: File name prefix for GDX solution files ↩

Default: `soln`

solnpoolVerbosity *(integer)*: Controls verbosity of solution pool routines ↩

Default: `0`

value	meaning
-1	no output
0	output only messages coming from the XPRESS libraries
1	add some messages logging the effect of solution pool options
2	debugging mode

solTimeLimit *(real)*: Maximum time in seconds that the Optimizer will run a MIP solve before it terminates, given that a solution has been found ↩

As long as no solution has been found, this control will have no effect.

This control has been newly introduced with Xpress Optimizer version 9.0 and should be used instead of the deprecated MAXTIME control. It can be combined with the timeLimit control.

Default: `1e+20`

value	meaning
>0	If an integer solution has been found, stop MIP search after the given number of seconds, otherwise continue until an integer solution is finally found.

sosRefTol *(real)*: Minimum relative gap between the ordering values of elements in a special ordered set ↩

The gap divided by the absolute value of the larger of the two adjacent values must be at least sosRefTol.

This tolerance must not be set lower than 1.0E-06.

Range: [0, ∞]

Default: `1e-06`

symmetry *(integer)*: Adjusts the overall amount of effort for symmetry detection ↩

Default: `1`

value	meaning
0	No symmetry detection.
1	Conservative effort.
2	Intensive symmetry search.

symSelect *(integer)*: Adjusts the overall amount of effort for symmetry detection ↩

Default: `-1`

value	meaning
0	Search the whole matrix (otherwise the 0, 1 and -1 coefficients only).
1	Search all entities (otherwise binaries only).

threads *(integer)*: Default number of threads used during optimization ↩

Controls the number of threads to use. Positive values will be compared to the number of available cores detected and reduced if greater than this amount. Non- positive values are interpreted as the number of cores to leave free so setting threads to 0 uses all available cores while setting threads to -1 leaves one core free for other tasks.

Range: {-∞, ..., ∞}

Default: `1`

timeLimit *(real)*: Maximum time in seconds that the Optimizer will run before it terminates, including the problem setup time and solution time ↩

For MIP problems, this is the total time taken to solve all nodes.

This control has been newly introduced with Xpress Optimizer version 9.0 and should be used instead of the deprecated MAXTIME control. Note that the meaning of positive values differs between timeLimit and MAXTIME. When both controls are set, timeLimit takes precedence. The functionality of positive MAXTIME values is covered by solTimeLimit.

Default: `1e+20`

value	meaning
>0	Stop LP or MIP search after the given number of seconds.

trace *(integer)*: Display the infeasibility diagnosis during presolve ↩

If non-zero, an explanation of the logical deductions made by presolve to deduce infeasibility or unboundedness will be displayed on screen or sent to the message callback function.

Presolve is sometimes able to detect infeasibility and unboundedness in problems. The set of deductions made by presolve can allow the user to diagnose the cause of infeasibility or unboundedness in their problem. However, not all infeasibility or unboundedness can be detected and diagnosed in this way.

Range: {0, ..., ∞}

Default: `0`

treeCompression *(integer)*: When writing nodes to the global file, the optimizer can try to use data-compression techniques to reduce the size of the tree file on disk ↩

The treeCompression control determines the strength of the data-compression algorithm used; higher values give superior data- compression at the affect of decreasing performance, while lower values compress quicker but not as effectively. Where treeCompression is set to 0, no data compression will be used on the tree file.

Range: {0, ..., ∞}

Default: `2`

treeCoverCuts *(integer)*: Branch and Bound: Number of rounds of lifted cover inequalities generated at nodes other than the top node in the tree ↩

Compare with the description for coverCuts. A value of -1 indicates the number of rounds is determined automatically.

Range: {-1, ..., ∞}

Default: `auto`

treeCutSelect *(integer)*: Bit vector providing detailed control of the cuts created during the tree search of a MIP solve ↩

Use cutSelect to control cuts on the root node.

The default value is -1 which enables all bits. Any bits not listed in the above table should be left in their default 'on' state, since the interpretation of such bits might change in future versions of the optimizer.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: `-1`

value	meaning
bit 5 = 32	Equivalent to treeCutSelect_clique.
bit 6 = 64	Equivalent to treeCutSelect_mir.
bit 7 = 128	Equivalent to treeCutSelect_cover.
bit 8 = 256	Equivalent to treeCutSelect_mirRowAggregation.
bit 11 = 2048	Equivalent to treeCutSelect_flowpath.
bit 12 = 4096	Equivalent to treeCutSelect_implication.
bit 13 = 8192	Equivalent to treeCutSelect_liftAndProject.
bit 14 = 16384	Equivalent to treeCutSelect_disableCutRows.
bit 15 = 32768	Equivalent to treeCutSelect_gubCover.
bit 16 = 65536	Equivalent to treeCutSelect_zeroHalf.
bit 17 = 131072	Equivalent to treeCutSelect_indicator.
bit 18 = 262144	Equivalent to treeCutSelect_gomory.
bit 20 = 1048576	Equivalent to treeCutSelect_farkas.

treeCutSelect_clique *(boolean)*: Clique cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_cover *(boolean)*: Lifted cover cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_disableCutRows *(boolean)*: Disable cutting from cut rows ↩

See also treeCutSelect.

Default: 1

treeCutSelect_farkas *(boolean)*: Farkas cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_flowpath *(boolean)*: Flow path cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_gomory *(boolean)*: Strong Chvatal-Gomory cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_gubCover *(boolean)*: Lifted GUB cover cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_implication *(boolean)*: Implication cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_indicator *(boolean)*: Indicator constraint cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_liftAndProject *(boolean)*: Turn on automatic Lift and Project cutting strategy ↩

See also treeCutSelect.

Default: 1

treeCutSelect_mir *(boolean)*: Mixed Integer Rounding (MIR) cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_mirRowAggregation *(boolean)*: Turn on row aggregation for MIR cuts ↩

See also treeCutSelect.

Default: 1

treeCutSelect_zeroHalf *(boolean)*: Zero-half cuts ↩

See also treeCutSelect.

Default: 1

treeGomCuts *(integer)*: Branch and Bound: Number of rounds of Gomory cuts generated at nodes other than the first node in the tree ↩

Compare with the description for gomCuts. A value of -1 indicates the number of rounds is determined automatically.

Range: {-1, ..., ∞}

Default: auto

treeMemoryLimit *(integer)*: Soft limit, in megabytes, for the amount of memory to use in storing the branch and bound search tree ↩

This doesn't include memory used for presolve, heuristics, solving the LP relaxation, etc. When set to 0 (the default), the optimizer will calculate a limit automatically based on the amount of free physical memory detected in the machine. When the memory used by the branch and bound tree exceeds this limit, the optimizer will try to reduce the memory usage by writing lower-rated sections of the tree to a file called the "tree file". Though the solve can continue if it cannot bring the tree memory usage below the specified limit, performance will be inhibited and a message will be printed to the log.

Range: {0, ..., ∞}

Default: auto

treeMemorySavingTarget *(real)*: When the memory used by the branch-and-bound search tree exceeds the limit specified by the treeMemoryLimit control, the optimizer will try to save memory by writing lower-rated sections of the tree to the tree file ↩

The target amount of memory to save will be enough to bring memory usage back below the limit, plus enough extra to give the tree room to grow. The treeMemorySavingTarget control specifies the extra proportion of the tree's size to try to save; for example, if the tree memory limit is 1000Mb and treeMemorySavingTarget is 0.1, when the tree size exceeds 1000Mb the optimizer will try to reduce the tree size to 900Mb. Reducing the value of treeMemorySavingTarget will cause less extra nodes of the tree to be written to the tree file, but will result in the memory saving routine being triggered more often (as the tree will have less room in which to grow), which can reduce performance. Increasing the value of treeMemorySavingTarget will cause additional, more highly-rated nodes, of the tree to be written to the tree file, which can cause performance issues if these nodes are required later in the solve.

Range: [0, ∞]

Default: 0.4

treeQCCuts *(integer)*: Branch and Bound: Limit on the number of rounds of outer approximation cuts generated for nodes other than the root node, when solving a mixed integer quadratic constrained or mixed integer second order conic problem with outer approximation ↩

This control only has an effect for problems with quadratic or second order cone constraints, and only if outer approximation has not been disabled by setting miqcpAlg to 0.

Range: {-1, ..., ∞}

Default: auto

varSelection *(integer)*: Branch and Bound: Determines the formula used to calculate the estimate of each integer variable, and thus which integer variable is selected to be branched on at a given node ↩

The variable selected to be branched on is the one with the maximum estimate.

Default: auto

value	meaning
-1	Determined automatically.
1	The minimum of the 'up' and 'down' pseudo costs.
2	The 'up' pseudo cost plus the 'down' pseudo cost.
3	The maximum of the 'up' and 'down' pseudo costs, plus twice the minimum of the 'up' and 'down' pseudo costs.
4	The maximum of the 'up' and 'down' pseudo costs.
5	The 'down' pseudo cost.
6	The 'up' pseudo cost.
7	A weighted combination of the 'up' and 'down' pseudo costs, where the weights depend on how fractional the variable is.
8	The product of the 'up' and 'down' pseudo costs.

writePrtSol *(boolean)*: Directs optimizer to output a "printsol" file ↩

Default: `0`

xslp_algorithm *(integer)*: Bit map describing the SLP algorithm(s) to be used ↩

Setting single boolean options will overwrite the single bits of this bit map option.

Default: `166`

value	meaning
`bit 0 = 1`	Equivalent to xslp_algorithm_noStepBounds.
`bit 1 = 2`	Equivalent to xslp_algorithm_stepBoundsAsRequired.
`bit 2 = 4`	Equivalent to xslp_algorithm_estimateStepBounds.
`bit 3 = 8`	Equivalent to xslp_algorithm_dynamicDamping.
`bit 4 = 16`	Equivalent to xslp_algorithm_holdValues.
`bit 5 = 32`	Equivalent to xslp_algorithm_retainPreviousValue.
`bit 6 = 64`	Equivalent to xslp_algorithm_resetDeltaZ.
`bit 7 = 128`	Equivalent to xslp_algorithm_quickConvergenceCheck.
`bit 8 = 256`	Equivalent to xslp_algorithm_escalatePenalties.
`bit 9 = 512`	Equivalent to xslp_algorithm_switchToPrimal.
`bit 11 = 2048`	Equivalent to xslp_algorithm_maxCostOption.
`bit 12 = 4096`	Equivalent to xslp_algorithm_residualErrors.
`bit 13 = 8192`	Equivalent to xslp_algorithm_noLPPolishing.
`bit 14 = 16384`	Equivalent to xslp_algorithm_cascadeBounds.
`bit 15 = 32768`	Equivalent to xslp_algorithm_clampExtendedActiveSB.
`bit 16 = 65536`	Equivalent to xslp_algorithm_clampExtendedAll.

xslp_algorithm_cascadeBounds *(boolean)*: Step bounds are updated to accomodate cascaded values (otherwise cascaded values are pushed to respect step bounds) ↩

Normally, cascading will respect the step bounds of the SLP variable being cascaded. However, allowing the cascaded value to fall outside the step bounds (i.e. expanding the step bounds) can lead to better linearizations, as cascading will set better values for the SLP variables regarding their determining rows; note, that this later strategy might interfere with convergence of the cascaded variables.

See also xslp_algorithm.

Default: `0`

xslp_algorithm_clampExtendedActiveSB *(boolean)*: Apply clamping when converged on extended criteria only with some variables having active step bounds ↩

When clamping is applied, then in any iteration when the solution would normally be deemed converged on extended criteria only, an extra step bound shrinking step is applied to help imposing strict convergence. In this variant, clamping is only applied on variables that have converged on extended criteria only and have active step bounds.

See also xslp_algorithm.

Default: `0`

xslp_algorithm_clampExtendedAll *(boolean)*: Apply clamping when converged on extended criteria only ↩

When clamping is applied, then in any iteration when the solution would normally be deemed converged on extended criteria only, an extra step bound shrinking step is applied to help imposing strict convergence. In this variant, clamping is applied on all variables that have converged on extended criteria only.

See also xslp_algorithm.

Default: `0`

xslp_algorithm_dynamicDamping *(boolean)*: Use dynamic damping ↩

Dynamic damping is sometimes an alternative to step bounding as a means of encouraging convergence, but it does not have the same power to force convergence as do step bounds.

See also xslp_algorithm.

Default: `0`

xslp_algorithm_escalatePenalties *(boolean)*: Escalate penalties ↩

Constraint penalties are increased after each SLP iteration where penalty vectors are present in the solution. Escalation applies an additional scaling factor to the penalty costs for active errors. This helps to prevent successive solutions becoming "stuck" because of a particular constraint, because its cost will be raised so that other constraints may become more attractive to violate instead and thus open up a new region to explore.

See also xslp_algorithm.

Default: `0`

xslp_algorithm_estimateStepBounds *(boolean)*: Estimate step bounds from early SLP iterations ↩

If initial step bounds are not being explicitly provided, this gives a good method of calculating reasonable values. Values will tend to be larger rather than smaller, to reduce the risk of infeasibility caused by excessive tightness of the step bounds.

See also xslp_algorithm.

Default: `1`

xslp_algorithm_holdValues *(boolean)*: Do not update values which are converged within strict tolerance ↩

Models which are numerically unstable may benefit from this setting, which does not update values which have effectively hardly changed. If a variable subsequently does move outside its strict convergence tolerance, it will be updated as usual.

See also xslp_algorithm.

Default: `0`

xslp_algorithm_maxCostOption *(boolean)*: Continue optimizing after penalty cost reaches maximum ↩

Normally if the penalty cost reaches its maximum (by default the value of infinity), the optimization will terminate with an unconverged solution. If the maximum value is set to a smaller value, then it may make sense to continue, using other means to determine when to stop.

See also xslp_algorithm.

Default: 0

xslp_algorithm_noLPPolishing *(boolean)*: Skip the solution polishing step if the LP postsolve returns a slightly infeasible, but claimed optimal solution ↩

Due to the nature of the SLP linearizations, and in particular because of the large differences in the objective function (model objective against penalty costs) some dual reductions in the linear presolver might introduce numerically instable reductions that cause slight infeasibilities to appear in postsolve. It is typically more efficient to remove these infeasibilities with an extra call to the linear optimizer; compared to switching these reductions off, which usually has a significant cost in performance. This bit is provided for numerically very hard problems, when the polishing step proves to be too expensive (Xpress SLP will report these if any in the final log summary).

See also xslp_algorithm.

Default: 0

xslp_algorithm_noStepBounds *(boolean)*: Do not apply step bounds ↩

The default algorithm uses step bounds to force convergence. Step bounds may not be appropriate if dynamic damping is used.

See also xslp_algorithm.

Default: 0

xslp_algorithm_quickConvergenceCheck *(boolean)*: Quick convergence check ↩

Normally, each variable is checked against all convergence criteria until either a criterion is found which it passes, or it is declared "not converged". Later (extended convergence) criteria are more expensive to test and, once an unconverged variable has been found, the overall convergence status of the solution has been established. The quick convergence check carries out checks on the strict criteria, but omits checks on the extended criteria when an unconverged variable has been found.

See also xslp_algorithm.

Default: 1

xslp_algorithm_resetDeltaZ *(boolean)*: Reset xslp_delta_z to zero when converged and continue SLP ↩

One of the mechanisms to avoid local optima is to retain small non-zero coefficients between delta vectors and constraints, even when the coefficient should strictly be zero. If this option is set, then a converged solution will be continued with zero coefficients as appropriate.

See also xslp_algorithm.

Default: 0

xslp_algorithm_residualErrors *(boolean)*: Accept a solution which has converged even if there are still significant active penalty error vectors ↩

Normally, the optimization will continue if there are active penalty vectors in the solution. However, it may be that there is no feasible solution (and so active penalties will always be present). Setting bit 12 means that, if other convergence criteria are met, then the solution will be accepted as converged and the optimization will stop.

See also xslp_algorithm.

Default: 0

xslp_algorithm_retainPreviousValue *(boolean)*: Retain previous value when cascading if determining row is zero ↩

If the determining row is zero (that is, all the coefficients interacting with it are either zero or in columns with a zero activity), then it is impossible to calculate a new value for the vector being cascaded. The choice is to use the solution value as it is, or to revert to the assumed value

See also xslp_algorithm.

Default: 1

xslp_algorithm_stepBoundsAsRequired *(boolean)*: Apply step bounds to SLP delta vectors only when required ↩

Step bounds can be applied to all vectors simultaneously, or applied only when oscillation of the delta vector (change in sign between successive SLP iterations) is detected.

See also xslp_algorithm.

Default: 1

xslp_algorithm_switchToPrimal *(boolean)*: Use the primal simplex algorithm when all error vectors become inactive ↩

The primal simplex algorithm often performs better than dual during the final stages of SLP optimization when there are relatively few basis changes between successive solutions. As it is impossible to establish in advance when the final stages are being reached, the disappearance of error vectors from the solution is used as a proxy.

See also xslp_algorithm.

Default: 0

xslp_analyze *(integer)*: Bit map activating additional options supporting model / solution path analyzis ↩

In most cases, the value of this control does not affect the solution process itself. However, bit 3 (extended summary) will cause SLP to do more function evaluations, and the presence of non-deterministic user functions might cause changes in the solution process. These options are off by default due to performance considerations.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0

value	meaning
bit 3 = 8	Equivalent to xslp_analyze_extendedFinalSummary.
bit 4 = 16	Equivalent to xslp_analyze_infeasibleIteration.
bit 6 = 64	Equivalent to xslp_analyze_saveLinearizations.
bit 7 = 128	Equivalent to xslp_analyze_saveIterBasis.
bit 8 = 256	Equivalent to xslp_analyze_saveFile.

xslp_analyze_extendedFinalSummary *(boolean)*: Include an extended iteration summary ↩

See also xslp_analyze.

Default: `0`

xslp_analyze_infeasibleIteration *(boolean)*: Run infeasibility analysis on infeasible iterations ↩

See also xslp_analyze.

Default: `0`

xslp_analyze_saveFile *(boolean)*: Create an Xpress SLP save file at every xslp_autosave iterations ↩

See also xslp_analyze.

Default: `0`

xslp_analyze_saveIterBasis *(boolean)*: Write the initial basis of the linearizations to disk at every xslp_autosave iterations ↩

See also xslp_analyze.

Default: `0`

xslp_analyze_saveLinearizations *(boolean)*: Write the linearizations to disk at every xslp_autosave iterations ↩

See also xslp_analyze.

Default: `0`

xslp_aTol_a *(real)*: Absolute delta convergence tolerance ↩

The absolute delta convergence criterion assesses the change in value of a variable (δX) against the absolute delta convergence tolerance. If $\delta X <$ xslp_aTol_a then the variable has converged on the absolute delta convergence criterion. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_aTol_r *(real)*: Relative delta convergence tolerance ↩

The relative delta convergence criterion assesses the change in value of a variable (δX) relative to the value of the variable (X), against the relative delta convergence tolerance. If $\delta X <$ $X *$ xslp_aTol_r then the variable has converged on the relative delta convergence criterion. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_augmentation *(integer)*: Bit map describing the SLP augmentation method(s) to be used ↩

Setting single boolean options will overwrite the single bits of this bit map option.

Default: `12`

value	meaning
`bit 0 = 1`	Equivalent to xslp_augmentation_minimum.
`bit 1 = 2`	Equivalent to xslp_augmentation_evenHanded.
`bit 2 = 4`	Equivalent to xslp_augmentation_equalityErrorVectors.
`bit 3 = 8`	Equivalent to xslp_augmentation_allErrorVectors.
`bit 4 = 16`	Equivalent to xslp_augmentation_penaltyDeltaVectors.
`bit 5 = 32`	Equivalent to xslp_augmentation_aMeanWeight.
`bit 6 = 64`	Equivalent to xslp_augmentation_sbFromValues.
`bit 7 = 128`	Equivalent to xslp_augmentation_sbFromAbsValues.
`bit 8 = 256`	Equivalent to xslp_augmentation_stepBoundRows.
`bit 9 = 512`	Equivalent to xslp_augmentation_allRowErrorVectors.
`bit 10 = 1024`	Equivalent to xslp_augmentation_noUpdateIfOnlyIV.

xslp_augmentation_allErrorVectors *(boolean)*: Penalty error vectors on all non-linear inequality constraints ↩

The linearization of a nonlinear constraint is inevitably an approximation and so may not be feasible except at the point of linearization. Adding penalty error vectors allows the linear approximation to be violated at a cost and so ensures that the linearized constraint is feasible.

See also xslp_augmentation.

Default: `1`

xslp_augmentation_allRowErrorVectors *(boolean)*: Penalty error vectors on all constraints ↩

If the linear portion of the underlying model may actually be infeasible, then applying penalty vectors to all rows may allow identification of the infeasibility and may also allow a useful solution to be found.

See also xslp_augmentation.

Default: `0`

xslp_augmentation_aMeanWeight *(boolean)*: Use arithmetic means to estimate penalty weights ↩

Penalty weights are estimated from the magnitude of the elements in the constraint or interacting rows. Geometric means are normally used, so that a few excessively large or small values do not distort the weights significantly. Arithmetic means will value the coefficients more equally.

See also xslp_augmentation.

Default: `0`

xslp_augmentation_equalityErrorVectors *(boolean)*: Penalty error vectors on all non-linear equality constraints ↩

The linearization of a nonlinear equality constraint is inevitably an approximation and so will not generally be feasible except at the point of linearization. Adding penalty error vectors allows the linear approximation to be violated at a cost and so ensures that the linearized constraint is feasible.

See also xslp_augmentation.

Default: `1`

xslp_augmentation_evenHanded *(boolean)*: Even handed augmentation ↩

Standard augmentation treats variables which appear in non-constant coefficients in a different way from those which contain non-constant coefficients. Even- handed augmentation treats them all in the same way by replacing each non- constant coefficient C in a vector V by a new coefficient C∗V in the "equals" column (which has a fixed activity of 1) and creating delta vectors for all types of variable in the same way.

See also xslp_augmentation.

Default: `0`

xslp_augmentation_minimum *(boolean)*: Minimum augmentation ↩

Standard augmentation includes delta vectors for all variables involved in nonlinear terms (in non-constant coefficients or as vectors containing non- constant coefficients). includes delta vectors only for variables in non- constant coefficients. This produces a smaller linearization, but there is less control on convergence, because convergence control (for example, step bounding) cannot be applied to variables without deltas.

See also xslp_augmentation.

Default: `0`

xslp_augmentation_noUpdateIfOnlyIV *(boolean)*: Intial values do not imply an SLP variable ↩

Having an initial value will not cause the augmentation to include the corresponding delta variable; i.e. treat the variable as an SLP variable. Useful to provide initial values necessary in the first linearization in case of a minimal augmentation, or as a convenience option when it's easiest to set an initial value for all variables for some reason.

See also xslp_augmentation.

Default: `0`

xslp_augmentation_penaltyDeltaVectors *(boolean)*: Penalty vectors to exceed step bounds ↩

Although it has rarely been found necessary or desirable in practice, Xpress-SLP allows step bounds to be violated at a cost. This may help with feasibility but it generally slows down or prevents convergence, so it should be used only if found absolutely necessary.

See also xslp_augmentation.

Default: `0`

xslp_augmentation_sbFromAbsValues *(boolean)*: Estimate step bounds from absolute values of row coefficients ↩

If step bounds are to be imposed from the start, the best approach is to provide explicit values for the bounds. Alternatively, Xpress-SLP can estimate the values from the largest estimated magnitude of the coefficients in the relevant rows.

See also xslp_augmentation.

Default: `0`

xslp_augmentation_sbFromValues *(boolean)*: Estimate step bounds from values of row coefficients ↩

If step bounds are to be imposed from the start, the best approach is to provide explicit values for the bounds. Alternatively, Xpress-SLP can estimate the values from the range of estimated coefficient sizes in the relevant rows.

See also xslp_augmentation.

Default: 0

xslp_augmentation_stepBoundRows *(boolean)*: Row-based step bounds ↩

Step bounds are normally applied as bounds on the delta variables. Some applications may find that using explicit rows to bound the delta vectors gives better results.

See also xslp_augmentation.

Default: 0

xslp_autosave *(integer)*: Frequency with which to save the model ↩

A value of zero means that the model will not automatically be saved. A positive value of n will save model information at every nth SLP iteration as requested by xslp_analyze.

Range: {0, ..., ∞}

Default: 0

xslp_barCrossOverStart *(integer)*: Default crossover activation behaviour for barrier start ↩

When xslp_barLimit is set, xslp_barCrossOverStart offers an overwrite control on when crossover is applied. A positive value indicates that crossover should be disabled in iterations smaller than xslp_barCrossOverStart and should be enabled afterwards, or when stalling is detected as described in xslp_barStartOps. A value of 0 indicates to respect the value of crossover and only overwrite its value when stalling is detected. A value of -1 indicates to always rely on the value of crossover.

Range: {0, ..., ∞}

Default: 0

xslp_barLimit *(integer)*: Number of initial SLP iterations using the barrier method ↩

Particularly for larger models, using the Newton barrier method is faster in the earlier SLP iterations. Later on, when the basis information becomes more useful, a simplex method generally performs better. xslp_barLimit sets the number of SLP iterations which will be performed using the Newton barrier method.

Range: {0, ..., ∞}

Default: 0

xslp_barStallingLimit *(integer)*: Number of iterations to allow numerical failures in barrier before switching to dual ↩

On large problems, it may be beneficial to warm start progress by running a number of iterations with the barrier solver as specified by xslp_barLimit. On some numerically difficult problems, the barrier may stop prematurely due to numerical issues. Such solves can sometimes be finished if crossover is applied. After xslp_barStallingLimit such attempts, SLP will automatically switch to use the dual simplex.

Range: {0, ..., ∞}

Default: 3

xslp_barStallingObjLimit *(integer)*: Number of iterations over which to measure the objective change for barrier iterations with no crossover ↩

On large problems, it may be beneficial to warm start progress by running a number of iterations with the barrier solver without crossover by setting xslp_barLimit to a positive value and setting crossover to 0. A potential drawback is slower convergence due to the interior point provided by the barrier solve keeping a higher number of variables active. This may lead to stalling in progress, negating the benefit of using the barrier. When in the last xslp_barStallingObjLimit iterations no significant progress has been made, crossover is automatically enabled.

Range: {0, ..., ∞}

Default: 3

xslp_barStallingTol *(real)*: Required change in the objective when progress is measured in barrier iterations without crossover ↩

Minumum objective variability change required in relation to control xslp_barStallingObjLimit for the iterations to be regarded as making progress. The net objective, error cost and error sum are taken into account.

Range: [0, ∞]

Default: 0.05

xslp_barStartOps *(integer)*: Controls behaviour when the barrier is used to solve the linearizations ↩

Setting single boolean options will overwrite the single bits of this bit map option.

Default: -1

value	meaning
bit 0 = 1	Equivalent to xslp_barStartOps_stallingObjective.
bit 1 = 2	Equivalent to xslp_barStartOps_stallingNumerical.
bit 2 = 4	Equivalent to xslp_barStartOps_allowInteriorSol.

xslp_barStartOps_allowInteriorSol *(boolean)*: If a non-vertex converged solution found by barrier without crossover can be returned as a final solution ↩

See also xslp_barStartOps.

Default: 1

xslp_barStartOps_stallingNumerical *(boolean)*: Fall back to dual simplex if too many numerical problems are reported by the barrier ↩

See also xslp_barStartOps.

Default: 1

xslp_barStartOps_stallingObjective *(boolean)*: Check objective progress when no crossover is applied ↩

See also xslp_barStartOps.

Default: 1

xslp_calcThreads *(integer)*: Number of threads used for formula and derivatives evaluations ↩

When beneficial, SLP can calculate formula values and partial derivative information in parallel.

Range: {-1, ..., 1}

Default: auto

xslp_cdTol_a *(real)*: Absolute tolerance for deducing constant derivatives ↩

The absolute tolerance test for constant derivatives is used as follows: If the value of the user function at point X0 is Y0 and the values at (X0-δX) and (X0+δX) are Yd and Yu respectively, then the numerical derivatives at X0 are: "down" derivative Dd = (Y0 - Yd) / δX "up" derivative Du = (Yu - Y0) / δX If abs(Dd-Du) $\leq$ xslp_cdTol_a then the derivative is regarded as constant.

Range: [0, ∞]

Default: 1e-08

xslp_cdTol_r *(real)*: Relative tolerance for deducing constant derivatives ↩

The relative tolerance test for constant derivatives is used as follows: If the value of the user function at point X0 is Y0 and the values at (X0-δX) and (X0+δX) are Yd and Yu respectively, then the numerical derivatives at X0 are: "down" derivative Dd = (Y0 - Yd) / δX "up" derivative Du = (Yu - Y0) / δX If abs(Dd-Du) $\leq$ xslp_cdTol_r $*$ abs(Yd+Yu)/2 then the derivative is regarded as constant.

Range: [0, ∞]

Default: 1e-08

xslp_clampShrink *(real)*: Shrink ratio used to impose strict convergence on variables converged in extended criteria only ↩

If the solution has converged but there are variables converged on extended criteria only, the xslp_clampShrink acts as a shrinking ratio on the step bounds and the problem is optimized (if necessary multiple times), with the purpose of expediting strict convergence on all variables. xslp_algorithm controls if this shrinking is applied at all, and if shrinking is applied to of the variables converged on extended criteria only with active step bounds only, or if on all variables.

Range: [0, 1]

Default: 0.3

xslp_clampValidationTol_a *(real)*: Absolute validation tolerance for applying xslp_clampShrink ↩

If set and the absolute validation value is larger than this value, then control xslp_clampShrink is checked once the solution has converged, but there are variables converged on extended criteria only.

Range: [0, ∞]

Default: not set

xslp_clampValidationTol_r *(real)*: Relative validation tolerance for applying xslp_clampShrink ↩

If set and the relative validation value is larger than this value, then control xslp_clampShrink is checked once the solution has converged, but there are variables converged on extended criteria only.

Range: [0, ∞]

Default: not set

xslp_convergenceOps *(integer)*: Bit map describing which convergence tests should be carried out ↩

Provides fine tuned control (over setting the related convergence tolerances) of which convergence checks are carried out.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 7167

value	meaning
bit 0 = 1	Equivalent to xslp_convergenceOps_cTol.
bit 1 = 2	Equivalent to xslp_convergenceOps_aTol.
bit 2 = 4	Equivalent to xslp_convergenceOps_mTol.
bit 3 = 8	Equivalent to xslp_convergenceOps_iTol.
bit 4 = 16	Equivalent to xslp_convergenceOps_sTol.
bit 5 = 32	Check for user provided convergence.
bit 6 = 64	Equivalent to xslp_convergenceOps_vTol.
bit 7 = 128	Equivalent to xslp_convergenceOps_xTol.
bit 8 = 256	Equivalent to xslp_convergenceOps_oTol.
bit 9 = 512	Equivalent to xslp_convergenceOps_wTol.
bit 10 = 1024	Equivalent to xslp_convergenceOps_extendedScaling.
bit 11 = 2048	Equivalent to xslp_convergenceOps_validation.
bit 12 = 4096	Equivalent to xslp_convergenceOps_validationK.

xslp_convergenceOps_aTol *(boolean)*: Execute the delta tolerance checks ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_cTol *(boolean)*: Execute the closure tolerance checks ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_extendedScaling *(boolean)*: Take scaling of individual variables / rows into account ↩

See also xslp_convergenceOps.

Default: 0

xslp_convergenceOps_iTol *(boolean)*: Execute the impact tolerance checks ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_mTol *(boolean)*: Execute the matrix tolerance checks ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_oTol *(boolean)*: Execute the objective range + active step bound check ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_sTol *(boolean)*: Execute the slack impact tolerance checks ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_validation *(boolean)*: Execute the validation target convergence checks ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_validationK *(boolean)*: Execute the first order optimality target convergence checks ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_vTol *(boolean)*: Execute the objective range checks ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_wTol *(boolean)*: Execute the convergence continuation check ↩

See also xslp_convergenceOps.

Default: 1

xslp_convergenceOps_xTol *(boolean)*: Execute the objective range + constraint activity check ↩

See also xslp_convergenceOps.

Default: 1

xslp_cTol *(real)*: Closure convergence tolerance ↩

The closure convergence criterion measures the change in value of a variable (δX) relative to the value of its initial step bound (B), against the closure convergence tolerance. If $\delta X < B * xslp\_cTol$ then the variable has converged on the closure convergence criterion. If no explicit initial step bound is provided, then the test will not be applied and the variable can never converge on the closure criterion. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_cutStrategy *(integer)*: Determines whihc cuts to apply in the MISLP search when the default SLP-in-MIP strategy is used ↩

Cuts are derived from the linearizations and are local cuts in that they are valid in the linearization and not necessarily valid for the full problem. The values mirror that of XPRS_CUTSTRATEGY.

Range: {-1, ..., 3}

Default: 0

xslp_damp *(real)*: Damping factor for updating values of variables ↩

The damping factor sets the next assumed value for a variable based on the previous assumed value (X0) and the actual value (X1). The new assumed value is given by X1∗xslp_damp + X0∗(1-xslp_damp)

Range: [0, 1]

Default: 1

xslp_dampExpand *(real)*: Multiplier to increase damping factor during dynamic damping ↩

If dynamic damping is enabled, the damping factor for a variable will be increased if successive changes are in the same direction. More precisely, if there are xslp_sameDamp successive changes in the same direction for a variable, then the damping factor (D) for the variable will be reset to D∗xslp_dampExpand + xslp_dampMax∗(1-xslp_dampExpand)

Range: [0, 1]

Default: 1

xslp_dampMax *(real)*: Maximum value for the damping factor of a variable during dynamic damping ↩

If dynamic damping is enabled, the damping factor for a variable will be increased if successive changes are in the same direction. More precisely, if there are xslp_sameDamp successive changes in the same direction for a variable, then the damping factor (D) for the variable will be reset to D∗xslp_dampExpand + xslp_dampMax∗(1-xslp_dampExpand)

Range: [0, 1]

Default: 1

xslp_dampMin *(real)*: Minimum value for the damping factor of a variable during dynamic damping ↩

If dynamic damping is enabled, the damping factor for a variable will be decreased if successive changes are in the opposite direction. More precisely, the damping factor (D) for the variable will be reset to D∗xslp_dampShrink + xslp_dampMin∗(1-xslp_dampExpand)

Range: [0, 1]

Default: 1

xslp_dampShrink *(real)*: Multiplier to decrease damping factor during dynamic damping ↩

If dynamic damping is enabled, the damping factor for a variable will be decreased if successive changes are in the opposite direction. More precisely, the damping factor (D) for the variable will be reset to D∗xslp_dampShrink + xslp_dampMin∗(1-xslp_dampExpand)

Range: [0, 1]

Default: 1

xslp_dampStart *(integer)*: SLP iteration at which damping is activated ↩

If damping is used as part of the SLP algorithm, it can be delayed until a specified SLP iteration. This may be appropriate when damping is used to encourage convergence after an un-damped algorithm has failed to converge.

Range: {0, ..., ∞}

Default: 0

xslp_defaultStepBound *(real)*: Minimum initial value for the step bound of an SLP variable if none is explicitly given ↩

If no initial step bound value is given for an SLP variable, this will be used as a minimum value. If the algorithm is estimating step bounds, then the step bound actually used for a variable may be larger than the default. A default initial step bound is ignored when testing for the closure tolerance xslp_cTol: if there is no specific value, then the test will not be applied.

Range: [0, ∞]

Default: 16

xslp_deltaCost *(real)*: Initial penalty cost multiplier for penalty delta vectors ↩

If penalty delta vectors are used, this parameter sets the initial cost factor. If there are active penalty delta vectors, then the penalty cost may be increased.

Range: [0, ∞]

Default: 200

xslp_deltaCostFactor *(real)*: Factor for increasing cost multiplier on total penalty delta vectors ↩

If there are active penalty delta vectors, then the penalty cost multiplier will be increased by a factor of xslp_deltaCostFactor up to a maximum of xslp_deltaMaxCost

Range: [1, ∞]

Default: 1.3

xslp_deltaMaxCost *(real)*: Maximum penalty cost multiplier for penalty delta vectors ↩

If there are active penalty delta vectors, then the penalty cost multiplier will be increased by a factor of xslp_deltaCostFactor up to a maximum of xslp_deltaMaxCost

Range: [0, ∞]

Default: 1e+20

xslp_deltaZLimit *(integer)*: Number of SLP iterations during which to apply xslp_delta_z ↩

xslp_delta_z is used to retain small derivatives which would otherwise be regarded as zero. This is helpful in avoiding local optima, but may make the linearized problem more difficult to solve because of the number of small nonzero elements in the resulting matrix. xslp_deltaZLimit can be set to a nonzero value, which is then the number of iterations for which xslp_delta_z will be used. After that, small derivatives will be set to zero. A negative value indicates no automatic perturbations to the derivatives in any situation.

Range: {0, ..., ∞}

Default: 0

xslp_delta_a *(real)*: Absolute perturbation of values for calculating numerical derivatives ↩

First-order derivatives are calculated by perturbing the value of each variable in turn by a small amount. The amount is determined by the absolute and relative delta factors as follows: xslp_delta_a + abs(X)*xslp_delta_r where (X) is the current value of the variable. If the perturbation takes the variable outside a bound, then the perturbation normally made only in the opposite direction.

Range: [0, ∞]

Default: 0.001

xslp_delta_r *(real)*: Relative perturbation of values for calculating numerical derivatives ↩

First-order derivatives are calculated by perturbing the value of each variable in turn by a small amount. The amount is determined by the absolute and relative delta factors as follows: xslp_delta_a + abs(X)*xslp_delta_r where (X) is the current value of the variable. If the perturbation takes the variable outside a bound, then the perturbation normally made only in the opposite direction.

Range: [0, ∞]

Default: 0.001

xslp_delta_x *(real)*: Minimum absolute value of delta coefficients to be retained ↩

If the value of a coefficient in a delta column is less than this value, it will be reset to zero. Larger values of xslp_delta_x will result in matrices with fewer elements, which may be easier to solve. However, there will be increased likelihood of local optima as some of the small relationships between variables and constraints are deleted. There may also be increased difficulties with singular bases resulting from deletion of pivot elements from the matrix.

Range: [0, ∞]

Default: 1e-06

xslp_delta_z *(real)*: Tolerance used when calculating derivatives ↩

If the absolute value of a variable is less than this value, then a value of xslp_delta_z will be used instead for calculating derivatives. If a nonzero derivative is calculated for a formula which always results in a matrix coefficient less than xslp_delta_z, then a larger value will be substituted so that at least one of the coefficients is xslp_delta_z in magnitude. If xslp_deltaZLimit is set to a positive number, then when that number of iterations have passed, values smaller than xslp_delta_z will be set to zero.

Range: [0, ∞]

Default: 1e-05

xslp_delta_zero *(real)*: Absolute zero acceptance tolerance used when calculating derivatives ↩

Provides an override value for the xslp_delta_z behavior. Derivatives smaller than xslp_delta_zero will not be substituted by xslp_delta_z, defining a range in which derivatives are deemed nonzero and are affected by xslp_delta_z. A negative value means that this tolerance will not be applied.

Range: [-∞, ∞]

Default: -1

xslp_derivatives *(boolean)*: Bitmap describing the method of calculating derivatives ↩

If no bits are set then numerical derivatives are calculated using finite differences. Analytic derivatives cannot be used for formulae involving discontinuous functions. They may not work well with functions which are not smooth (such as MAX), or where the derivative changes very quickly with the value of the variable (such as LOG of small values). Both first and second order analytic derivatives can either be calculated as symbolic formulas, or by the means of auto-differentiation, with the exception that the second order symbolic derivatives require that the first order derivatives are also calculated using the symbolic method.

Default: 1

value	meaning
bit 0 = 1	Analytic derivatives where possible
bit 1 = 2	Avoid embedding numerical derivatives of instantiated functions into analytic derivatives

xslp_djTol *(real)*: Tolerance on DJ value for determining if a variable is at its step bound ↩

If a variable is at its step bound and within the absolute delta tolerance xslp_aTol_a or closure tolerance xslp_cTol then the step bounds will not be further reduced. If the DJ is greater in magnitude than xslp_djTol then the step bound may be relaxed if it meets the necessary criteria.

Range: [0, ∞]

Default: 1e-06

xslp_ecfCheck *(integer)*: Check feasibility at the point of linearization for extended convergence criteria ↩

The extended convergence criteria measure the accuracy of the solution of the linear approximation compared to the solution of the original nonlinear problem. For this to work, the linear approximation needs to be reasonably good at the point of linearization. In particular, it needs to be reasonably close to feasibility. xslp_ecfCheck is used to determine what checking of feasibility is carried out at the point of linearization. If the point of linearization at the start of an SLP iteration is deemed to be infeasible, then the extended convergence criteria are not used to decide convergence at the end of that SLP iteration. If all that is required is to decide that the point of linearization is not feasible, then the search can stop after the first infeasible constraint is found (parameter is set to 1). If the actual number of infeasible constraints is required, then xslp_ecfCheck should be set to 2, and all constraints will be checked. The number of infeasible constraints found at the point of linearization is returned in XSLP_ECFCOUNT.

Default: 1

value	meaning
0	No check (extended criteria are always used);
1	Check until one infeasible constraint is found;
2	Check all constraints.

xslp_ecfTol_a *(real)*: Absolute tolerance on testing feasibility at the point of linearization ↩

The extended convergence criteria test how well the linearization approximates the true problem. They depend on the point of linearization being a reasonable approximation — in particular, that it should be reasonably close to feasibility. Each constraint is tested at the point of linearization, and the total positive and negative contributions to the constraint from the columns in the problem are calculated. A feasibility tolerance is calculated as the largest of xslp_ecfTol_a and max(abs(Positive), abs(Negative)) ∗ xslp_ecfTol_r If the calculated infeasibility is greater than the tolerance, the point of linearization is regarded as infeasible and the extended convergence criteria will not be applied. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-1 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_ecfTol_r *(real)*: Relative tolerance on testing feasibility at the point of linearization ↩

The extended convergence criteria test how well the linearization approximates the true problem. They depend on the point of linearization being a reasonable approximation — in particular, that it should be reasonably close to feasibility. Each constraint is tested at the point of linearization, and the total positive and negative contributions to the constraint from the columns in the problem are calculated. A feasibility tolerance is calculated as the largest of xslp_ecfTol_a and max(abs(Positive), abs(Negative)) ∗ xslp_ecfTol_r If the calculated infeasibility is greater than the tolerance, the point of linearization is regarded as infeasible and the extended convergence criteria will not be applied. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-1 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_enforceCostShrink *(real)*: Factor by which to decrease the current penalty multiplier when enforcing rows ↩

When feasiblity of a row cannot be achieved by increasing the penalty cost on its error variable, removing the variable (fixing it to zero) can force the row to be satisfied, as set by xslp_enforceMaxCost. After the error variables have been removed (which is equivalent to setting to row to be enforced) the penalties on the remaining error variables are rebalanced to allow for a reduction in the size of the penalties in the objective in order to achive better numerical behaviour.

Range: [`0`, `1`]

Default: `1e-05`

xslp_enforceMaxCost *(real)*: Maximum penalty cost in the objective before enforcing most violating rows ↩

When feasiblity of a row cannot be achieved by increasing the penalty cost on its error variable, removing the variable (fixing it to zero) can force the row to be satisfied. After the error variables have been removed (which is equivalent to setting to row to be enforced) the penalties on the remaining error variables are rebalanced to allow for a reduction in the size of the penalties in the objective in order to achive better numerical behaviour, controlled by xslp_enforceCostShrink.

Range: [`0`, ∞]

Default: `1e+11`

xslp_errorCost *(real)*: Initial penalty cost multiplier for penalty error vectors ↩

If penalty error vectors are used, this parameter sets the initial cost factor. If there are active penalty error vectors, then the penalty cost may be increased.

Range: [`0`, ∞]

Default: `200`

xslp_errorCostFactor *(real)*: Factor for increasing cost multiplier on total penalty error vectors ↩

If there are active penalty error vectors, then the penalty cost multiplier will be increased by a factor of xslp_errorCostFactor up to a maximum of xslp_errorMaxCost

Range: [1, ∞]

Default: 1.3

xslp_errorMaxCost *(real)*: Maximum penalty cost multiplier for penalty error vectors ↩

If there are active penalty error vectors, then the penalty cost multiplier will be increased by a factor of xslp_errorCostFactor up to a maximum of xslp_errorMaxCost

Range: [0, ∞]

Default: 1e+20

xslp_errorTol_a *(real)*: Absolute tolerance for error vectors ↩

The solution will be regarded as having no active error vectors if one of the following applies: every penalty error vector and penalty delta vector has an activity less than xslp_errorTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_r ∗ Obj where Obj is the current objective function value.

Range: [0, ∞]

Default: 1e-05

xslp_errorTol_p *(real)*: Absolute tolerance for printing error vectors ↩

The solution log includes a print of penalty delta and penalty error vectors with an activity greater than xslp_errorTol_p.

Range: [0, ∞]

Default: 0.0001

xslp_escalation *(real)*: Factor for increasing cost multiplier on individual penalty error vectors ↩

If penalty cost escalation is activated in xslp_algorithm then the penalty cost multiplier will be increased by a factor of xslp_escalation for any active error vector up to a maximum of xslp_maxWeight.

Range: [1, ∞]

Default: 1.25

xslp_eTol_a *(real)*: Absolute tolerance on penalty vectors ↩

For each penalty error vector, the contribution to its constraint is calculated, together with the total positive and negative contributions to the constraint from other vectors. If its contribution is less than xslp_eTol_a or less than Positive∗xslp_eTol_r or less than abs(Negative)∗xslp_eTol_r then it will be regarded as insignificant and will not have its penalty increased. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: 0.0001

xslp_eTol_r *(real)*: Relative tolerance on penalty vectors ↩

For each penalty error vector, the contribution to its constraint is calculated, together with the total positive and negative contributions to the constraint from other vectors. If its contribution is less than xslp_eTol_a or less than Positive∗xslp_eTol_r or less than abs(Negative)∗xslp_eTol_r then it will be regarded as insignificant and will not have its penalty increased. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: `0.0001`

xslp_evTol_a *(real)*: Absolute tolerance on total penalty costs ↩

The solution will be regarded as having no active error vectors if one of the following applies: every penalty error vector and penalty delta vector has an activity less than xslp_errorTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_r ∗ Obj where Obj is the current objective function value. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-2 and 1e-6, but normally a magnitude larger than xslp_eTol_a.

Range: [-∞, ∞]

Default: `-1`

xslp_evTol_r *(real)*: Relative tolerance on total penalty costs ↩

The solution will be regarded as having no active error vectors if one of the following applies: every penalty error vector and penalty delta vector has an activity less than xslp_errorTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_a; the sum of the cost contributions from all the penalty error and penalty delta vectors is less than xslp_evTol_r ∗ Obj where Obj is the current objective function value. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-2 and 1e-6, but normally a magnitude larger than xslp_eTol_r.

Range: [-∞, ∞]

Default: `-1`

xslp_expand *(real)*: Multiplier to increase a step bound ↩

If step bounding is enabled, the step bound for a variable will be increased if successive changes are in the same direction. More precisely, if there are xslp_sameCount successive changes reaching the step bound and in the same direction for a variable, then the step bound (B) for the variable will be reset to B∗xslp_expand.

Range: [1, ∞]

Default: `2`

xslp_feasTolTarget *(real)*: When set, this defines a target feasibility tolerance to which the linearizations are solved to ↩

This is a soft version of XPRS_FEASTOL, and will dynamically revert back to XPRS_FEASTOL if the desired accuracy could not be achieved.

Range: [0, ∞]

Default: `not set`

xslp_filter *(integer)*: Bit map for controlling solution updates ↩

Bits 0 determine if XSLPgetslpsol should return the final converged solution, or the solution which had the best value according to the merit function. If bit 1 is set, a cascaded solution which does not improve the merit function will be rejected (Xpress SLP will revert to the solution of the linearization). Bits 2-3 determine the strategy for when the step direction is not improving according to the merit function.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: `3`

value	meaning
`bit 0 = 1`	Equivalent to xslp_filterKeepBest.
`bit 1 = 2`	Check cascaded solutions against improvements in the merit function.
`bit 2 = 4`	Equivalent to xslp_filterZeroLineSearch.
`bit 3 = 8`	Equivalent to xslp_filterZeroLineSearchTR.

xslp_filterKeepBest *(boolean)*: Retrain solution best according to the merit function ↩

See also xslp_filter.

Default: `1`

xslp_filterZeroLineSearch *(boolean)*: Force minimum step sizes in line search ↩

See also xslp_filter.

Default: `0`

xslp_filterZeroLineSearchTR *(boolean)*: Accept the trust region step is the line search returns a zero step size ↩

See also xslp_filter.

Default: `0`

xslp_findIV *(integer)*: Option for running a heuristic to find a feasible initial point ↩

The procedure uses bound reduction (and, up to an extent, probing) to obtain a point in the initial bounding box that is feasible for the bound reduction techniques. If an initial point is already specified and is found not to violate bound reduction, then the heuristic is not run and the given point is used as the initial solution.

Default: `auto`

value	meaning
`-1`	Automatic (default).
`0`	Disable the heuristic.
`1`	Enable the heuristic.

xslp_granularity *(real)*: Base for calculating penalty costs ↩

If xslp_granularity > 1, then initial penalty costs will be powers of xslp_granularity.

Range: [1, ∞]

Default: 4

xslp_hessian *(integer)*: Second order differentiation mode when using analytical derivatives ↩

Symbolic mode differentiation for the second order derivatives is only available when xslp_jacobian is also set to symbolic mode.

Default: -1

value	meaning
1	Numerical derivatives (finite difference)
2	Symbolic differentiation
3	Automatic differentiation
-1,0	Automatic selection

xslp_heurStrategy *(integer)*: Branch and Bound: MINLP heuristic strategy ↩

On some problems it is worth trying more comprehensive heuristic strategies by setting HEURSTRATEGY to 2 or 3. Note that HEURSTRATEGY is deprecated, use heurEmphasis instead.

Default: auto

value	meaning
-1	Automatic selection of heuristic strategy.
0	No heuristics.
1	Basic heuristic strategy.
2	Enhanced heuristic strategy.
3	Extensive heuristic strategy.
4	Run all heuristics without effort limits.

xslp_infeasLimit *(integer)*: Maximum number of consecutive infeasible SLP iterations which can occur before Xpress-SLP terminates ↩

An infeasible solution to an SLP iteration means that is likely that Xpress-SLP will create a poor linear approximation for the next SLP iteration. Sometimes, small infeasibilities arise because of numerical difficulties and do not seriously affect the solution process. However, if successive solutions remain infeasible, it is unlikely that Xpress-SLP will be able to find a feasible converged solution. xslp_infeasLimit sets the number of successive SLP iterations which must take place before Xpress-SLP terminates with a status of "infeasible solution".

Range: {0, ..., ∞}

Default: 3

xslp_infinity *(real)*: Value returned by a divide-by-zero in a formula ↩

Range: [0, ∞]

Default: 1e+10

xslp_iterLimit *(integer)*: Maximum number of SLP iterations ↩

If Xpress-SLP reaches xslp_iterLimit without finding a converged solution, it will stop. For MISLP, the limit is on the number of SLP iterations at each node.

Range: {-1, ..., ∞}

Default: auto

xslp_iTol_a *(real)*: Absolute impact convergence tolerance ↩

The absolute impact convergence criterion assesses the change in the effect of a coefficient in a constraint. The effect of a coefficient is its value multiplied by the activity of the column in which it appears. E = X ∗ C where X is the activity of the matrix column in which the coefficient appears, and C is the value of the coefficient. The linearization approximates the effect of the coefficient as E1 = X ∗ C0 + δX ∗ C'0 where X is as before, C0 is the value of the coefficient C calculated using the assumed values for the variables and C'0 is the value of ∂C ∂X calculated using the assumed values for the variables. If C1 is the value of the coefficient C calculated using the actual values for the variables, then the error in the effect of the coefficient is given by δE = X ∗ C1 - (X ∗ C0 + δX ∗ C'0) If δE < xslp_iTol_a then the variable has passed the absolute impact convergence criterion for this coefficient. If a variable which has not converged on strict (closure or delta) criteria passes the (relative or absolute) impact or matrix criteria for all the coefficients in which it appears, then it is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_iTol_r *(real)*: Relative impact convergence tolerance ↩

The relative impact convergence criterion assesses the change in the effect of a coefficient in a constraint in relation to the magnitude of the constituents of the constraint. The effect of a coefficient is its value multiplied by the activity of the column in which it appears. E = X ∗ C where X is the activity of the matrix column in which the coefficient appears, and C is the value of the coefficient. The linearization approximates the effect of the coefficient as E1 = X ∗ C0 + δX ∗ C'0 where X is as before, C0 is the value of the coefficient C calculated using the assumed values for the variables and C'0 is the value of ∂C ∂X calculated using the assumed values for the variables. If C1 is the value of the coefficient C calculated using the actual values for the variables, then the error in the effect of the coefficient is given by δE = X ∗ C1 - (X ∗ C0 + δX ∗ C'0) All the elements of the constraint are examined, excluding delta and error vectors: for each, the contribution to the constraint is evaluated as the element multiplied by the activity of the vector in which it appears; it is then included in a total positive contribution or total negative contribution depending on the sign of the contribution. If the predicted effect of the coefficient is positive, it is tested against the total positive contribution; if the effect of the coefficient is negative, it is tested against the total negative contribution. If T0 is the total positive or total negative contribution to the constraint (as appropriate) and δE < T0∗xslp_iTol_r then the variable has passed the relative impact convergence criterion for this coefficient. If a variable which has not converged on strict (closure or delta) criteria passes the (relative or absolute) impact or matrix criteria for all the coefficients in which it appears, then it is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_jacobian *(integer)*: First order differentiation mode when using analytical derivatives ↩

Symbolic mode differentiation for the second order derivatives is only available when xslp_jacobian is set to symbolic mode.

Default: `-1`

value	meaning
`1`	Numerical derivatives (finite difference)
`2`	Symbolic differentiation
`3`	Automatic differentiation
`-1,0`	Automatic selection

xslp_linQuadBR *(integer)*: Use linear and quadratic constraints and objective function to further reduce bounds on all variables ↩

While bound reduction is effective when performed on nonlinear, nonquadratic constraints and objective function, it can be useful to obtain tightened bounds from linear and quadratic constraints, as the corresponding variables may appear in other nonlinear constraints. This option then allows for a slightly more expensive bound reduction procedure, at the benefit of further reduction in the problem's bounds.

Default: `auto`

value	meaning
`-1`	Automatic selection
`0`	Disable
`1`	Enable

xslp_log *(integer)*: Level of printing during SLP iterations ↩

Default: `0`

value	meaning
`-1`	None
`0`	Minimal
`1`	Normal: iteration, penalty vectors
`2`	Omit from convergence log any variables which have converged
`3`	Omit from convergence log any variables which have already converged (except variables on step bounds)
`4`	Include all variables in convergence log
`5`	Include user function call communications in the log

xslp_lsIterLimit *(integer)*: Number of iterations in the line search ↩

The line search attempts to refine the step size suggested by the trust region step bounds. The line search is a local method; the control sets a maximum on the number of model evaluations during the line search.

Range: {`0`, ..., ∞}

Default: `0`

xslp_lsPatternLimit *(integer)*: Number of iterations in the pattern search preceding the line search ↩

When positive, defines the number of samples taken along the step size suggested by the trust region step bounds before initiating the line search. Useful for highly non-convex problems.

Range: {0, ..., ∞}

Default: 0

xslp_lsStart *(integer)*: Iteration in which to active the line search ↩

Range: {0, ..., ∞}

Default: 8

xslp_lsZeroLimit *(integer)*: Maximum number of zero length line search steps before line search is deactivated ↩

When the line search repeatedly returns a zero step size, counteracted by bits set on xslp_filter, the effort spent in line search is redundant, and line search will be deactivated after xslp_lsZeroLimit consecutive such iteration.

Range: {0, ..., ∞}

Default: 5

xslp_maxWeight *(real)*: Maximum penalty weight for delta or error vectors ↩

When penalty vectors are created, or when their weight is increased by escalation, the maximum weight that will be used is given by xslp_maxWeight.

Range: [0, ∞]

Default: 100

xslp_meritLambda *(real)*: Factor by which the net objective is taken into account in the merit function ↩

The merit function is evaluated in the original, non-augmented / linearized space of the problem. A solution is deemed improved, if either feasibility improved, or if feasibility is not deteriorated but the net objective is improved, or if the combination of the two is improved, where the value of the xslp_meritLambda control is used to combine the two measures. A nonpositive value indicates that the combined effect should not be checked.

Range: [0, ∞]

Default: 0

xslp_minSBFactor *(real)*: Factor by which step bounds can be decreased beneath xslp_aTol_a ↩

Normally, step bounds are not decreased beneath xslp_aTol_a, as such variables are treated as converged. However, it may be beneficial to decrease step bounds further, as individual variable value changes might affect the convergence of other variables in the model, even if the variablke itself is deemed converged.

Range: [0, ∞]

Default: 1

xslp_minWeight *(real)*: Minimum penalty weight for delta or error vectors ↩

When penalty vectors are created, the minimum weight that will be used is given by xslp_minWeight.

Range: [0, ∞]

Default: 0.01

xslp_mipAlgorithm *(integer)*: Bitmap describing the MISLP algorithms to be used ↩

xslp_mipAlgorithm determines the strategy of XSLPnlpoptimize for solving MINLP problems. The recommended approach is to solve the problem first without reference to the discrete variables. This can be handled automatically by setting bit 0 of xslp_mipAlgorithm; if done manually, then optimize using the "l" option to prevent the Optimizer presolve from changing the problem. Some versions of the optimizer re-run the initial node as part of the tree search; it is possible to initiate a new SLP optimization at this point by relaxing or fixing step bounds (use bits 2 and 3). If step bounds are fixed for a class of variable, then the variables in that class will not change their value in any child node. At each node, it is possible to relax or fix step bounds. It is recommended that step bounds are relaxed, so that the new problem can be solved starting from its parent, but without undue restrictions cased by step bounding (use bit 4). Exceptionally, it may be preferable to restrict the freedom of child nodes by relaxing fewer types of step bound or fixing the values of some classes of variable (use bit 5). When the optimal node has been found, it is possible to fix the discrete variables and then re-optimize with SLP. Step bounds can be relaxed or fixed for this optimization as well (use bits 7 and 8). Although it is ultimately necessary to solve the optimal node to convergence, individual nodes can be truncated after xslp_mipIterLimit SLP iterations. Set bit 6 to activate this feature. The normal MISLP algorithm uses SLP at each node. One alternative strategy is to use the MIP optimizer for solving each SLP iteration. Set bit 9 to implement this strategy ("MIP within SLP"). Another strategy is to solve the problem to convergence ignoring the nature of the integer variables. Then, fixing the linearization, use MIP to find the optimal setting of the discrete variables. Then, fixing the discrete variables, but varying the linearization, solve to convergence. Set bit 10 to implement this strategy ("SLP then MIP"). For mode details about MISLP algorithms and strategies, see the separate section.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 17

value	meaning
bit 0 = 1	Equivalent to xslp_mipAlgorithm_initialSLP.
bit 2 = 4	Equivalent to xslp_mipAlgorithm_initialRelaxSLP.
bit 3 = 8	Equivalent to xslp_mipAlgorithm_initialFixSLP.
bit 4 = 16	Equivalent to xslp_mipAlgorithm_nodeRelaxSLP.
bit 5 = 32	Equivalent to xslp_mipAlgorithm_nodeFixSLP.
bit 6 = 64	Equivalent to xslp_mipAlgorithm_nodeLimitSLP.
bit 7 = 128	Equivalent to xslp_mipAlgorithm_finalRelaxSLP.
bit 8 = 256	Equivalent to xslp_mipAlgorithm_finalFixSLP.
bit 9 = 512	Equivalent to xslp_mipAlgorithm_withinSLP.
bit 10 = 1024	Equivalent to xslp_mipAlgorithm_slpThenMIP.

xslp_mipAlgorithm_finalFixSLP *(boolean)*: Fix step bounds according to xslp_mipFixStepBounds after MIP solution is found ↩

See also xslp_mipAlgorithm.

Default: 0

xslp_mipAlgorithm_finalRelaxSLP *(boolean)*: Relax step bounds according to xslp_mipRelaxStepBounds after MIP solution is found ↩

See also xslp_mipAlgorithm.

Default: 0

xslp_mipAlgorithm_initialFixSLP *(boolean)*: Fix step bounds according to xslp_mipFixStepBounds after initial node ↩

See also xslp_mipAlgorithm.

Default: 0

xslp_mipAlgorithm_initialRelaxSLP *(boolean)*: Relax step bounds according to xslp_mipRelaxStepBounds after initial node ↩

See also xslp_mipAlgorithm.

Default: 0

xslp_mipAlgorithm_initialSLP *(boolean)*: Solve initial SLP to convergence ↩

See also xslp_mipAlgorithm.

Default: 1

xslp_mipAlgorithm_nodeFixSLP *(boolean)*: Fix step bounds according to xslp_mipFixStepBounds at each node ↩

See also xslp_mipAlgorithm.

Default: 0

xslp_mipAlgorithm_nodeLimitSLP *(boolean)*: Limit iterations at each node to xslp_mipIterLimit ↩

See also xslp_mipAlgorithm.

Default: 0

xslp_mipAlgorithm_nodeRelaxSLP *(boolean)*: Relax step bounds according to xslp_mipRelaxStepBounds at each node ↩

See also xslp_mipAlgorithm.

Default: 1

xslp_mipAlgorithm_slpThenMIP *(boolean)*: Use MIP on converged SLP solution and then SLP on the resulting MIP solution ↩

See also xslp_mipAlgorithm.

Default: 0

xslp_mipAlgorithm_withinSLP *(boolean)*: Use MIP at each SLP iteration instead of SLP at each node ↩

See also xslp_mipAlgorithm.

Default: 0

xslp_mipCutOffCount *(integer)*: Number of SLP iterations to check when considering a node for cutting off ↩

If the objective function is worse by a defined amount than the best integer solution obtained so far, then the SLP will be terminated (and the node will be cut off). The node will be cut off at the current SLP iteration if the objective function for the last xslp_mipCutOffCount SLP iterations are all worse than the best obtained so far, and the difference is greater than xslp_mipCutOff_a and OBJ ∗ xslp_mipCutOff_r where OBJ is the best integer solution obtained so far. The test is not applied until at least xslp_mipCutOffLimit SLP iterations have been carried out at the current node.

Range: {0, ..., ∞}

Default: 5

xslp_mipCutOffLimit *(integer)*: Number of SLP iterations to check when considering a node for cutting off ↩

If the objective function is worse by a defined amount than the best integer solution obtained so far, then the SLP will be terminated (and the node will be cut off). The node will be cut off at the current SLP iteration if the objective function for the last xslp_mipCutOffCount SLP iterations are all worse than the best obtained so far, and the difference is greater than xslp_mipCutOff_a and OBJ ∗ xslp_mipCutOff_r where OBJ is the best integer solution obtained so far. The test is not applied until at least xslp_mipCutOffLimit SLP iterations have been carried out at the current node.

Range: {0, ..., ∞}

Default: 10

xslp_mipCutOff_a *(real)*: Absolute objective function cutoff for MIP termination ↩

If the objective function is worse by a defined amount than the best integer solution obtained so far, then the SLP will be terminated (and the node will be cut off). The node will be cut off at the current SLP iteration if the objective function for the last xslp_mipCutOffCount SLP iterations are all worse than the best obtained so far, and the difference is greater than xslp_mipCutOff_a and OBJ ∗ xslp_mipCutOff_r where OBJ is the best integer solution obtained so far. The MIP cutoff tests are only applied after xslp_mipCutOffLimit SLP iterations at the current node.

Range: [0, ∞]

Default: 1e-05

xslp_mipCutOff_r *(real)*: Absolute objective function cutoff for MIP termination ↩

If the objective function is worse by a defined amount than the best integer solution obtained so far, then the SLP will be terminated (and the node will be cut off). The node will be cut off at the current SLP iteration if the objective function for the last xslp_mipCutOffCount SLP iterations are all worse than the best obtained so far, and the difference is greater than xslp_mipCutOff_a and OBJ ∗ xslp_mipCutOff_r where OBJ is the best integer solution obtained so far. The MIP cutoff tests are only applied after xslp_mipCutOffLimit SLP iterations at the current node.

Range: [0, ∞]

Default: 1e-05

xslp_mipDefaultAlgorithm *(integer)*: Default algorithm to be used during the tree search in MISLP ↩

The default algorithm used within SLP during the MISLP optimization can be set using xslp_mipDefaultAlgorithm. It will not necessarily be the same as the one best suited to the initial SLP optimization.

Range: {1, ..., 5}

Default: auto

xslp_mipErrorTol_a *(real)*: Absolute penalty error cost tolerance for MIP cut-off ↩

The penalty error cost test is applied at each node where there are active penalties in the solution. If xslp_mipErrorTol_a is nonzero and the absolute value of the penalty costs is greater than xslp_mipErrorTol_a, the node will be declared infeasible. If xslp_mipErrorTol_a is zero then no test is made and the node will not be declared infeasible on this criterion.

Range: [0, ∞]

Default: 0

xslp_mipErrorTol_r *(real)*: Relative penalty error cost tolerance for MIP cut-off ↩

The penalty error cost test is applied at each node where there are active penalties in the solution. If xslp_mipErrorTol_r is nonzero and the absolute value of the penalty costs is greater than xslp_mipErrorTol_r ∗ abs(Obj) where Obj is the value of the objective function, then the node will be declared infeasible. If xslp_mipErrorTol_r is zero then no test is made and the node will not be declared infeasible on this criterion.

Range: [0, ∞]

Default: 0

xslp_mipFixStepBounds *(integer)*: Bitmap describing the step-bound fixing strategy during MISLP ↩

At any node (including the initial and optimal nodes) it is possible to fix the step bounds of classes of variables so that the variables themselves will not change. This may help with convergence, but it does increase the chance of a local optimum because of excessive artificial restrictions on the variables.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 0

value	meaning
bit 0 = 1	Equivalent to xslp_mipFixStepBounds_structNotCoef.
bit 1 = 2	Equivalent to xslp_mipFixStepBounds_structAll.
bit 2 = 4	Equivalent to xslp_mipFixStepBounds_coefOnly.
bit 3 = 8	Equivalent to xslp_mipFixStepBounds_coef.

xslp_mipFixStepBounds_coef *(boolean)*: Fix step bounds on SLP variables appearing in coefficients ↩

See also xslp_mipFixStepBounds.

Default: 0

xslp_mipFixStepBounds_coefOnly *(boolean)*: Fix step bounds on SLP variables appearing only in coefficients ↩

See also xslp_mipFixStepBounds.

Default: 0

xslp_mipFixStepBounds_structAll *(boolean)*: Fix step bounds on all structural SLP variables ↩

See also xslp_mipFixStepBounds.

Default: 0

xslp_mipFixStepBounds_structNotCoef *(boolean)*: Fix step bounds on structural SLP variables which are not in coefficients ↩

See also xslp_mipFixStepBounds.

Default: 0

xslp_mipIterLimit *(integer)*: Maximum number of SLP iterations at each node ↩

If bit 6 of xslp_mipAlgorithm is set, then the number of iterations at each node will be limited to xslp_mipIterLimit.

Range: {0, ..., ∞}

Default: 0

xslp_mipLog *(integer)*: Frequency with which MIP status is printed ↩

By default (zero or negative value) the MIP status is printed after syncronization points. If xslp_mipLog is set to a positive integer, then the current MIP status (node number, best value, best bound) is printed every xslp_mipLog nodes.

Range: {0, ..., ∞}

Default: 0

xslp_mipOCount *(integer)*: Number of SLP iterations at each node over which to measure objective function variation ↩

The objective function test for MIP termination is applied only when step bounding has been applied (or xslp_sbStart SLP iterations have taken place if step bounding is not being used). The node will be terminated at the current SLP iteration if the range of the objective function values over the last xslp_mipOCount SLP iterations is within xslp_mipOTol_a or within OBJ ∗ xslp_mipOTol_r where OBJ is the average value of the objective function over those iterations.

Range: {0, ..., ∞}

Default: 5

xslp_mipOTol_a *(real)*: Absolute objective function tolerance for MIP termination ↩

The objective function test for MIP termination is applied only when step bounding has been applied (or xslp_sbStart SLP iterations have taken place if step bounding is not being used). The node will be terminated at the current SLP iteration if the range of the objective function values over the last xslp_mipOCount SLP iterations is within xslp_mipOTol_a or within OBJ * xslp_mipOTol_r where OBJ is the average value of the objective function over those iterations.

Range: [0, ∞]

Default: 1e-05

xslp_mipOTol_r *(real)*: Relative objective function tolerance for MIP termination ↩

The objective function test for MIP termination is applied only when step bounding has been applied (or xslp_sbStart SLP iterations have taken place if step bounding is not being used). The node will be terminated at the current SLP iteration if the range of the objective function values over the last xslp_mipOCount SLP iterations is within xslp_mipOTol_a or within OBJ * xslp_mipOTol_r where OBJ is the average value of the objective function over those iterations.

Range: [0, ∞]

Default: 1e-05

xslp_mipRelaxStepBounds *(integer)*: Bitmap describing the step-bound relaxation strategy during MISLP ↩

At any node (including the initial and optimal nodes) it is possible to relax the step bounds of classes of variables so that the variables themselves are completely free to change. This may help with finding a global optimum, but it may also increase the solution time, because more SLP iterations are necessary at each node to obtain a converged solution.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 15

value	meaning
bit 0 = 1	Equivalent to xslp_mipRelaxStepBounds_structNotCoef.
bit 1 = 2	Equivalent to xslp_mipRelaxStepBounds_structAll.
bit 2 = 4	Equivalent to xslp_mipRelaxStepBounds_coefOnly.
bit 3 = 8	Equivalent to xslp_mipRelaxStepBounds_coef.

xslp_mipRelaxStepBounds_coef *(boolean)*: Relax step bounds on SLP variables appearing in coefficients ↩

See also xslp_mipRelaxStepBounds.

Default: 1

xslp_mipRelaxStepBounds_coefOnly *(boolean)*: Relax step bounds on SLP variables appearing only in coefficients ↩

See also xslp_mipRelaxStepBounds.

Default: 1

xslp_mipRelaxStepBounds_structAll *(boolean)*: Relax step bounds on all structural SLP variables ↩

See also xslp_mipRelaxStepBounds.

Default: 1

xslp_mipRelaxStepBounds_structNotCoef *(boolean)*: Relax step bounds on structural SLP variables which are not in coefficients ↩

See also xslp_mipRelaxStepBounds.

Default: 1

xslp_msMaxBoundRange *(real)*: Defines the maximum range inside which initial points are generated by multistart presets ↩

The is the maximum range in which initial points are generated; the actual range is expected to be smaller as bounds are domains are also considered.

Range: [0, ∞]

Default: 1000

xslp_mTol_a *(real)*: Absolute effective matrix element convergence tolerance ↩

The absolute effective matrix element convergence criterion assesses the change in the effect of a coefficient in a constraint. The effect of a coefficient is its value multiplied by the activity of the column in which it appears. E = X ∗ C where X is the activity of the matrix column in which the coefficient appears, and C is the value of the coefficient. The linearization approximates the effect of the coefficient as E = X ∗ C0 + δX ∗ C'0 where V is as before, C0 is the value of the coefficient C calculated using the assumed values for the variables and C'0 is the value of ∂C ∂X calculated using the assumed values for the variables. If C1 is the value of the coefficient C calculated using the actual values for the variables, then the error in the effect of the coefficient is given by δE = X ∗ C1 - (X ∗ C0 + δX ∗ C'0) If δE < X ∗ xslp_mTol_a then the variable has passed the absolute effective matrix element convergence criterion for this coefficient. If a variable which has not converged on strict (closure or delta) criteria passes the (relative or absolute) impact or matrix criteria for all the coefficients in which it appears, then it is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_mTol_r *(real)*: Relative effective matrix element convergence tolerance ↩

The relative effective matrix element convergence criterion assesses the change in the effect of a coefficient in a constraint relative to the magnitude of the coefficient. The effect of a coefficient is its value multiplied by the activity of the column in which it appears. E = X ∗ C where X is the activity of the matrix column in which the coefficient appears, and C is the value of the coefficient. The linearization approximates the effect of the coefficient as E1 = X ∗ C0 + δX ∗ C'0 where V is as before, C0 is the value of the coefficient C calculated using the assumed values for the variables and C'0 is the value of ∂C ∂X calculated using the assumed values for the variables. If C1 is the value of the coefficient C calculated using the actual values for the variables, then the error in the effect of the coefficient is given by δE = X ∗ C1 - (X ∗ C0 + δX ∗ C'0) If δE < E1 ∗ xslp_mTol_r then the variable has passed the relative effective matrix element convergence criterion for this coefficient. If a variable which has not converged on strict (closure or delta) criteria passes the (relative or absolute) impact or matrix criteria for all the coefficients in which it appears, then it is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_multistartPreset *(integer)*: Enable multistart ↩

Default: 0

value	meaning
0	Disable multistart preset.
1	Generate xslp_multistart_maxSolves number of random base points.
2	Generate xslp_multistart_maxSolves number of random base points, filtered by a merit function centred on initial feasibility.
3	Load the most typical SLP tuning settings. A maximum of xslp_multistart_maxSolves jobs are loaded.
4	Load a comprehensive set of SLP tuning settings. A maximum of xslp_multistart_maxSolves jobs are loaded.

xslp_multistart_maxSolves *(integer)*: Maximum number of jobs to create during the multistart search ↩

This control can be increased on the fly during the mutlistart search: for example, if a job gets refused by a user callback, the callback may increase this limit to account for the rejected job.

Range: {-1, ..., ∞}

Default: `unlimited`

xslp_multistart_maxTime *(integer)*: Maximum total time to be spent in the mutlistart search ↩

XSLP_MAXTIME applies on a per job instance basis. There will be some time spent even after xslp_multistart_maxTime has elapsed, while the running jobs get terminated and their results collected.

Range: {0, ..., ∞}

Default: `unlimited`

xslp_multistart_poolsize *(integer)*: Maximum number of problem objects allowed to pool up before synchronization in the deterministic multistart ↩

Deterministic multistart is ensured by guaranteeing that the multistart solve results are evaluated in the same order every time. Solves that finish too soon can be pooled until all earlier started solves finish, allowing the system to start solving other multistart instances in the meantime on idle threads. Larger pool sizes will provide better speedups, but will require larger amounts of memory. Positive values are interpreted as a multiplier on the maximum number of active threads used, while negative values are interpreted as an absolute limit (and the absolute value is used). A value of zero will mean no result pooling.

Range: {0, ..., ∞}

Default: 2

xslp_multistart_seed *(integer)*: Random seed used for the automatic generation of initial point when loading multistart presets ↩

Range: {-∞, ..., ∞}

Default: 0

xslp_multistart_threads *(integer)*: Maximum number of threads to be used in multistart ↩

The current hard upper limit on the number of threads to be sued in multistart is 64.

Range: {-1, ..., ∞}

Default: auto

xslp_mvTol *(real)*: Marginal value tolerance for determining if a constraint is slack ↩

If the absolute value of the marginal value of a constraint is less than xslp_mvTol, then (1) the constraint is regarded as not constraining for the purposes of the slack tolerance convergence criteria; (2) the constraint is not regarded as an active constraint when identifying unconverged variables in active constraints. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_objToPenaltyCost *(real)*: Factor to estimate initial penalty costs from objective function ↩

The setting of initial penalty error costs can affect the path of the optimization and, indeed, whether a solution is achieved at all. If the penalty costs are too low, then unbounded solutions may result although Xpress-SLP will increase the costs in an attempt to recover. If the penalty costs are too high, then the requirement to achieve feasibility of the linearized constraints may be too strong to allow the system to explore the nonlinear feasible region. Low penalty costs can result in many SLP iterations, as feasibility of the nonlinear constraints is not achieved until the penalty costs become high enough; high penalty costs force feasibility of the linearizations, and so tend to find local optima close to an initial feasible point. Xpress-SLP can analyze the problem to estimate the size of penalty costs required to avoid an initial unbounded solution. xslp_objToPenaltyCost can be used in conjunction with this procedure to scale the costs and give an appropriate initial value for balancing the requirements of feasibility and optimality. Not all models are amenable to the Xpress-SLP analysis. As the analysis is initially concerned with establishing a cost level to avoid unboundedness, a model which is sufficiently constrained will never show unboundedness regardless of the cost. Also, as the analysis is done at the start of the optimization to establish a penalty cost, significant changes in the coefficients, or a high degree of nonlinearity, may invalidate the initial analysis. A setting for xslp_objToPenaltyCost of zero disables the analysis. A setting of 3 or 4 has proved successful for many models. If xslp_objToPenaltyCost cannot be used because of the problem structure, its effect can still be emulated by some initial experiments to establish the cost required to avoid unboundedness, and then manually applying a suitable factor. If the problem is initially unbounded, then the penalty cost will be increased until either it reaches its maximum or the problem becomes bounded.

Range: [0, ∞]

Default: 0

xslp_oCount *(integer)*: Number of SLP iterations over which to measure objective function variation for static objective (2) convergence criterion ↩

The static objective (2) convergence criterion does not measure convergence of individual variables. Instead, it measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables interacting with active constraints (those that have a marginal value of at least xslp_mvTol) have converged. The rationale is that if the remaining unconverged variables are not involved in active constraints and if the objective function is not changing significantly between iterations, then the solution is more-or-less practical. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_oCount most recent SLP iterations and Obj is the

corresponding objective function value. If ABS(δObj) $\leq$ xslp_oTol_a then the problem has converged on the absolute static objective (2) convergence criterion. The static objective function (2) test is applied only if xslp_oCount is at least 2.

Range: {0, ..., ∞}

Default: 5

xslp_optimalityTolTarget *(real)*: When set, this defines a target optimality tolerance to which the linearizations are solved to ↩

This is a soft version of optimalityTol, and will dynamically revert back to optimalityTol if the desired accuracy could not be achieved.

Range: [0, ∞]

Default: not set

xslp_oTol_a *(real)*: Absolute static objective (2) convergence tolerance ↩

The static objective (2) convergence criterion does not measure convergence of individual variables. Instead, it measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables interacting with active constraints (those that have a marginal value of at least xslp_mvTol) have converged. The rationale is that if the remaining unconverged variables are not involved in active constraints and if the objective function is not changing significantly between iterations, then the solution is more-or-less practical. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_oCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ xslp_oTol_a then the problem has converged on the absolute static objective (2) convergence criterion. The static objective function (2) test is applied only if xslp_oCount is at least 2. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_oTol_r *(real)*: Relative static objective (2) convergence tolerance ↩

The static objective (2) convergence criterion does not measure convergence of individual variables. Instead, it measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables interacting with active constraints (those that have a marginal value of at least xslp_mvTol) have converged. The rationale is that if the remaining unconverged variables are not involved in active constraints and if the objective function is not changing significantly between iterations, then the solution is more-or-less practical. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_oCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ AVGIter(Obj)$*$xslp_oTol_r then the problem has converged on the relative static objective (2) convergence criterion. The static objective function (2) test is applied only if xslp_oCount is at least 2. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_penaltyInfoStart *(integer)*: Iteration from which to record row penalty information ↩

Information about the size (current and total) of active penalties of each row and the number of times a penalty vector has been active is recorded starting at the SLP iteration number given by xslp_penaltyInfoStart.

Range: {0, ..., ∞}

Default: 3

xslp_postsolve *(integer)*: Determines whether postsolving should be performed automatically ↩

Default: -1

value	meaning
-1	Postsolve if the problem could be solved to optimality/infeasibility.
0	Do not automatically postsolve.
1	Postsolve automatically.

xslp_presolve *(integer)*: Determines whether presolving should be performed prior to starting the main algorithm ↩

The Xpress NonLinear nonlinear presolve (which is carried out once, before augmentation) is independent of the Optimizer presolve (which is carried out during each SLP iteration).

Default: 1

value	meaning
0	Disable SLP presolve.
1	Activate SLP presolve.
2	Low memory presolve. Original problem is not restored by postsolve and dual solution may not be completely postsolved.

xslp_presolveLevel *(integer)*: Determines the level of changes presolve may carry out on the problem ↩

xslp_presolveOps controls the operations carried out in presolve. xslp_presolveLevel controls how those operations may change the problem.

Default: 4

value	meaning
1	Individual rows only presolve, no nonlinear transformations.
2	Individual rows and bounds only presolve, no nonlinear transformations.
3	Presolve allowing changing problem dimension, no nonlinear transformations.
4	Full presolve.

xslp_presolveOps *(integer)*: Bitmap indicating the SLP presolve actions to be taken ↩

The Xpress NonLinear nonlinear presolve (which is carried out once, before augmentation) is independent of the Optimizer presolve (which is carried out during each SLP iteration). Linear reductions are performed according to presolveOps if bit 12 is not set.

Setting single boolean options will overwrite the single bits of this bit map option.

Default: 2104

value	meaning
bit 0 = 1	Equivalent to xslp_presolveOps_general.
bit 1 = 2	Equivalent to xslp_presolveOps_fixZero.
bit 2 = 4	Equivalent to xslp_presolveOps_fixAll.
bit 3 = 8	Equivalent to xslp_presolveOps_setBounds.
bit 4 = 16	Equivalent to xslp_presolveOps_intBounds.
bit 5 = 32	Equivalent to xslp_presolveOps_domain.
bit 8 = 256	Equivalent to xslp_presolveOps_noCoefficients.

value	meaning
`bit 9 = 512`	Equivalent to xslp_presolveOps_noDeltas.
`bit 10 = 1024`	Equivalent to xslp_presolveOps_noDualSide.
`bit 11 = 2048`	Equivalent to xslp_presolveOps_eliminations.
`bit 12 = 4096`	Equivalent to xslp_presolveOps_noLinear.
`bit 13 = 8192`	Equivalent to xslp_presolveOps_noSimplifier.

xslp_presolveOps_domain *(boolean)*: Bound tightening based on function domains ↩

See also xslp_presolveOps.

Default: `1`

xslp_presolveOps_eliminations *(boolean)*: Allow eliminations on determined variables ↩

See also xslp_presolveOps.

Default: `1`

xslp_presolveOps_fixAll *(boolean)*: Explicitly fix all columns identified as fixed ↩

See also xslp_presolveOps.

Default: `0`

xslp_presolveOps_fixZero *(boolean)*: Explicitly fix columns identified as fixed to zero ↩

See also xslp_presolveOps.

Default: `0`

xslp_presolveOps_general *(boolean)*: Generic SLP presolve ↩

See also xslp_presolveOps.

Default: `0`

xslp_presolveOps_intBounds *(boolean)*: MISLP bound tightening ↩

See also xslp_presolveOps.

Default: `1`

xslp_presolveOps_noCoefficients *(boolean)*: Do not presolve coefficients ↩

See also xslp_presolveOps.

Default: `0`

xslp_presolveOps_noDeltas *(boolean)*: Do not remove delta variables ↩

See also xslp_presolveOps.

Default: `0`

xslp_presolveOps_noDualSide *(boolean)*: Avoid reductions that can not be dual postsolved ↩

See also xslp_presolveOps.

Default: `0`

xslp_presolveOps_noLinear *(boolean)*: Avoid performing linear reductions at the nlp level ↩

See also xslp_presolveOps.

Default: `0`

xslp_presolveOps_noSimplifier *(boolean)*: Avoid simplifying nonlinear expressions ↩

See also xslp_presolveOps.

Default: `0`

xslp_presolveOps_setBounds *(boolean)*: SLP bound tightening ↩

See also xslp_presolveOps.

Default: `1`

xslp_presolveZero *(real)*: Minimum absolute value for a variable which is identified as nonzero during SLP presolve ↩

During the SLP (nonlinear)presolve, a variable may be identified as being nonzero (for example, because it is used as a divisor). A bound of plus or minus xslp_presolveZero will be applied to the variable if it is identified as non-negative or non-positive.

Range: [0, ∞]

Default: `1e-09`

xslp_primalIntegralRef *(real)*: Reference solution value to take into account when calculating the primal integral ↩

When a global optimum is known, this can used to calculate a globally valid primal integral. It can also be used to indicate the target objective value still to be taken into account in the integral.

Range: [-∞, ∞]

Default: `1e+20`

xslp_probing *(integer)*: Determines whether probing on a subset of variables should be performed prior to starting the main algorithm ↩

Probing runs multiple times bound reduction in order to further tighten the bounding box.

The Xpress NonLinear nonlinear probing, which is carried out once, is independent of the Optimizer presolve (which is carried out during each SLP iteration). The probing level allows for probing on an expanding set of variables, allowing for probing on all variables (level 5) or only those for which probing is more likely to be useful (binary variables).

Default: `auto`

value	meaning
-1	Automatic.
0	Disable SLP probing.
1	Activate SLP probing only on binary variables.
2	Activate SLP probing only on binary or unbounded integer variables.
3	Activate SLP probing only on binary or integer variables.
4	Activate SLP probing only on binary, integer variables, and unbounded continuous variables.
5	Activate SLP probing on any variable.

xslp_sameCount *(integer)*: Number of steps reaching the step bound in the same direction before step bounds are increased ↩

If step bounding is enabled, the step bound for a variable will be increased if successive changes are in the same direction. More precisely, if there are xslp_sameCount successive changes reaching the step bound and in the same direction for a variable, then the step bound (B) for the variable will be reset to B∗xslp_expand.

Range: {0, ..., ∞}

Default: 3

xslp_sameDamp *(integer)*: Number of steps in same direction before damping factor is increased ↩

If dynamic damping is enabled, the damping factor for a variable will be increased if successive changes are in the same direction. More precisely, if there are xslp_sameDamp successive changes in the same direction for a variable, then the damping factor (D) for the variable will be reset to D∗xslp_dampExpand + xslp_dampMax∗(1-xslp_dampExpand)

Range: {0, ..., ∞}

Default: 3

xslp_sbStart *(integer)*: SLP iteration after which step bounds are first applied ↩

If step bounds are used, they can be applied for the whole of the SLP optimization process, or started after a number of SLP iterations. In general, it is better not to apply step bounds from the start unless one of the following applies: (1) the initial estimates are known to be good, and explicit values can be provided for initial step bounds on all variables; or (2) the problem is unbounded unless all variables are step-bounded.

Range: {0, ..., ∞}

Default: 8

xslp_scale *(integer)*: When to re-scale the SLP problem ↩

During the SLP optimization, matrix entries can change considerably in magnitude, even when the formulae in the coefficients are not very nonlinear. Re-scaling of the matrix can reduce numerical errors, but may increase the time taken to achieve convergence.

Default: 1

value	meaning
0	No re-scaling.
1	Re-scale every SLP iteration up to xslp_scaleCount iterations after the end of barrier optimization.
2	Re-scale every SLP iteration up to xslp_scaleCount iterations in total.
3	Re-scale every SLP iteration until primal simplex is automatically invoked.
4	Re-scale every SLP iteration

xslp_scaleCount *(integer)*: Iteration limit used in determining when to re-scale the SLP matrix ↩

If xslp_scale is set to 1 or 2, then xslp_scaleCount determines the number of iterations (after the end of barrier optimization or in total) in which the matrix is automatically re-scaled.

Range: {0, ..., ∞}

Default: 0

xslp_shrink *(real)*: Multiplier to reduce a step bound ↩

If step bounding is enabled, the step bound for a variable will be decreased if successive changes are in opposite directions. The step bound (B) for the variable will be reset to B∗xslp_shrink. If the step bound is already below the strict (delta or closure) tolerances, it will not be reduced further.

Range: [0, 1]

Default: 0.5

xslp_shrinkBias *(real)*: Defines an overwrite / adjustment of step bounds for improving iterations ↩

Positive values overwrite xslp_shrink only if the objective is improving. A negative value is used to scale all step bounds in improving iterations.

Range: [-∞, ∞]

Default: not set

xslp_slpLog *(integer)*: Frequency with which SLP status is printed ↩

If xslp_log is set to zero (minimal logging) then a nonzero value for xslp_slpLog defines the frequency (in SLP iterations) when summary information is printed out.

Range: {0, ..., ∞}

Default: 1

xslp_solver *(integer)*: Selects the library to use for local solves ↩

The presence of KNITRO is detected automatically. KNITRO can be used to solve any problem loaded into XSLP, independently from how the problem was loaded. xslp_solver is set to automatic, Xpress SLP will be selected if any SLP specific construct has been loaded (these are ignored if KNITRO is selected manually). When solving problems to global optimality, the xslp_solver control is used to decide which local solver to call for reoptimizing NLP- infeasible solutions heuristically.

Default: auto

value	meaning
-1	Automatic selection, based on model characteristics and solver availability
0	Use Xpress-SLP (always available)
1	Use Knitro if available
2	Use Xpress-Optimizer if possible (convex quadratic problems only)

xslp_sTol_a *(real)*: Absolute slack convergence tolerance ↩

The slack convergence criterion is identical to the impact convergence criterion, except that the tolerances used are xslp_sTol_a (instead of xslp_iTol_a) and xslp_sTol_r (instead of xslp_iTol_r). See xslp_iTol_a for a description of the test. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_sTol_r *(real)*: Relative slack convergence tolerance ↩

The slack convergence criterion is identical to the impact convergence criterion, except that the tolerances used are xslp_sTol_a (instead of xslp_iTol_a) and xslp_sTol_r (instead of xslp_iTol_r). See xslp_iTol_r for a description of the test. When the value is set to be negative, the value is adjusted automatically by SLP, based on the feasibility target xslp_validationTarget_r. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_stopOutOfRange *(boolean)*: Stop optimization and return error code if internal function argument is out of range ↩

If xslp_stopOutOfRange is set to 1, then if an internal function receives an argument which is out of its allowable range (for example, LOG of a negative number), an error code is set and the optimization is terminated.

Default: `0`

xslp_threads *(integer)*: Default number of threads to be used ↩

Overall thread control value, used to determine the number of threads used where parallel calculations are possible.

Range: {`-1`, ..., ∞}

Default: `auto`

xslp_unfinishedLimit *(integer)*: Number of consecutive SLP iterations that may have an unfinished status before the solve is terminated ↩

If the optimization of the current linear approximation terminates with an "unfinished" status, then first a number of strategies are applied to attempt a successful solve of the same linearization. If this fails, then a new iteration is started to change the linearization itself. This control limits the numner of such repeated attempts.

Range: {`0`, ..., ∞}

Default: `3`

xslp_validationTarget_k *(real)*: Optimality target tolerance ↩

Primary optimality control for SLP. When the relevant optimality based convergence controls are left at their default values, SLP will adjust their value to match the target. The control defines a target value, that may not necessarily be attainable for problem with no strong constraint qualifications.

Range: [0, ∞]

Default: `1e-06`

xslp_validationTarget_r *(real)*: Feasiblity target tolerance ↩

Primary feasiblity control for SLP. When the relevant feasibility based convergence controls are left at their default values, SLP will adjust their value to match the target. The control defines a target value, that may not necessarily be attainable.

Range: [0, ∞]

Default: `1e-06`

xslp_vCount *(integer)*: Number of SLP iterations over which to measure static objective (3) convergence ↩

The static objective (3) convergence criterion does not measure convergence of individual variables, and in fact does not in any way imply that the solution has converged. However, it is sometimes useful to be able to terminate an optimization once the objective function appears to have stabilized. One example is where a set of possible schedules are being evaluated and initially only a good estimate of the likely objective function value is required, to eliminate the worst candidates. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_vCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ xslp_vTol_a then the problem has converged on the absolute static objective function (3) criterion. The static objective function (3) test is applied only if after at least xslp_vLimit + xslp_sbStart SLP iterations have taken place and only if xslp_vCount is at least 2. Where step bounding is being used, this ensures that the test is not applied until after step bounding has been introduced.

Range: {0, ..., ∞}

Default: `0`

xslp_vLimit *(integer)*: Number of SLP iterations after which static objective (3) convergence testing starts ↩

The static objective (3) convergence criterion does not measure convergence of individual variables, and in fact does not in any way imply that the solution has converged. However, it is sometimes useful to be able to terminate an optimization once the objective function appears to have stabilized. One example is where a set of possible schedules are being evaluated and initially only a good estimate of the likely objective function value is required, to eliminate the worst candidates. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_vCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ xslp_vTol_a then the problem has converged on the absolute static objective function (3) criterion. The static objective function (3) test is applied only if after at least xslp_vLimit + xslp_sbStart SLP iterations have taken place and only if xslp_vCount is at least 2. Where step bounding is being used, this ensures that the test is not applied until after step bounding has been introduced.

Range: {0, ..., ∞}

Default: `0`

xslp_vTol_a *(real)*: Absolute static objective (3) convergence tolerance ↩

The static objective (3) convergence criterion does not measure convergence of individual variables, and in fact does not in any way imply that the solution has converged. However, it is sometimes useful to be able to terminate an optimization once the objective function appears to have stabilized. One example is where a set of possible schedules are being evaluated and initially only a good estimate of the likely objective function value is required, to eliminate the worst candidates. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_vCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ xslp_vTol_a then the problem has converged on the absolute static objective function (3) criterion. The static objective function (3) test is applied only if after at least xslp_vLimit + xslp_sbStart SLP iterations have taken place and only if xslp_vCount is at least 2. Where step bounding is being used, this ensures that the test is not applied until after step bounding has been introduced. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_vTol_r *(real)*: Relative static objective (3) convergence tolerance ↩

The static objective (3) convergence criterion does not measure convergence of individual variables, and in fact does not in any way imply that the solution has converged. However, it is sometimes useful to be able to terminate an optimization once the objective function appears to have stabilized. One example is where a set of possible schedules are being evaluated and initially only a good estimate of the likely objective function value is required, to eliminate the worst candidates. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_vCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ AVGIter(Obj) $*$ xslp_vTol_r then the problem has converged on the absolute static objective function (3) criterion. The static objective function (3) test is applied only if after at least xslp_vLimit

- xslp_sbStart SLP iterations have taken place and only if xslp_vCount is at least 2. Where step bounding is being used, this ensures that the test is not applied until after step bounding has been introduced. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_wCount *(integer)*: Number of SLP iterations over which to measure the objective for the extended convergence continuation criterion ↩

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. The extended convergence continuation criterion is applied after a converged solution has been found where at least one variable has converged on extended criteria and is at its step bound limit. The extended convergence continuation test measures whether any improvement is being achieved when additional SLP iterations are carried out. If not, then the last converged solution will be restored and the optimization will stop. For a maximization problem, the improvement in the objective function at the current iteration compared to the objective function at the last

converged solution is given by: δObj = Obj - LastConvergedObj For a minimization problem, the sign is reversed. If δObj > xslp_wTol_a and δObj > ABS(ConvergedObj) * xslp_wTol_r then the solution is deemed to have a significantly better objective function value than the converged solution. When a solution is found which converges on extended criteria and with active step bounds, the solution is saved and SLP optimization continues until one of the following: (1) a new solution is found which converges on some other criterion, in which case the SLP optimization stops with this new solution; (2) a new solution is found which converges on extended criteria and with active step bounds, and which has a significantly better objective function, in which case this is taken as the new saved solution; (3) none of the xslp_wCount most recent SLP iterations has a significantly better objective function than the saved solution, in which case the saved solution is restored and the SLP optimization stops. If xslp_wCount is zero, then the extended convergence continuation criterion is disabled.

Range: {0, ..., ∞}

Default: 0

xslp_wTol_a *(real)*: Absolute extended convergence continuation tolerance ↩

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. The extended convergence continuation criterion is applied after a converged solution has been found where at least one variable has converged on extended criteria and is at its step bound limit. The extended convergence continuation test measures whether any improvement is being achieved when additional SLP iterations are carried out. If not, then the last converged solution will be restored and the optimization will stop. For a maximization problem, the improvement in the objective function at the current iteration compared to the objective function at the last converged solution is given by: δObj = Obj - LastConvergedObj For a minimization problem, the sign is reversed. If δObj > xslp_wTol_a and δObj > ABS(ConvergedObj) * xslp_wTol_r then the solution is deemed to have a significantly better objective function value than the converged solution. When a solution is found which converges on extended criteria and with active step bounds, the solution is saved and SLP optimization continues until one of the following: (1) a new solution is found which converges on some other criterion, in which case the SLP optimization stops with this new solution; (2) a new solution is found which converges on extended criteria and with active step bounds, and which has a significantly better objective function, in which case this is taken as the new saved solution; (3) none of the xslp_wCount most recent SLP iterations has a significantly better objective function than the saved solution, in which case the saved solution is restored and the SLP optimization stops. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_wTol_r *(real)*: Relative extended convergence continuation tolerance ↩

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. The extended convergence continuation criterion is applied after a converged solution has been found where at least one variable has converged on extended criteria and is at its step bound limit. The extended convergence continuation test measures whether any improvement is being achieved when additional SLP iterations are carried out. If not, then the last converged solution

will be restored and the optimization will stop. For a maximization problem, the improvement in the objective function at the current iteration compared to the objective function at the last converged solution is given by: δObj = Obj - LastConvergedObj For a minimization problem, the sign is reversed. If δObj > xslp_wTol_a and δObj > ABS(ConvergedObj) * xslp_wTol_r then the solution is deemed to have a significantly better objective function value than the converged solution. If xslp_wCount is greater than zero, and a solution is found which converges on extended criteria and with active step bounds, the solution is saved and SLP optimization continues until one of the following: (1) a new solution is found which converges on some other criterion, in which case the SLP optimization stops with this new solution; (2) a new solution is found which converges on extended criteria and with active step bounds, and which has a significantly better objective function, in which case this is taken as the new saved solution; (3) none of the xslp_wCount most recent SLP iterations has a significantly better objective function than the saved solution, in which case the saved solution is restored and the SLP optimization stops. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-4 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_xCount *(integer)*: Number of SLP iterations over which to measure static objective (1) convergence ↩

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. However, if the objective function has already been stable for several SLP iterations, then there is less likelihood of an improved result, and the converged solution can be accepted. The static objective function (1) test measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. Because all the variables have converged, the solution is already converged but the fact that some variables are at their step bound limit suggests that the objective function could be improved by going further. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_xCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ xslp_xTol_a then the objective function is deemed to be static according to the absolute static objective function (1) criterion. If ABS(δObj) $\leq$ AVGIter(Obj) * xslp_xTol_r then the objective function is deemed to be static according to the relative static objective function (1) criterion. The static objective function (1) test is applied only until xslp_xLimit SLP iterations have taken place. After that, if all the variables have converged on strict or extended criteria, the solution is deemed to have converged. If the objective function passes the relative or absolute static objective function (1) test then the solution is deemed to have converged.

Range: {0, ..., ∞}

Default: `5`

xslp_xLimit *(integer)*: Number of SLP iterations up to which static objective (1) convergence testing starts ↩

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration.

However, if the objective function has already been stable for several SLP iterations, then there is less likelihood of an improved result, and the converged solution can be accepted. The static objective function (1) test measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. Because all the variables have converged, the solution is already converged but the fact that some variables are at their step bound limit suggests that the objective function could be improved by going further. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_xCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ xslp_xTol_a then the objective function is deemed to be static according to the absolute static objective function (1) criterion. If ABS(δObj) $\leq$ AVGIter(Obj) $*$ xslp_xTol_r then the objective function is deemed to be static according to the relative static objective function (1) criterion. The static objective function (1) test is applied only until xslp_xLimit SLP iterations have taken place. After that, if all the variables have converged on strict or extended criteria, the solution is deemed to have converged. If the objective function passes the relative or absolute static objective function (1) test then the solution is deemed to have converged.

Range: {0, ..., ∞}

Default: 100

xslp_xTol_a *(real)*: Absolute static objective function (1) tolerance ↩

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it is still possible that an improved result could be obtained by taking another SLP iteration. However, if the objective function has already been stable for several SLP iterations, then there is less likelihood of an improved result, and the converged solution can be accepted. The static objective function (1) test measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. Because all the variables have converged, the solution is already converged but the fact that some variables are at their step bound limit suggests that the objective function could be improved by going further. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_xCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ xslp_xTol_a then the objective function is deemed to be static according to the absolute static objective function (1) criterion. If ABS(δObj) $\leq$ AVGIter(Obj) $*$ xslp_xTol_r then the objective function is deemed to be static according to the relative static objective function (1) criterion. The static objective function (1) test is applied only until xslp_xLimit SLP iterations have taken place. After that, if all the variables have converged on strict or extended criteria, the solution is deemed to have converged. If the objective function passes the relative or absolute static objective function (1) test then the solution is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-3 and 1e-6.

Range: [-∞, ∞]

Default: auto

xslp_xTol_r *(real)*: Relative static objective function (1) tolerance ↩

It may happen that all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. This means that, at least in the linearization, if the variable were to be allowed to move further the objective function would improve. This does not necessarily imply that the same is true of the original problem, but it

is still possible that an improved result could be obtained by taking another SLP iteration. However, if the objective function has already been stable for several SLP iterations, then there is less likelihood of an improved result, and the converged solution can be accepted. The static objective function (1) test measures the significance of the changes in the objective function over recent SLP iterations. It is applied when all the variables have converged, but some have converged on extended criteria and at least one of these variables is at its step bound. Because all the variables have converged, the solution is already converged but the fact that some variables are at their step bound limit suggests that the objective function could be improved by going further. The variation in the objective function is defined as δObj = MAXIter(Obj) - MINIter(Obj) where Iter is the xslp_xCount most recent SLP iterations and Obj is the corresponding objective function value. If ABS(δObj) $\leq$ xslp_xTol_a then the objective function is deemed to be static according to the absolute static objective function (1) criterion. If ABS(δObj) $\leq$ AVGIter(Obj) $*$ xslp_xTol_r then the objective function is deemed to be static according to the relative static objective function (1) criterion. The static objective function (1) test is applied only until xslp_xLimit SLP iterations have taken place. After that, if all the variables have converged on strict or extended criteria, the solution is deemed to have converged. If the objective function passes the relative or absolute static objective function (1) test then the solution is deemed to have converged. When the value is set to be negative, the value is adjusted automatically by SLP, based on the optimality target xslp_validationTarget_k. Good values for the control are usually fall between 1e-4 and 1e-6.

Range: [-∞, ∞]

Default: `auto`

xslp_zero *(real)*: Absolute tolerance ↩

If a value is below xslp_zero in magnitude, then it will be regarded as zero in certain formula calculations: an attempt to divide by such a value will give a "divide by zero" error; an exponent of a negative number will produce a "negative number, fractional exponent" error if the exponent differs from an integer by more than xslp_zero.

Range: [0, ∞]

Default: `1e-15`

xslp_zeroCriterion *(integer)*: Bitmap determining the behavior of the placeholder deletion procedure ↩

Setting single boolean options will overwrite the single bits of this bit map option.

Default: `0`

value	meaning
`bit 0 = 1`	Equivalent to xslp_zeroCriterion_nbSLPVar.
`bit 1 = 2`	Equivalent to xslp_zeroCriterion_nbDelta.
`bit 2 = 4`	Equivalent to xslp_zeroCriterion_slpVarNBUpdateRow.
`bit 3 = 8`	Equivalent to xslp_zeroCriterion_deltaNBUpdateRow.
`bit 4 = 16`	Equivalent to xslp_zeroCriterion_deltaNBDRRow.
`bit 5 = 32`	Equivalent to xslp_zeroCriterion_print.

xslp_zeroCriterionCount *(integer)*: Number of consecutive times a placeholder entry is zero before being considered for deletion ↩

Range: {0, ..., ∞}

Default: `0`

xslp_zeroCriterionStart *(integer)*: SLP iteration at which criteria for deletion of placeholder entries are first activated ↩

Range: {0, ..., ∞}

Default: 0

xslp_zeroCriterion_deltaNBDRRow *(boolean)*: Remove placeholders in a basic delta variable if the determining row for the corresponding SLP variable is nonbasic ↩

See also xslp_zeroCriterion.

Default: 0

xslp_zeroCriterion_deltaNBUpdateRow *(boolean)*: Remove placeholders in a basic delta variable if its update row is nonbasic and the corresponding SLP variable is nonbasic ↩

See also xslp_zeroCriterion.

Default: 0

xslp_zeroCriterion_nbDelta *(boolean)*: Remove placeholders in nonbasic delta variables ↩

See also xslp_zeroCriterion.

Default: 0

xslp_zeroCriterion_nbSLPVar *(boolean)*: Remove placeholders in nonbasic SLP variables ↩

See also xslp_zeroCriterion.

Default: 0

xslp_zeroCriterion_print *(boolean)*: Print information about zero placeholders ↩

See also xslp_zeroCriterion.

Default: 0

xslp_zeroCriterion_slpVarNBUpdateRow *(boolean)*: Remove placeholders in a basic SLP variable if its update row is nonbasic ↩

See also xslp_zeroCriterion.

Default: 0

5.38.5 Helpful Hints

The comments below should help both novice and experienced GAMS users to better understand and make use of GAMS/XPRESS.

- **Infeasible and unbounded models** The fact that a model is infeasible/unbounded can be detected at two stages: during the presolve and during the simplex or barrier algorithm. In the first case we cannot recover a solution, nor is any information regarding the infeasible/unbounded constraint or variable provided (at least in a way that can be returned to GAMS). In such a situation, the GAMS link will automatically rerun the model using primal simplex with presolve turned off (this can be avoided by setting the rerun option to 0). It is possible (but very unlikely) that the simplex method will solve a model to optimality while the presolve claims the model is infeasible/unbounded (due to feasibility tolerances in the simplex and barrier algorithms).
- The barrier method does not make use of `iterlim`. Use `bariterlim` in an options file instead. The number of barrier iterations is echoed to the log and listing file. If the barrier iteration limit is reached during the barrier algorithm, XPRESS continues with a simplex algorithm, which will obey the iterlim setting.
- Semi-integer variables are not implemented in the link, nor are they supported by XPRESS; if present, they trigger an error message.
- SOS1 and SOS2 variables are required by XPRESS to have lower bounds of 0 and nonnegative upper bounds.

5.38.6 Setting up a GAMS/XPRESS-Link license

To use the GAMS/XPRESS solver with a GAMS/XPRESS-Link license you have to set up the XPRESS portion of the licensing. To do so, copy your XPRESS license `xpauth.xpr` to the GAMS system directory. As of version 24.2 GAMS already comes with a file `xpauth.xpr` . You might consider copying this file to `xpauth.xpr.bak` or similar before overwriting it with your own XPRESS license file.

Chapter 6

Tools Manuals

A large number of tools are included in GAMS distribution. Below are a functional categorization of all tools, an alphabetically sorted list of all tools, and a brief description of each tool with their Supported Platforms.

Note

> Traditionally, GAMS tools consisted of a collection of *executables* with a file (mostly GDX) interface. These tool executables will be replaced over time by GAMS Connect agents and a collection of tools in a new GAMS tools library. For some time both ways will be supported but the executable tools will go away, so when in doubt what tool to pick, select Connect or a GAMS tool.

6.1 Tools Category

All tools included in GAMS distribution are categorized as

6.1.1 GAMS Integrated Development Environments

There are two integrated model development environments including a general text editor with the ability to launch and monitor the compilation and execution of GAMS models: GAMS Studio and the `GAMS IDE` (deprecated).

6.1.2 GAMS Tools Library

In the GAMS Tools Library, various tools are collected to provide an easy access to complex task. The tools in this library complement facilities of GAMS Connect and tools available as executables (e.g. GDXXRW). Currently the library includes the following tool categories:

6.1.2.1 Algorithmic tools (alg)

Provide algorithmic functionality like sorting data: Rank.

6.1.2.2 Data tools (data)

Provide access to external data sources: ExcelDump.

6.1.2.3 GDX Service (gdxservice)

Provide functionality for GDX file manipulation: GDXEncoding and GDXRename.

6.1.2.4 Linear Algebra (linalg)

Provide functionality for computational linear algebra: Cholesky, Eigenvalue, Eigenvector, Invert and Ordinary Least Squares (OLS).

6.1.2.5 Windows Only Tools (win32)

Provide functionality specific to the Windows operating system: ShellExecute, MSAppAvail, ExcelTalk and ExcelMerge.

There are a couple of things to note:

- A tool can be invoked directly in the model code during compile ($callTool) or execution time (executeTool) or by using the standalone command line utility gamstool.
- When invoked from the model code the exchange of GAMS symbols is done in memory. If a tool is called from the command line, this is not possible and hence for some tools that deal with GAMS symbols it is necessary to specify GDX input and/or output files. Even for use inside GAMS models the GDX file interface can be useful (e.g. for debugging) hence the specification of GDX input/output files is optional in that case.
- Similar to the $call (compile time) and execute (execution time) commands, a GAMS tool returns a shell code that can be checked via `errorLevel`. In case one expects the tools to perform without error, it is recommended to add the suffix .checkErrorLevel. This will stop the entire execution of GAMS if an error occurs while executing the tool.
- If a tool of the GAMS Tool library is used at execution time to populate a GAMS symbol with data for the first time, the compiler does not know the outcome of the tool. As a result, the compiler does not have information about any symbols that might be filled with data, and therefore cannot reliably define these symbols at compile time. Code that later on references a corresponding symbol would result in a compilation error 141: `Symbol declared but no values have been assigned`. In order to be able to load symbols implicitly during execution time, the dollar control option $onImplicitAssign needs to be set. There are other methods to convince the compiler that the symbols has been defined, e.g. `execute_load$0 symName;` which is otherwise a no-op.
- The compile time variant of a tool ($callTool [...]) is ignored while $onExternalInput is active and IDCGDXInput is set.
- Tool arguments: All tools follow the same logic to process arguments. The list of arguments starts with a number of *positional* arguments followed by *named* argument `[-]name=val`.
- For each tool there is a short and a long help available. If a tool is called without arguments or with `-h` , a short description is displayed, e.g. `gamstool linalg.eigenvector -h`. With the argument `--help` a detailed help text appears: `gamstool linalg.eigenvector --help`. If only the category and not a contained tool is requested, the help text of all included tools will be displayed: `gamstool linalg` or `gamstool linalg --help`.

For more information please refer to the individual tool manuals.

6.1.2.6 Command Line Utility gamstool

The GAMS system directory contains the utility `gamstool` to run GAMS Tool instructions directly from the command line. On Windows the utility has the callable extension `.cmd` which does not need to part of the command because the shell automatically checks for the extension. This script wraps the Python script `tooldriver.py` by calling the Python interpreter that ships with GAMS. `gamstool` is called like this:

```
gamstool [toolCategory.]toolName positionalArguments [namedArguments]
```

6.1.3 Data Exchange

A collection of tools that provide functionality to exchange data between GAMS and other data sources. This category contains tools for popular data sources and high-level programming environment and like databases (GDX2ACCESS, GDX2SQLITE, MDB2GMS, SQL2GMS), Matlab (GDXMRW), and R (GDXRRW). There are also tools for specialized systems like VEDA (GDX2VEDA).

6.1.3.1 Excel

A collection of tools that provide functionality to exchange data between GAMS and Excel. The tools in this category are GDX2XLS, GDXXRW, XLS2GMS, and ExcelDump. Many of the tools described here use the GAMS Data eXchange facility GAMS Data eXchange (GDX). Note that the executable tools in this category will be or have been replaced over time by GAMS GAMS Connect agents and tools from the GAMS Tools Library.

6.1.4 GDX Service

A collection of tools that operate directly on GAMS Data eXchange (GDX) containers to e.g. compare (GDXDIFF), copy (GDXCOPY), merge (GDXMERGE), label rename (GDXRename) and encoding (GDXEncoding).

6.1.5 Data Transformation

A collection of tools that perform very specific tasks that are awkward or inefficient to implement in GAMS directly. The tools in this category are MessageReceiverWindow, Rank, SCENRED, SCENRED2 and all tools from Linear Algebra (linalg).

6.1.6 Other Tools

A collection of more exotic tools that can become handy in some some special circumstances. The tools in this category are ASK, ENDECRYPT, FINDTHISGAMS, GAMS Posix Utilities, MODEL2TEX, and all tools from Windows Only Tools (win32). Most notably, the collection contains the tool MODEL2TEX to document the model algebra in LaTeX format.

6.2 List of Tools

The following table gives an alphabetically sorted list of all available tools.

Tool	Description
ASK	The utility can be used to get input from an user interactively.
[LINALG.]CHOLESKY	Calculates the cholesky decomposition of a symmetric positive definite matrix.
CSV2GDX	Reads a CSV file (comma separated values) and writes to a GDX file.
[LINALG.]EIGENVALUE	Calculates the Eigenvalues of a symmetric positive definite matrix.
[LINALG.]EIGENVECTOR	Calculates the Eigenvalues and Eigenvectors of a symmetric positive definite matrix.
ENDECRYPT	A tool to encrypt and decrypt text files.
[DATA.]EXCELDUMP	Writes all worksheets of an Excel workbook to GAMS symbols.
[WIN32.]EXCELMERGE	Merges the sheets of the source Excel workbook into the destination workbook.
[WIN32.]EXCELTALK	Performs command on an Excel workbook specified by filename.
FINDTHISGAMS	Windows command line tool for modifying GAMS specific registry entries created by the GAMS installer.
GAMS IDE	Classic Integrated Development Environment.
GAMS STUDIO	Integrated Development Environment.
GDX2ACCESS	Converts GDX data to MS Access tables.
GDX2SQLITE	Dumps the complete contents of a GDX file into a SQLite2 database. From Amsterdam Optimization Modeling Group.
GDX2VEDA	Translates a GDX file into the VEDA format.
GDX2XLS	Converts GDX data into a MS Excel spreadsheet.
GDXCOPY	Converts a GDX file into different GDX formats.
GDXDIFF	Compares the data of symbols with the same name, type and dimension in two GDX files and writes the differences to a third GDX file.
GDXDUMP	Writes scalars, sets and parameters (tables) to standard output formatted as a GAMS program with data statements.
[GDXSERVICE.]GDXENCODING	Label encoding conversion.
GDXMERGE	Combines multiple GDX files into one file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.
GDXMRW	A suite of utilities to import/export data between GAMS and MATLAB and to call GAMS models from MATLAB and get results back into MATLAB.
[GDXSERVICE.]GDXRENAME	Renames labels in a GDX file.
GDXRRW	An interface between GAMS and R. It includes functions to transfer data between GDX and R and a function to call GAMS from R.
GDXVIEWER	Views and converts data contained in GDX files.
GDXXRW	Preferred utility to read and write MS Excel spreadsheet data.
GMSUNZIP	Decompression tool unzip with Debian patches, but renamed to "gmsunzip".
GMSZIP	Compression and archiving tool zip with Debian patches, but renamed to "gmszip".
IDECMDS	Sends commands to the GAMSIDE.
[LINALG.]INVERT	Calculates the inverse of a square matrix A.
MDB2GMS	Converts data from an MS Access database into a GAMS readable format.

Tool	Description
MESSAGE RECEIVER WINDOW	A graphical tool that receives and displays Windows messages.
MODEL2TEX	Translates a GAMS model into LaTeX
MPS2GMS	Translates an MPS or LP file into an equivalent short generic GAMS program using a GDX file to store data.
[WIN32.]MSAPPAVAIL	Checks if a MS Office Application is available.
[LINALG.]OLS	Ordinary Least Squares: Estimates the unknown parameters in a linear regression model.
POSIX	A collection of `POSIX` utilities which are usually available for Windows and the different Unix systems and therefore help to write platform independent scripts.
[ALG.]RANK	Ranks one-dimensional numeric data.
SCENRED	A tool for the reduction of scenarios that model random data processes of a stochastic program. From Humboldt-University Berlin.
SCENRED2	Scenred2 is a fundamental update of Scenred and offers a scenario tree construction algorithm. From Humboldt-University Berlin.
[WIN32.]SHELLEXECUTE	Spawns an external program.
SQL2GMS	Converts data from an SQL database into a GAMS readable format.
XLS2GMS	Converts spreadsheet data from a MS Excel spreadsheet into a GAMS readable format.
XLSDUMP	Writes all worksheets of a MS Excel workbook to a GDX file. Unlike gdxxrw, the program does not require that Excel is installed.

6.3 Supported Platforms

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
ASK	✓*			
[LINALG.]CHOLESKY	✓	✓	✓	✓
CSV2GDX	✓	✓	✓	✓
[LINALG.]EIGENVALUE	✓	✓	✓	✓
[LINALG.]EIGENVECTOR	✓	✓	✓	✓
ENDECRYPT	✓	✓	✓	✓
[DATA.]EXCELDUMP	✓	✓	✓	✓
[WIN32.]EXCELMERGE	✓			
[WIN32.]EXCELTALK	✓			
FINDTHISGAMS	✓			
GAMSIDE	✓*			
GAMSSTUDIO	✓	✓	✓	✓
GDX2ACCESS	✓*			
GDX2SQLITE	✓	✓	✓	✓
GDX2VEDA	✓	✓	✓	✓
GDX2XLS	✓*			
GDXCOPY	✓	✓	✓	✓
GDXDIFF	✓	✓	✓	✓
GDXDUMP	✓	✓	✓	✓

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
[GDXSERVICE.]GDXENCODING	✓	✓	✓	✓
GDXMERGE	✓	✓	✓	✓
GDXMRW	✓	✓	✓	
[GDXSERVICE.]GDXRENAME	✓	✓	✓	✓
GDXRRW	✓	✓	✓	
GDXVIEWER	✓∗			
GDXXRW	✓∗			
GMSUNZIP	✓	✓	✓	✓
GMSZIP	✓	✓	✓	✓
IDECMDS	✓∗			
[LINALG.]INVERT	✓	✓	✓	✓
MDB2GMS	✓∗			
MSGRWIN	✓			
MODEL2TEX	✓	✓	✓	✓
MPS2GMS	✓	✓	✓	✓
[WIN32.]MSAPPAVAIL	✓			
[LINALG.]OLS	✓	✓	✓	✓
POSIX∗∗	✓∗			
[ALG.]RANK	✓	✓	✓	✓
SCENRED	✓	✓	✓	✓
SCENRED2	✓	✓	✓	✓
[WIN32.]SHELLEXECUTE	✓			
SQL2GMS	✓∗			
XLS2GMS	✓∗			
XLSDUMP	✓∗			

∗ Note that the tool is 32 bit but runs fine on 64 bit Windows.
∗∗ awk, cat, cksum, cmp, comm, cp, cut, diff, expr, fold, gdate, grep, gsort, gunzip, gzip, head, join, make, mkdir, mv, od, paste, printf, rm, sed, sleep, tail, tar, tee, test, touch, tr, uniq, wc, xargs

6.3.1 LibInclude Tools Library

In the LibInclude Tools Library, various tools are collected to provide an easy access to complex task. The tools are located in the `inclib` folder in the GAMS system directory and can be invoked using the $libInclude command.

Note

The default library include directory `inclib` can be changed with the libIncDir command line parameter. Hence, make sure you point to the correct directory when using `$libInclude`.

Usage:

```
$libInclude <library_file> [<tool_name>] [<option(s)>]
```

Currently the library includes the following tools:

Libinclude file	Description
moo	Provides methods for multi-objective optimization in GAMS.
pyEmbMI	Provide access to a model instance that can be modified and resolved without regenerating the model over and over.
rank	Routine for ranking one-dimensional numeric data. Unlike the GAMS tool alg.rank, rank can handle percentile levels.

6.4 ASK

Writing a GUI (Graphical User Interface) for a GAMS application requires some serious programming, and therefore place a burden on the knowledge and time of the modeler. In this section we show how extremely simple user interfaces can be built using a few simple tools. The main purpose of these tools is to allow a developer quickly put an application together such that an end-user does not have to edit GAMS files. We assume the end-user runs a GAMS model from the GAMS-IDE.

The ASK utility can be used to get input from an end-user and the GDXViewer tool can be used to present end-results. Together these tools allow you to build a minimalist GUI without any programming.

6.4.1 Usage

```
ask <options>
```

where the **options** are

`T=string`

where the `string` identifies the type of input item to go after and can be

- `integer` - when one wants an integer number
- `float` - when one wants a real number
- `radiobutton` - when one wants a radio button choice
- `combobox` - when one wants a combo (drop down choice) box
- `checklistbox` - when one wants a check list box
- `fileopenbox` - when one wants the name of a file to open
- `filesavebox` - when one wants the name of a file to save

Example: `T=integer`

`M="string"`

where the `string` is the text to in the box.
Example: `M="Enter a number"`

`O="filename"`

where the `filename` is the name of a file in which to place the results for subsequent inclusion into GAMS
Example: `O="file.inc"`

`D="string 1|string 2..."`

where the "string 1|string 2|string 3|...|string n" gives the n strings to be associated with multiple choices when using checkbox, radiobutton, combobox, or checklistbox. The individual strings are separated by the delimiter "|"
Example: `D="Small data set|Medium data set|Large data set"`

`E="number 1|number 2..."`

where the "number 1|number 2|number 3|...|number n" gives the n numbers to be returned to GAMS associated with the choices made when using checkbox, radiobutton, combobox, or checklistbox. The individual numbers are separated by the delimiter "|"
Example: `E="1|2|3|4|5"`

`I="filepath"`

where `filepath` gives the path in which to look for the file under the fileopenbox and filesavebox dialogues. If not specified this is the project directory
Example: I="C:\gams\mine"

`F="filemask"`

where `filemask` gives the mask for acceptable files under the fileopenbox and filesavebox dialogues. If not specified this is ∗.∗.
Example: `I="*.gdx"`

`R="string"`

where the `string` gives a line of GAMS code to place in the include file.
This can contain a s parameter in which the information to return is substituted
Example: R="$include '%s'" or R="set i /1990∗%s/;"

`C="string"`

A title for the dialogue box being used
Example: `C="Box to ask for a file"`

`L=number`

where the `number` gives a lower bound on a numeric entry
Example: `L=15`

`U=number`

where the `number` gives an upper bound on a numeric entry
Example: `U=15`

@"filename"

where `filename` gives the name of a file of input instructions containing the options above in this table
Example: `@ask.opt`

In addition a number by itself can be entered to put multiple entries into columns under the `checkbox`, `radiobutton`, `combobox`, or `checklistbox` entries.

6.4.2 Calling ASK utility from GAMS

The ASK utility is a simple tool to ask simple interactive questions to the end-user. For instance, if your model requires a scalar to be changed regularly, instead of letting the end-user change the `.gms` source file, it may be better to pop up a window, with a question text, where the required number can be entered. The ASK tool allows you to do this. As the ASK tool generates a standard GAMS include file, this file can then be used through a `$include` statement:

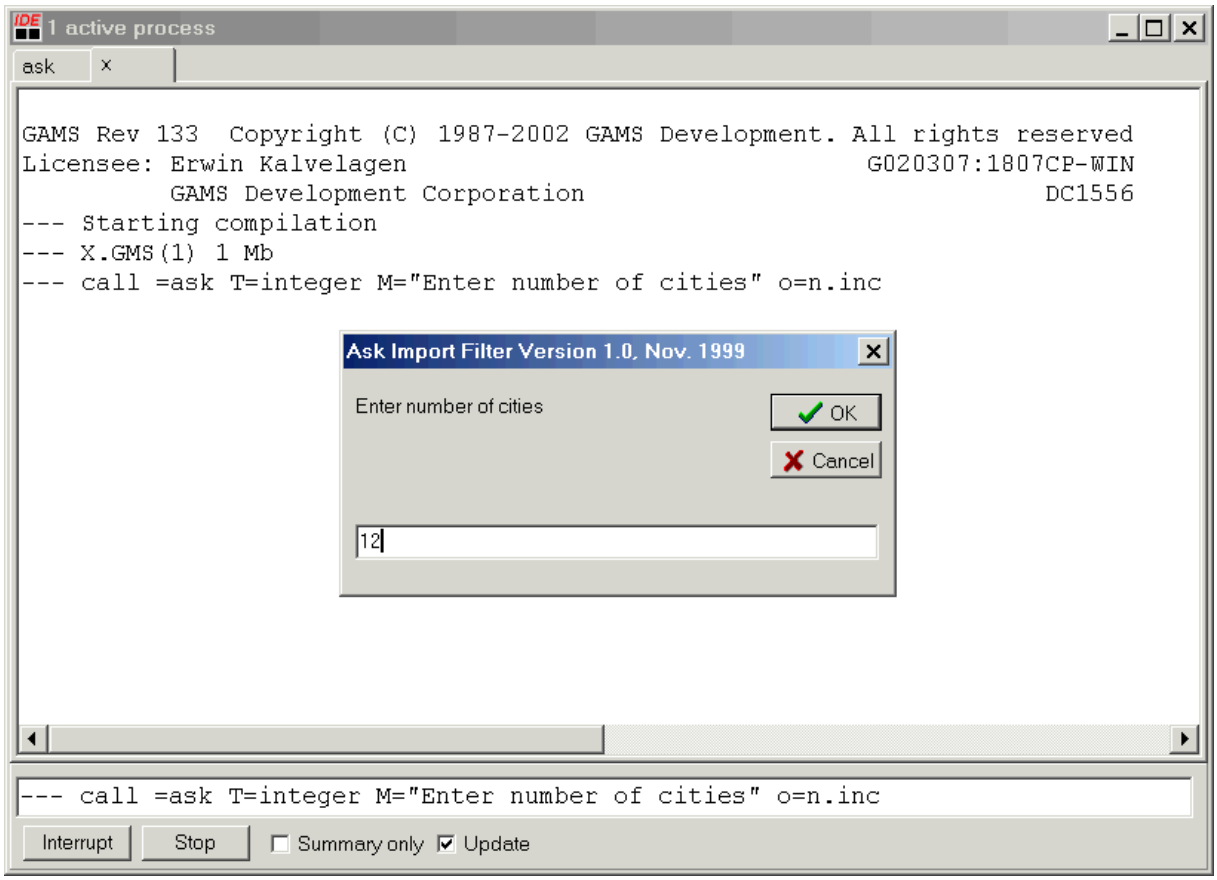
```
$call =ask T=integer M="Enter number of cities" o=n.inc
Scalar n 'number of cities' /
$include n.inc
/;
display n;
```

The `$call` statement will invoke the ASK tool. If `ASK.EXE` is not located in the GAMS system directory but placed somewhere else you may have to provide a path, as in:

```
$call =d:\util\ask T=integer M="Enter number of cities" o=n.inc
```

The parameter `T=integer` indicates we want to ask for an integer number (`T` means type). The parameter `M="text"` specifies the question text. Finally `O=filename` sets the name of the include file that ASK should create.

When we run this GAMS fragment from the IDE, we will see:



After enter a number and hitting the OK button GAMS will continue. The listing file will demonstrate clearly how the include file was formed:

```
 2  scalar n 'number of cities' /
INCLUDE    D:\GAMS PROJECTS\ASK\N.INC
 4  * Ask Import Filter Version 1.0, Nov. 1999
 5  * Erwin Kalvelagen, GAMS Development Corp.
 6  12
 7  /;
 8  display n;
Include File Summary
 SEQ   GLOBAL TYPE      PARENT   LOCAL  FILENAME
   1        1 INPUT          0       0  D:\GAMS PROJECTS\ASK\X.GMS
   2        1 CALL           1       1  =ask T=integer M="Enter number of cities" o=n.inc
   3        3 INCLUDE        1       3  .D:\GAMS PROJECTS\ASK\N.INC
```

In this case `12` was entered and the `OK` button was pressed. If the user pressed the `CANCEL` button, an error will be generated and the listing file will show that the `CANCEL` button was pressed:

```
   2  scalar n 'number of cities' /
  INCLUDE    D:\GAMS PROJECTS\ASK\N.INC
   4  * Ask Import Filter Version 1.0, Nov. 1999
   5  * Erwin Kalvelagen, GAMS Development Corp.
   6  * Cancel was pressed
   7  /;
* ****  $1
* **** LINE      4 INPUT       D:\GAMS PROJECTS\ASK\X.GMS
* ****   1  Real number expected
   8  display n;
```

In case you want to limit the integer that the user is allowed to enter you can specify a lower and an upperbound as in:

```
$call =ask T=integer M="Give integer, between 0 and 5" L=0 U=5 O=n1.inc
Scalar n1 'an integer 0..5' /
$include n1.inc
/;
display n1;
```

This will only accept values between `0` and `5`.

To allow the user to specify a floating point number, we can use `T=float`. An example is:

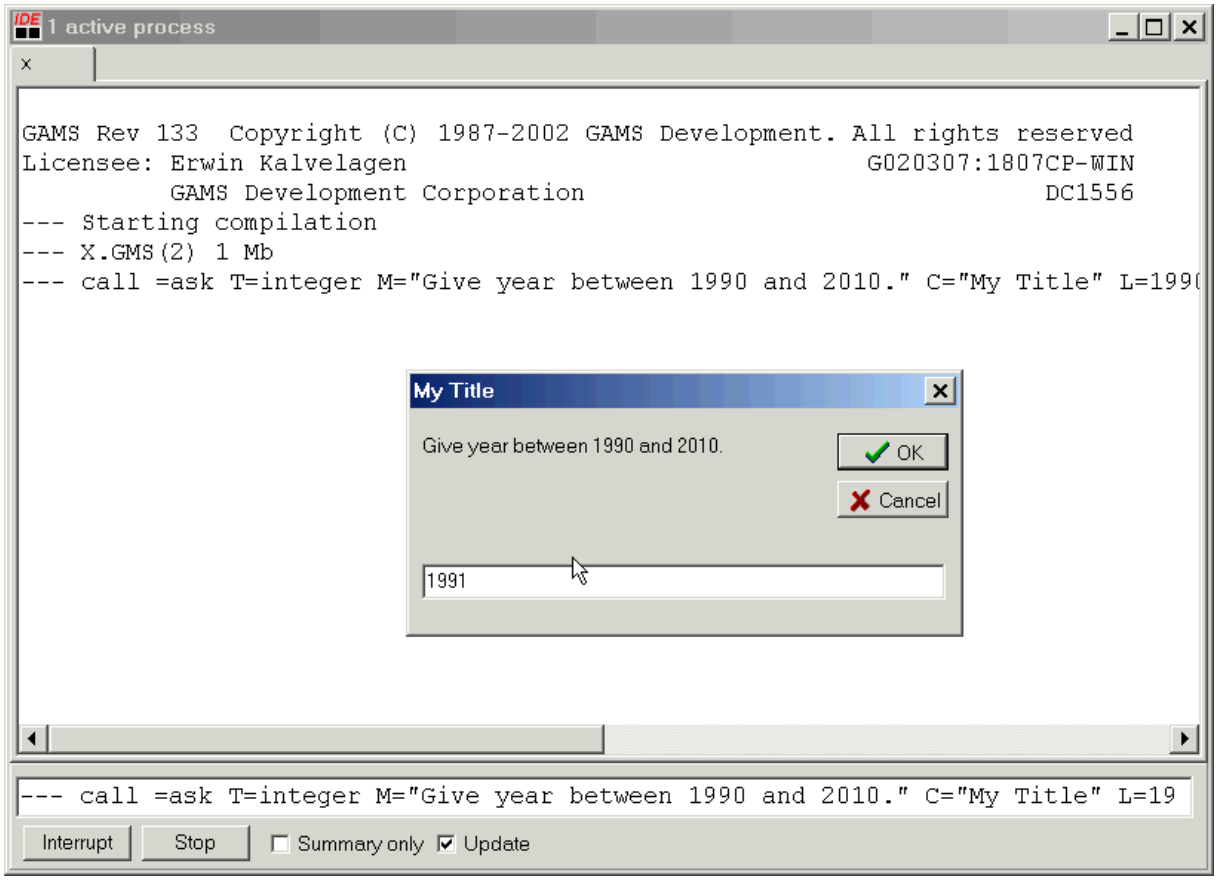
```
$call =ask T=float M="Give floating point number, no bounds" O=x.inc
Scalar x 'real' /
$include x.inc
/;
display x;
```

The floating point popup window can be told to make sure the number entered is within certain bounds, using the `L=lowerbound` and `U=upperbound` syntax:

```
$call =ask T=float M="Give floating point number, between 0 and 5" L=0 U=5.0 O=x1.inc
Scalar x1 'real' /
$include x1.inc
/;
display x1;
```

Up to now, the include file generated by ASK just contained a single number. ASK can generate mode complicated include files. Consider the example:

```
* import a set
$call =ask T=integer M="Give year between 1990 and 2010." C="My Title" L=1990 U=2010 R="set i /1990*
$include i.inc
display i;
```



The parameters `C="caption"` and `R="resultstring"` are new. The caption is simple: it sets the name of the window. The result string is a string that is returned after ASK has substituted `s` for the result. I.e. if we enter `1991`, then the result written to the include file is `set i /1990*1991/;`.

The listing file can be used to check the correct behavior:

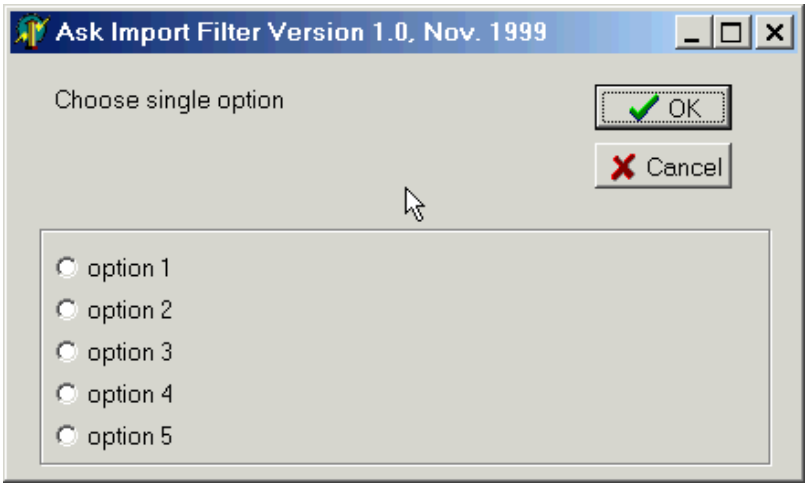
```
 1  * import a set
INCLUDE    D:\GAMS PROJECTS\ASK\I.INC
 4  * Ask Import Filter Version 1.0, Nov. 1999
 5  * Erwin Kalvelagen, GAMS Development Corp.
 6  set i /1990*1991/;
```

```
 7  display i;
Include File Summary
 SEQ   GLOBAL TYPE      PARENT   LOCAL  FILENAME
   1        1 INPUT          0       0  D:\GAMS PROJECTS\ASK\X.GMS
   2        2 CALL           1       2  =ask T=integer M="Give year between 1990 and 2010." C="My Ti
                                        R="set i /1990*%s/;" O=i.inc
   3        3 INCLUDE        1       3  .D:\GAMS PROJECTS\ASK\I.INC
```

6.4.3 Radio Button

Radio buttons can be used through the parameter `T=radiobuttons` as in:

```
* import a number through radio buttons
$call =ask T=radiobuttons M="Choose single option" D="option 1|option 2|option 3|option 4|option 5" I
$include n2.inc
display n2;
```



The parameter `D="option 1|option 2|option 3|option 4|option 5"` specifies the text of the options shown. The list `E="1|2|3|4|5"` gives the return (exit) strings when a certain option is chosen. I.e. if the second option os chosen in the list specified with the `D` parameter, then the second string in the `E` list is returned. The result is again substituted in the string specified in the `R` parameter if it exists.

In this example the command line became rather long and difficult to handle. In addition some Windows systems have restrictive maximum lengths for the command line. Therefore we offer the possibility to specify the command line arguments in a separate external text file. This text file is passed using

```
@filename
```

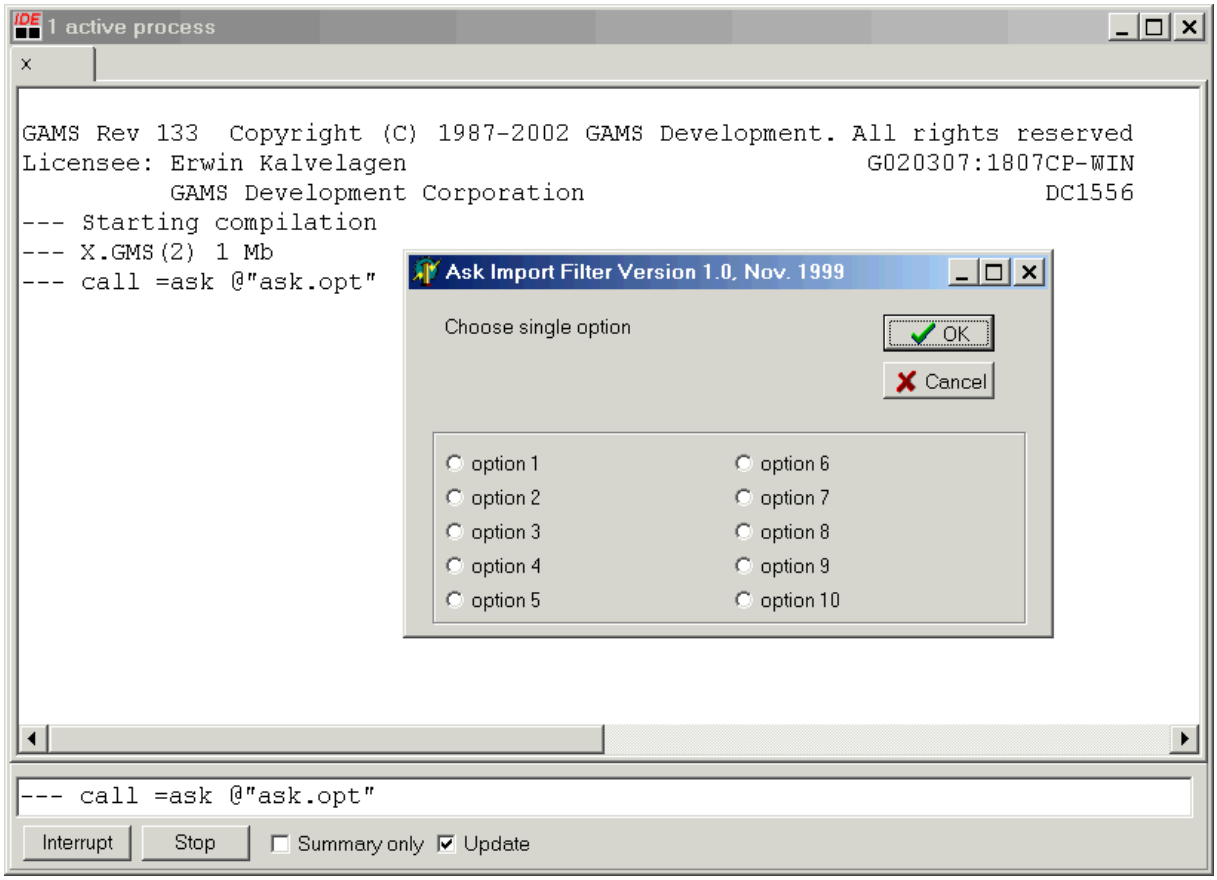
. Assume the file `ask.opt` looks like:

```
T=radiobuttons
M=Choose single option
D=option 1|option 2|option 3|option 4|option 5|option 6|option 7|option 8|option 9|option 10
E=1|2|3|4|5|6|7|8|9|10
2
R=scalar n3 option /%s/;
O=n3.inc
```

Every command line parameter is specified on a separate line. Notice the strange option 2; this tells ASK to display the radio buttons in two columns. We can use this option file as follows:

```
* id, now 2 columns and using a parameter file
$call =ask @"ask.opt"
$include n3.inc
display n3;
```

The quotes around the filename are optional, and really only needed if the filename contains blanks.



If you want to keep all the logic in one place, then one can use GAMS to generate the option file. It is noted that it is not possible to use the PUT facility for this. I.e.:

```
File f /asktest.opt/;
put f;
put 'T=checklistbox'/
  'M=Choose multiple options'/
  'D=option 1|option 2|option 3|option 4|option 5'/
  'E=1|2|3|4|5'/
  'R=%s checked list box choice'/
  'O=k2.inc'/;
putClose;
$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;
```

is not correct: the $call is handled at compile time, before the PUT statement has done its work. However, one could use the following:

```
$echo 'T=checklistbox'                                    > asktest.opt
$echo 'M=Choose multiple options'                        >> asktest.opt
$echo 'D=option 1|option 2|option 3|option 4|option 5' >> asktest.opt
$echo 'E=1|2|3|4|5'                                      >> asktest.opt
$echo 'R=%s checked list box choice'                     >> asktest.opt
$echo 'O=k2.inc'                                         >> asktest.opt
$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;
```

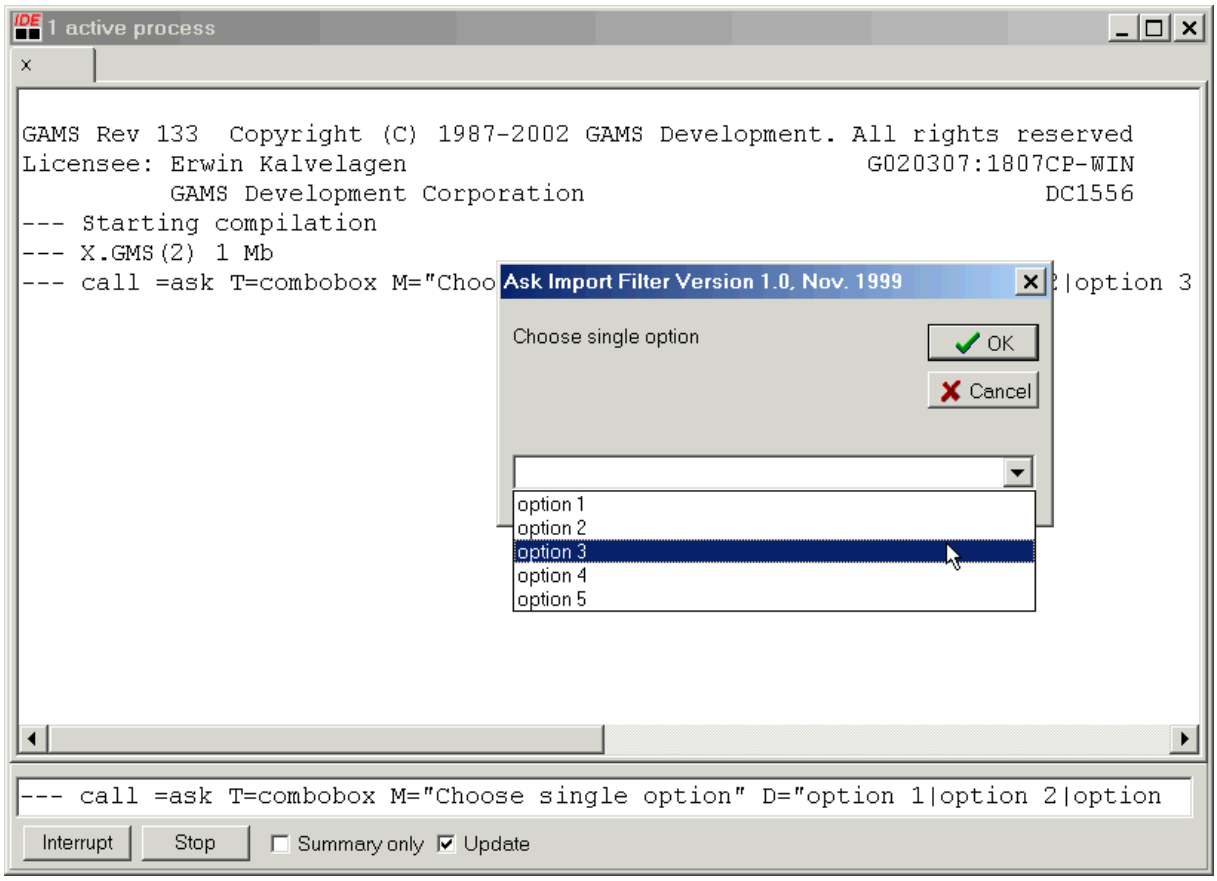
or better:

```
$onEcho > asktest.opt
T=checklistbox
M=Choose multiple options
D=option 1|option 2|option 3|option 4|option 5
E=1|2|3|4|5
R=%s checked list box choice
O=k2.inc
$offEcho
$call =ask @asktest.opt
Set k2 /
$include k2.inc
/;
display k2;
```

6.4.4 Combo Box

The next type is T=combobox which also allows a single selection:

```
* import a number through a combo box
$call =ask T=combobox M="Choose single option" D="option 1|option 2|option 3|option 4|option 5" E="1
$include n4.inc
display n4;
```



As an example consider the case where the model comes with three data sets: a small one, a medium sized one and a large data set. Each data set is stored in a separate include file: `small.inc`, `medium.inc` and `large.inc`. We want to ask the user which data set should be used and the correct include file should be used. This can be accomplished by:

```
$echo 'T=combobox' > ask.opt
$echo 'M=choose data set' >> ask.opt
$echo 'D=Small data set|Medium data set|Large data set' >> ask.opt
$echo 'E=small.inc|medium.inc|large.inc' >> ask.opt
$echo 'R=$include %s' >> ask.opt
$echo 'O=dataset.inc' >> ask.opt
$call =ask @ask.opt
$include dataset.inc
```

Newer GAMS systems allow:

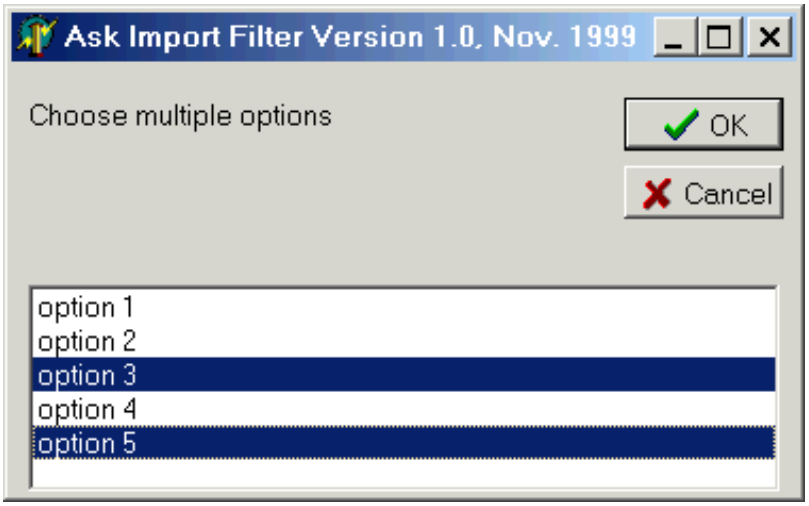
```
$onEcho > ask.opt
T=combobox
M=choose data set
D=Small data set|Medium data set|Large data set
E=small.inc|medium.inc|large.inc
R=$include %s
O=dataset.inc
$offEcho
$call =ask @ask.opt
$include dataset.inc
```

In this case `dataset.inc` will contain a single line: another include statement, which is either `$include small.inc`, `$include medium.inc` or `$include large.inc`.

6.4.5 List and Checklist Box

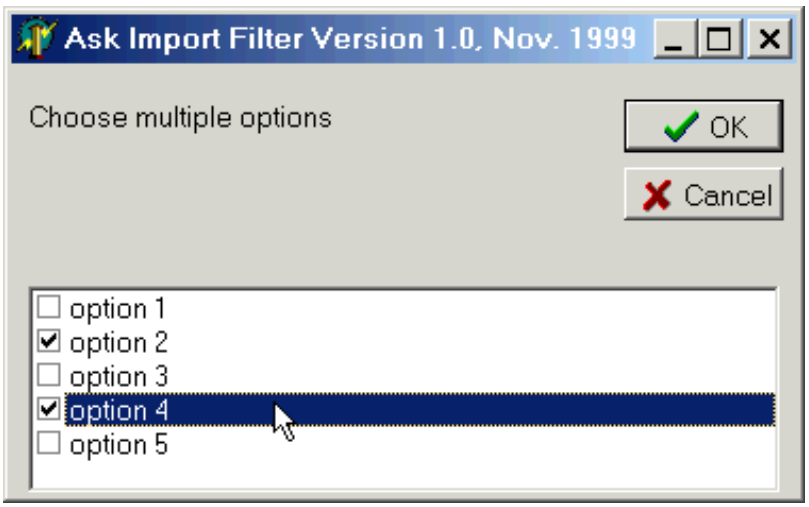
In some cases it may be useful to be able to select multiple items from a list. This can be done with T=listbox:

```
* import a set through a listbox
$call =ask T=listbox M="Choose multiple options" D="option 1|option 2|option 3|option 4|option 5" E=
Set k /
$include k.inc
/;
display k;
```



Selecting multiple entries involves holding down the CTRL key. Sometimes a convenient alternative is T=checklistbox :

```
* import a set through a checked listbox
$call =ask T=checklistbox M="Choose multiple options" D="option 1|option 2|option 3|option 4|option
Set k2 /
$include k2.inc
/;
display k2;
```



When we select options 1, 3 and 5, the following include file is generated:

```
* Ask Import Filter Version 1.1, Aug. 2002
* Erwin Kalvelagen, GAMS Development Corp.
1 'checked list box choice'
3 'checked list box choice'
5 'checked list box choice'
```

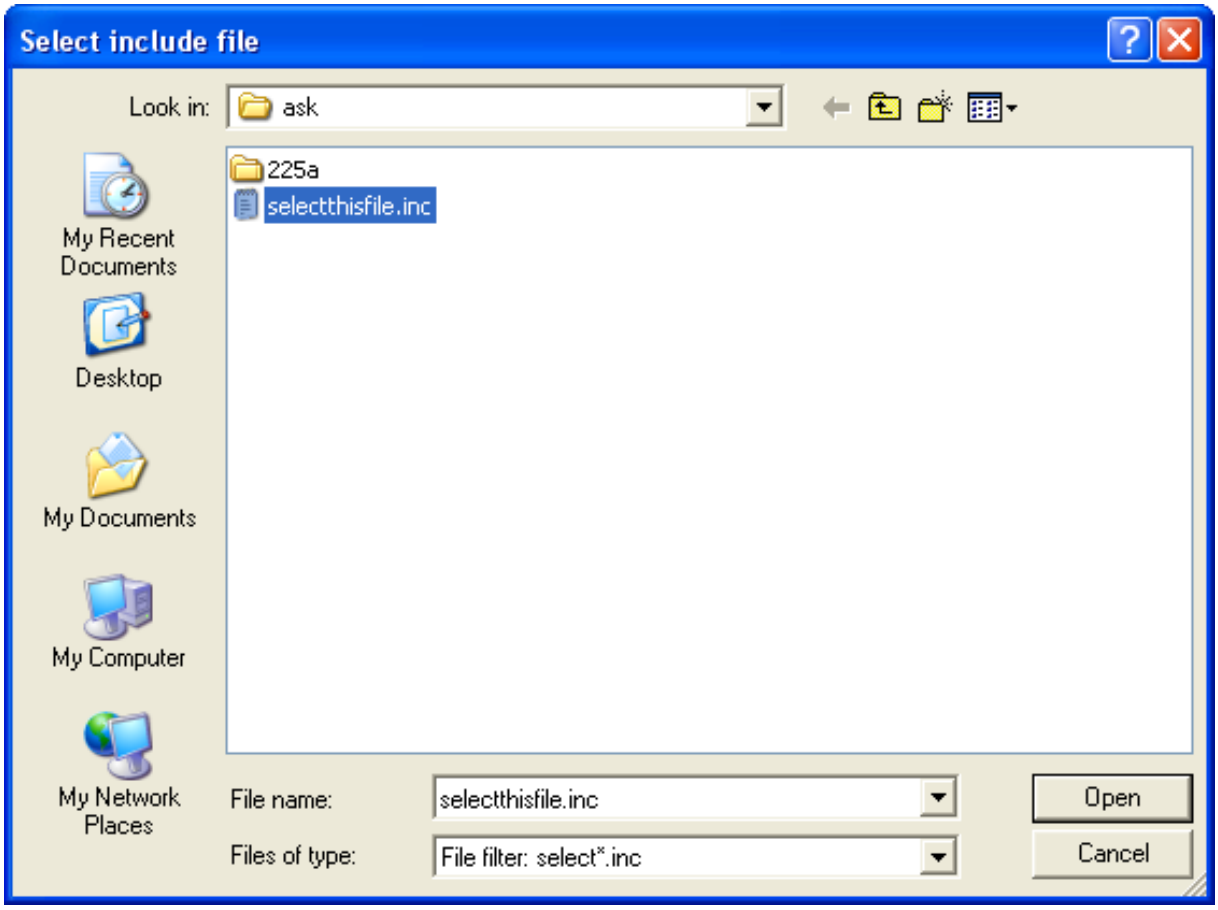
which will populate the set k2 as follows:

```
 1  * import a set through a checked listbox
 3  set k2 /
INCLUDE    D:\GAMS PROJECTS\ASK\K2.INC
 5  * Ask Import Filter Version 1.1, Aug. 2002
 6  * Erwin Kalvelagen, GAMS Development Corp.
 7  1 'checked list box choice'
 8  3 'checked list box choice'
 9  5 'checked list box choice'
10  /;
11  display k2;
----     11 SET k2
1,    3,    5
```

The last type is the generic string type: `T=string`. This allows the user to enter any string, which is then copied to an include file as is.

6.4.6 File Open Dialog Box

The type `T=fileopenbox` will display a file open dialog box from which the user can select a file.



Related options are `I=InitialDirectory` and `F=Filter`. A complete example to ask for an include file could be:

```
$call =ask T=fileopenbox I="%system.fp%" F="select*.inc" o=fln.inc R="$include '%s'" C="Select inclu
$include fln.inc
```

This will open a file open dialog box with the starting directory being the `GAMS project/working` directory (this is also where GAMS looks for include files by default). Only files with mask `SELECT*.INC` are shown. The file `FLN.INC` will contain an include statement with the file the user has selected.

A related method would be:

```
$call =ask T=fileopenbox I="%system.fp%" F="select*.inc" o=fln.inc R="$setglobal incfile '%s'" C="Se
$include fln.inc
$include %incfile%
```

where we use a `$setglobal` to set the macro incfile to contain the user-specified file name.

To let the user choose from a set of related GDX files, one could use something like:

```
$call =ask T=fileopenbox I="%system.fp%" F="myproject*.gdx" o=setgdxname.inc R="$setglobal gdxfile '
$include setgdxname.inc
$gdxIn %gdxfile%
$load i
$load j
```

In case you want to ask for a filename of a file to be written, use the type `T=filesavebox`. E.g.:

```
$call =ask T=filesavebox I="%system.fp%" o=fln.inc R="$setglobal gdxfile '%s'" C="Specify gdx file"
$include fln.inc
Set i / a, b, c /;
execute_unload '%gdxfile%', i;
```

In case you want to ask for a directory, use the type `T=selectdirectory`. In this case the filter option `F` is ignored. E.g.:

```
$call =ask T=selectdirectory I="%system.fp%" o=fln.inc R="$setglobal myInputdir '%s'" C="Select dire
$include fln.inc
$log Selected directory "%myInputdir%"
```

6.5 Cholesky

Note

> This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This calculates the Cholesky decomposition of a symmetric positive definite matrix A: $A = LL^t$ The matrix A is indexed over `A(i,i)`.

6.5.1 Usage

Command line:

```
gamstool [linalg.]Cholesky i A L gdxIn=fileIn.gdx gdxOut=fileOutgdx
```

Compile time:

```
$callTool [linalg.]Cholesky i A L [gdxIn=fileIn.gdx] [gdxOut=fileOutgdx]
```

Execution time:

```
executeTool '[linalg.]Cholesky i A L [gdxIn=fileIn.gdx] [gdxOut=fileOutgdx]';
```

Where:

Argument	Description
i	Name of set used in matrix i(*).
A	Name of two-dimensional matrix parameter A(i,i).
L	Name of two-dimensional parameter containing the factor L(i,i).

The following named parameters are available:

Parameter	Description
gdxIn=fileIn.gdx	Name of GDX file that contains symbols i and A. Mandatory if called from the command line, otherwise optional.
gdxOut=fileOut.gdx	Name of GDX file that contains symbol L after execution. Mandatory if called from the command line, otherwise optional.

6.5.2 Examples

```
executeTool.checkErrorLevel 'linalg.cholesky n Yl L';
* Check if Cholesky factorization is correct
Parameter Y_, Ydiff;
Y_(i,j)    = sum(n, L(i,n)*L(j,n));
Ydiff(i,j) = round(Y.l(i,j) - Y_(i,j),1e-6);
option Ydiff:8:0:1;
abort$card(Ydiff) Ydiff;
```

For the complete example, see model [**Maxcut**] in the GAMS Model Library.

```
execute_unload 'a.gdx', i, a;
executeTool.checkErrorLevel 'linalg.cholesky i a L gdxin=a.gdx gdxout=b.gdx';
execute_load 'b.gdx', L;
```

```
$callTool linalg.cholesky i a L;
```

6.6 CSV2GDX

CSV2GDX is a tool that reads a CSV file and writes to a GDX file. There are multiple ways to read [CSV] (https://en.wikipedia.org/wiki/Comma-separated_values) files (Comma Separated Values) inside GAMS (see Data Exchange with Text Files for instance), but a number of features available in CSV2GDX make it possible to read a CSV file where GAMS itself cannot be used. In addition to the syntax explanation and the basic functionalities demonstrated on some examples, this tutorial also discusses some advantages and disadvantages of CSV2GDX compared to the GAMS internal table statement.

6.6.1 Usage

CSV2GDX is called by specifying the CSV file and several options to define how to read the data.

```
csv2gdx filename {options}
```

Filename

The input file; the .csv file extension is assumed when no extension has been specified.

Parameters can also be read from a text file; the use of an external file for parameters is indicated by preceding the file name with a @ (at sign). When reading parameters from a text file, lines starting with an asterisk (∗) will be ignored. See also Example 8.

Option	Default	Description

6.6.2 Options

The following options can be used when calling `CSV2GDX`:

Option	Default	Description
acceptBadUels	`N`	Indicate if bad UELs are accepted or result in an error return code.
autoCol	none	Generate automatic UELs for each column.
autoRow	none	Generate automatic UELs for each row.
checkDate	`N`	Write GDX file only if the CSV file is more recent than the GDX file.
colCount	none	Number of columns in the input file.
decimalSep	Period	Specify a decimal separator.
fieldSep	Comma	Specify a field separator.
id	none	Identifier for the symbol in the GDX file.
index	none	Identify columns to get UELs from.
output	<CSVFilename>.gdx	Optional output filename.
password	none	Password for an encrypted input file.
storeZero	`N`	Indicate how zero values are handled.
text	none	Specify the column to get explanatory text from.
trace	1	Controls the amount of information written to the log.
useHeader	`N`	Indicate if the first row is a header row.
value	none	Specify the column to get the values from.
valueDim	`N`	Adds an extra dimension for values.
values	none	Specify the columns to get the values from.

Note

- The user has to specify an identifier for the symbol in the GDX file within the parameter `id`, regardless of the data structure in the CSV file.
- `CSV2GDX` determines the number of columns in the CSV file from the header row or from the user defined input specified in the parameter `colCount`. Therefore, the user must enable the option `useHeader` or has to specify the number of columns within `colCount` in any case.

Some more detailed remarks on the options:

acceptBadUels = *boolean* (default=`N`)

This option specifies how to proceed, when a bad UEL (e.g. too long) is encountered. If set to `N`, reading is stopped, and an error is signaled. When set to `Y`, a valid UEL is made up and reading is continued.

autoCol = *string*

Generate automatic UELs for each column. The autoCol string is used as the prefix for the column label numbers. This option overrides the use of a header row. However, if there is a header row, one must skip the row by enabling `useHeader`. This option is demonstrated in Example 3.

autoRow = *string*

Generate automatic UELs for each row. The autoRow string is used as the prefix for the row label numbers. The generated unique elements will be used in the first index position shifting other elements to the right. Using `autoRow` can be helpful when there are no labels that can be used as unique elements but also to store entries that would be a duplicate entry without an unique row label. This option is demonstrated in Example 3.

checkDate = *boolean* (default=`N`)

Write GDX file only if the CSV file is more recent than the GDX file to save resources when running the model containing the `CSV2GDX` call multiple times. This option is demonstrated in Example 5.

colCount = *integer*

Number of columns in the input file. This parameter is required if there is no header row, since `CSV2GDX` determines the number of columns from the header row. This option is demonstrated in Example 3. Note that the `lastCol` constant cannot be used for the `colCount` option.

decimalSep = [Period, Comma] (default=Period)

Specify a decimal separator. The decimal is normally a period, but this parameter allows a comma as the decimal separator, too. Special values recognized are `Eps`, `NA` resp. `N/A`, `Inf`, `True`, `False`, `None`, `Null` and `Undef` (case insensitive). A string that is not recognized as a valid number will be stored as `Undef`. This option is demonstrated in Example 2 and Example 6 (focusing on reading special values) for instance.

fieldSep = [Comma, SemiColon, Tab] (default=Comma)

Specify a field separator. Fields are normally separated by a comma, but this parameter allows some additional choices. Using tabs as delimiter should be avoided, since text editors act different on handling them. For instance, one must specify tabs in the GAMS IDE explicitly using %system.tab%:

```
$onEcho > tabSeparated.csv
USA%system.tab%100
GER%system.tab%70
$offEcho

$call csv2gdx tabSeparated.csv id=x fieldSep=tab index=1 colCount=2 value=lastCol
```

This option is demonstrated in Example 2 and Example 3 for instance.

id = *string*

Identifier for the symbol in the GDX file. Additional symbols, `Dim1`, `Dim2`, ... for the domain sets of the symbol id will be added automatically to the GDX file. Executing `CSV2GDX` without specifying an identifier will fail.

index = *list of columns*

Identify columns to get UELs from. The columns are represented as a list of integers separated by comma. For example `index=1,2,3,4` resp. `index=(1,2,3,4)`; in this case the notation `1..4` is allowed. Brackets can only be used on Windows systems. The `index` option is used in all examples.

output = *filename* (default=<CSVFilename>.gdx)

Optional output filename. If no output file is specified, `CSV2GDX` will use the input filename and change the file extension to .gdx. If no path is specified, the output file will be created in the current directory. This option is demonstrated in Example 2 for instance.

password = *string*

Password for an encrypted input file. Use ENDECRYPT to encrypt a file. This option is demonstrated in Example 7.

storeZero = *boolean* (default=`N`)

Indicate if zero values are ignored or written as `EPS`; an empty field is always ignored. This option is demonstrated in Example 6.

text = *integer*

Specify the column to get explanatory text from when reading a set. For example `text=5`. This option is demonstrated in Example 9.

trace = *integer* (default=1)

Controls the amount of information written to the log. Higher values will generate more output. Valid range is `0..3`. Set `trace=0` to prevent writing any information to the log. This feature is demonstrated in Example 6.

useHeader = *boolean* (default=`N`)

Indicate if the first row is a header row. The fields in the header row of the columns specified within the values option will be used as UELs. A header row is not needed or should be ignored when using the colCount or autoCol option. To skip one existing header row while using `autoCol`, enable `useHeader`. This option is demonstrated in Example 1 and Example 2 for instance.

value = *integer*

Specify the column to get the values from. For example `value=5`. This option is demonstrated in Example 2.

valueDim = *boolean* (default=`N`)

Indicate if an extra dimension for values is added to the parameter even if there is just one value column. This is ignored, if there is no value column. This feature is demonstrated in Example 2.

values = *list of columns*

Specify the columns to get the values from. When using a list of columns for the values and `useHeader` enabled, each field in the first row of the columns is used as UEL to identify the values in those columns. See also useHeader and autoCol below. If the number of columns is unknown, the symbolic constant `lastCol` can be useful: `values=2..lastCol`. This option is demonstrated in Example 1 and Example 5 for instance.

6.6.3 Advances and limitations

Advances

- CSV2GDX enables the user to read CSV data where the table statement cannot be used without doing further preprocessing, e.g. in case of semicolon- or tab separated data or comma separated decimals.
- In general, CSV2GDX is a quite performant tool compared to the GAMS internal table statement.

Limitations

- Suppose we want to skip some rows while reading the data. For example, if the CSV file contains some reference information in a specific row which we do not want to be stored in the domain sets or in the parameter. However, skipping rows cannot be done with CSV2GDX.
- There might be CSV files with no header and varying, unknown length of rows. Since CSV2GDX determines the number of columns based on the header row or by setting the colCount option in advance, CSV2GDX might return incorrect results or the execution is aborted.
- Reading several parameters from a CSV file cannot be done directly within the CSV2GDX call. The data must be split later on as demonstrated in Example 4.
- CSV2GDX cannot read CSV files containing line breaks within a (quoted) field. You will either get an error message - "Quoted field not terminated with closing quote" - or the result might not be correct for unquoted fields, because the field content will be cut by the line break.

6.6.4 Getting Started

We introduce the basic functionalities of CSV2GDX on some simple examples. Note that many CSV files can be read inside GAMS directly using a table statement, but a number of features available in CSV2GDX enable the user to read a CSV file where the table statement cannot be used, e.g. reading files with semicolon separated data or if the decimals are separated by comma instead of a period.

6.6.4.1 Example 1 - Reading CSV Files with CSV2GDX

The first example of this collection demonstrates the key commands of CSV2GDX. For instance, consider the table statement of the model [**trnsport**] from the GAMS Model Library:

```
Table d(i,j) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;
```

The data can be stored in distance.csv like this:

```
,new-york,chicago,topeka
seattle,2.5,1.7,1.8
san-diego,2.5,1.8,1.4
```

First of all, `CSV2GDX` is called now, to generate distance.gdx by processing the input file distance.csv:

```
csv2gdx distance.csv id=d index=1 values=2..lastCol useHeader=y
```

`CSV2GDX` generates one single parameter `d` and two domain sets from the input file. The name of the parameter in the GDX file must be declared within the id option, while the domain sets for this parameter will be labeled with `Dim1` and `Dim2` automatically. Column number one is specified as the first domain set within the index option. The values option is used to specify the column numbers `2,3,4` containing the data values. By enabling the useHeader option, the fields of the first row of the columns specified within the `values` option will be handled as the second domain set. If the number of columns is unknown in advance, one can use the lastCol constant in the `values` or `index` option.

However, to complete the declaration of the sets and parameter for the model trnsport, one must load the data from distance.gdx:

```
Set
   i 'canning plants'
   j 'markets';

$gdxIn distance.gdx
$load i = Dim1
$load j = Dim2

Parameter d(i,j) 'distance in thousands of miles';
$load d
$gdxIn
display i, j, d;
```

This example is also part of the GAMS Data Utilities Library, see model [**csv2gdx2**] for reference.

6.6.4.2 Example 2 - Reading Semicolon separated Data

In this example, the distances from the previous example are stored as a list. We want to read the cities and the column containing the miles measurement. Note, that the fields are separated by semicolon and the decimals by comma. As described in Data Exchange with Text Files the CSV file must be preprocessed with the `POSIX` tools in advance to replace commas by dots and semicolons by commas to read the data directly using a simple table statement.

```
i;j;miles
seattle;new-york;2,5
seattle;chicago;1,7
seattle;topeka;1,8
san-diego;new-york;2,5
san-diego;chicago;1,8
san-diego;topeka;1,4
```

The data will be stored as distanceOut.gdx by adding the output file option. One can specify the field and decimal separators within the fieldSep and decimalSep option. Domain sets for the parameter to be read are declared by `index=1,2`.

```
csv2gdx distance.csv output=distanceOut.gdx id=d fieldSep=semiColon decimalSep=comma index=1,2 useHea
```

In order to load the data from the GDX file, execute the commands from the previous example. However, note that `Dim2` does not contain the UELs from the header row this time, but the unique elements of the second column specified in the `index` option. The option `useHeader` is enabled to indicate that there is a header row to be skipped when reading the values. Also note, that the symbol `d` in the GDX file has exactly two dimensions. It may be useful to add an additional dimension to the parameter `dmod`, e.g. if different measurement units for the distances may become relevant later on and need to be calculated (kilometer for instance).

This can be done by adding the option valueDim:

```
csv2gdx distance.csv output=distanceOut.gdx id=d fieldSep=semiColon decimalSep=comma index=1,2 useHea
```

The option adds a third dimension to `d`. Now we want to add the distances in kilometer by calculating the values inside the model:

```
Set m 'measurement unit' / miles, km /;

Parameter dmod(i,j,m);

$gdxIn distanceOut.gdx
$load dmod = d
$gdxIn
display dmod;

dmod(i,j,'km') = 1.852*dmod(i,j,'miles');
display dmod;
```

Parameter `d` (`valueDim` disabled):

```
----     36 PARAMETER d  distance in thousands of miles

             new-york     chicago      topeka

seattle         2.500       1.700       1.800
san-diego       2.500       1.800       1.400
```

Parameter `dmod` resp. `d` with `valueDim` enabled before calculating further measurements:

```
----     49 PARAMETER dmod

                           miles

seattle  .new-york         2.500
seattle  .chicago          1.700
seattle  .topeka           1.800
san-diego.new-york         2.500
san-diego.chicago          1.800
san-diego.topeka           1.400
```

Parameter `dmod` with `valueDim` enabled after calculating the distances in kilometer inside the GAMS model:

```
----     52 PARAMETER dmod

                           miles          km

seattle  .new-york         2.500       4.630
seattle  .chicago          1.700       3.148
seattle  .topeka           1.800       3.334
san-diego.new-york         2.500       4.630
san-diego.chicago          1.800       3.334
san-diego.topeka           1.400       2.593
```

This example is also part of the GAMS Data Utilities Library, see model [**csv2gdx3**] for reference.

6.6.4.3 Example 3 - Dealing with missing Labels and Duplicates

Missing Labels

The file EUCData.csv contains the extracted euclidean coordinates of the first nine cities of berlin52.tsp from [TSPLib] (http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/). You might want to import the data to calculate a complete distance matrix inside GAMS to find an optimal traveling salesman tour for instance.

```
565.0;575.0
25.0;185.0
345.0;750.0
945.0;685.0
845.0;655.0
880.0;660.0
25.0;230.0
525.0;1000.0
580.0;1175.0
```

There is no header row, neither a column with labels to serve as domain sets for the coordinates of the cities. However, CSV2GDX automatically generates UELs for columns and rows by adding ascending numbers to an user-defined prefix specified within the autoCol and autoRow option. Also note, because of the missing header, the number of columns in the file must be manually determined and declared within colCount.

```
csv2gdx EUCData.csv id=coord fieldSep=semiColon autoCol=x autoRow=city colCount=2 values=1,2
```

The rows will be labeled with city1...city9, the columns with x1 and x2.

Load the parameter and sets from the GDX file to calculate the complete distance matrix. The set Dim1 contains the UELs city1...city9 for the set i of cities, Dim2 is the set of axes/coordinates i.e. containing the elements x1 and x2 to represent the x-axis and y-axis:

```
Set
   i    'cities'
   axes 'x1 and x2 axes';

$gdxIn EUCData.gdx
$load i    = Dim1
$load axes = Dim2

Parameter coord(i,axes) 'coordinate of city i';
$load coord
$gdxIn
display coord;
```

The display statement generates the following output:

```
----     43 PARAMETER coord  coordinate of city i

               x1          x2

city1     565.000     575.000
city2      25.000     185.000
city3     345.000     750.000
city4     945.000     685.000
city5     845.000     655.000
city6     880.000     660.000
city7      25.000     230.000
city8     525.000    1000.000
city9     580.000    1175.000
```

One could now easily proceed calculating a complete distance matrix:

```
Alias (i,j);

Parameter c(i,j) 'euclidean distance between city i and j';
c(i,j) = eDist(coord(i,"x1") - coord(j,"x1"),coord(i,"x2") - coord(j,"x2"));
```

Duplicates

In the previous example, using the `autoCol` and `autoRow` option had an additional benefit as we declared the set of the cities on the fly. However, one major advantage/purpose of these options is to prevent error messages or loss of data when reading rows with duplicate keys.

Consider the input file duplicates.csv:

```
red,red,1
red,red,2
red,green,3
blue,blue,4
```

Note the duplicate key in the first two rows. By the use of the `autoRow` parameter in the `CSV2GDX` call unique labels are added to each row. This way a GAMS program can store all data with duplicate keys and prepare for better error messages.

```
csv2gdx duplicates.csv id=data index=1,2 value=3 colCount=3 autoRow=row
```

Ascending numbers will be added to the 'row' prefix specified.

The data can be easily loaded into sets and parameter by executing the following lines:

```
Set
   row   'UELs generated by autoRow'
   color 'set of colors';

$gdxIn duplicates.gdx
$load  row   = Dim1
$load  color = Dim2
$loadm color = Dim3

Parameter data(row,color,color);
$load data
$gdxIn
display row, color, data;
```

Note the usage of the $loadm command to merge all colors into one set of colors. The display statement generates the following output in the listing file:

```
----     77 SET row  UELs generated by autoRow

row1,    row2,    row3,    row4


----     77 SET color  set of colors

red  ,    blue ,    green
```

```
----     77 PARAMETER data

                  red        blue       green

row1.red        1.000
row2.red        2.000
row3.red                                3.000
row4.blue                   4.000
```

The complete example is also part of the GAMS Data Utilities Library, see model [**csv2gdx4**] for reference.

6.6.4.4 Example 4 - Reading several Parameters from a single Input File

This example demonstrates how to read the data from a single input file into different parameters. However, this cannot be done directly using the `CSV2GDX` tool, since `CSV2GDX` writes to a single parameter (see section limitations). For instance, an energy supplier plans to build new transmission lines from their power plants to some distribution stations. There are different stages for the transmissions lines. The relevant data to be read are the plant and station identifiers, the capacity bounds per transmissions line on a certain stage and the associated cost. The data is all stored in a single file:

```
plant;station;length;minCap;maxCap;stage;cost
p1;s1;100;50;100;1;1200
p1;s2;75;35;65;1;500
p1;s1;100;100;150;2;1800
p2;s1;150;50;100;1;1400
p2;s1;150;100;150;2;2000
p2;s1;150;150;200;3;2350
p2;s2;75;25;50;1;600
p2;s2;75;50;75;2;800
p3;s1;80;40;100;1;1050
```

Read networkData.csv with `CSV2GDX` by calling:

```
csv2gdx networkData.csv id=dataPar useHeader=y fieldSep=semiColon index=1,2,6 values=4,5,7
```

Note that the stage stored in column six is a domain set, too, while the length of the transmission line in the third column is of no interest. Since `CSV2GDX` writes to a single parameter, one must split the data later on into `minCap`, `maxCap` and `cost` for instance:

```
Parameter
   dataPar
   minCap(plant,station,stage)
   maxCap(plant,station,stage)
   cost(plant,station,stage);

$gdxIn networkData.gdx
$load plant   = Dim1
$load station = Dim2
```

```
$load stage   = Dim3
$load dataPar
$gdxIn

minCap(plant,station,stage) = dataPar(plant,station,stage,'minCap');
maxCap(plant,station,stage) = dataPar(plant,station,stage,'maxCap');
cost(plant,station,stage)   = dataPar(plant,station,stage,'cost');
```

This example is also part of the GAMS Data Utilities Library, see model **[csv2gdx5]** for reference.

6.6.4.5 Example 5 - Reading economic Data from the World Bank Data Catalog

This example demonstrates how to read some real CSV data from the [World Bank Data Catalog] (https://datacatalog.worldbank.org/). Suppose we want to read some time series data, e.g. GDP growth rates. The data is structured as follows (rows shortened for presentation):

```
"Country Name","Country Code","Indicator Name","Indicator Code","1999","2000","2001","2002","2003","
"East Asia and Pacific","EAA","GDP growth, constant 2010 USD","NYGDPMKTPKDZ","","","","","","","","",
```

We are not interested in the "Country Code", "Indicator Name" and "Indicator Code". Since the annual GDP rate per country is unique, the `CSV2GDX` call is quite easy despite the large number of columns. Note that there are only a few limitations of `CSV2GDX`, discussed in the section Advances and limitations. Empty fields in the data will always be ignored, causing no trouble at all if the field separators are set correctly.

```
csv2gdx GDPData.csv id=GDPG index=1 values=5..lastCol useHeader=y checkDate=y
```

The option checkDate is enabled to save resources if you run the model multiple times, as the GDX file is only written if the CSV file is more recent than the GDX file. To load the sets and parameter from the GDX file, execute the following commands:

```
Set country, year;
$gdxIn GDPData.gdx
$load  country = Dim1
$load  year    = Dim2

Parameter GDPRate(country,year);
$load GDPRate
$gdxIn
display country, year, GDPRate;
```

The display statement generates the following output in the listing file. Obviously, double quotes are removed from the fields of the value columns, but also from the index columns:

```
----     12 SET country

East Asia and Pacific, Europe and Central Asia, Latin America and the Caribbean, Middle East and Nor
Afghanistan          , Albania                , Algeria                        , Angola

----     12 SET year

2015,    2016,    2017,    2018,    2019,    2020


----     12 PARAMETER GDPR

                                   2015        2016        2017        2018        2019        2
East Asia and Pacific             6.504       6.327       6.449       6.182       6.065       5.
Europe and Central Asia           0.963       1.672       3.782       2.901       3.029       2.
Latin America and the Caribbean  -0.562      -1.526       0.902       2.044       2.586       2.
```

Note that the listings have been shortened for presentation. However, there is no data for the years 1999-2014.

This example is also part of the GAMS Data Utilities Library, see model [**csv2gdx6**] for reference.

6.6.5 Additional Examples for extended Use

The examples in this section discuss some special features. Some topics were already mentioned briefly in the previous section like reading compressed and encrypted files or reading the CSV2GDX options from an external file.

6.6.5.1 Example 6 - Reading special Values

To illustrate how special values are interpreted, consider the following data:

```
one,two,three,four,five,six
red,red,,Undef,'3.3',red
red,red,"4.4",5.5,Eps,green
"red",'green',7.7e+02,8.8°,-Inf,blue
blue,blue,10,0,NA,purple
brown,blue,true,false,N/A,green
black,red,None,Null,"True",blue
```

Calling CSV2GDX to read the data and write to GDX:

```
csv2gdx data.csv id=A index=1,2,6 values=3..5 useHeader=y storeZero=y trace=3
```

The GAMS log reports three occurrences of undefined values. Note that the amount of information written to the log was increased by setting the option trace=3.

```
--- call csv2gdx specialValues.csv id=A index=1,2,6 values=3..5 useHeader=y storeZero=y trace=3
CSV2GDX          25.2.0 r67638 ALFA Released 15Aug18 WEI x86 64bit/MS Windows
Header enabled, number of columns = 6
  1: one
  2: two
  3: three
  4: four
  5: five
  6: six
  2: |red|, |red|, ||, |Undef|, |'3.3'|, |red|
  3: |red|, |red|, |4.4|, |5.5|, |Eps|, |green|
  4: |red|, |'green'|, |7.7e+02|, |8.8°|, |-Inf|, |blue|
  5: |blue|, |blue|, |10|, |0|, |NA|, |purple|
Undef Count = 3, No errors, CSV2GDX time = 157ms
```

As mentioned in decimalSep, `CSV2GDX` fails to recognize the number 8.8 from the string 8.8° because of the unknown special character, while 7.7e+02 is interpreted as a number of course. Watch out if there are quotes in the values or index columns. The number 4.4 enclosed by double quotes is interpreted as a number, while '3.3' is not (as you can see in the log, the double quotes are eliminated, while the single quotes remain). In the first column, the double quotes enclosing the string red are removed, while the single quotes in the second column enclosing the string green are not removed. The zero value is stored as `EPS` by setting `storeZero=y`. Note that no error was reported, even though some of the values stored as `Undef` may cause some trouble in your model later on (use $onUndf to enable loading parameters with undefined values).
The booleans `true` and `false` in the second last row are represented by the numericals 1 resp. 0 within GAMS. The special values `None` and `Null` will be converted to 0. Since `storeZore` is enabled in this example, the value displayed for `false`, `None` and `Null` is `EPS`.

Suppose we want to declare the parameter A as: A(color,color,color,number). We can proceed as follows:

```
Set color, number;
$gdxIn data.gdx
$load  color = Dim1
$loadm color = Dim2
$loadm color = Dim3
$load number = Dim4

Parameter A(color,color,color,number);
$onUndf
$load A
$offUndf
$gdxIn
display A;
```

Loading undefined values is enabled by adding $onUndf. This example is also part of the GAMS Data Utilities Library, see model [**csv2gdx7**] for reference.

6.6.5.2 Example 7 - Reading a compressed encrypted Input File

Reading a compressed input file is supported by `CSV2GDX`. The gzip program in the gbin sub-directory or ENDECRYPT must be used for compression. Call gzip to compress the CSV file by running the following command:

```
gzip compressMe.csv -c > compressedFile.csv.gz
```

Gzip writes to standard output by appending -c to keep the original file unchanged. The output is then redirected to compressedFile.csv.gz. To read the compressed file, call CSV2GDX with the same options as for processing the uncompressed file.

```
csv2gdx compressedFile.csv.gz output=unCompressedGzipFile.gdx id=d index=1 useHeader=y values=2..las
```

The data in this example is taken from Example 1.

While the main purpose of Endecrypt is about encrypting and decrypting files, it also compresses the file. If you want to compress a file with Endecrypt, do not specify an password file:

```
cat compressMe.csv | endecrypt -W compressedFile.csv
```

The option -W encrypts standard input and writes to compressedFile.csv. Although there will be no encryption because of the missing password file, one must set the option -W. You can read the compressed file with CSV2GDX:

```
csv2gdx compressedFile.csv output=unCompressedEndycryptFile.gdx id=d index=1 useHeader=y values=2..la
```

Note that there is no further file extension added.

By adding a password file (containing the password in the first line), Endecrypt encrypts and compresses the input file:

```
cat compressMe.csv | endecrypt -W compressedEncryptedFile.csv passwordFile.txt
```

The password file will be deleted. Execute the following command to read the compressed and encrypted file with CSV2GDX:

```
csv2gdx compressedEncryptedFile.csv output=unCompressedDecryptedFile.gdx password=Anton id=d index=1
```

The password option is added. You must set the password directly within the option, in this case "Anton", not the password file. Note that CSV2GDX does not support .gz, .7z etc. compressed files!

This example is also part of the GAMS Data Utilities Library, see model [**csv2gdx8**] for reference.

6.6.5.3 Example 8 - Reading Options from an external File

This example demonstrates how to read the options from an external text file as already mentioned in section Filename. The file distance.csv from the Getting Started Example 1 will be processed with CSV2GDX while reading the options from an external text file named howToRead.txt.

```
csv2gdx distance.csv @howToRead.txt
```

Swapping the order of the CSV filename and the instructions filename inside the call statement will cause an error. Note the leading @ (at sign) on the instructions file, containing the following options:

```
* These lines are interpreted as a comment
* This file specifies the options for reading distance.csv using CSV2GDX

id         = d
fieldSep   = semiColon
decimalSep = comma
index      = 1
useHeader  = y
values     = 2..lastCol
```

This example is also part of the GAMS Data Utilities Library, see model [**csv2gdx9**] for reference.

6.6.5.4 Example 9 - Reading Set Elements with explanatory Text

In this example, we will demonstrate how to read explanatory text of set elements using the CSV2GDX option text.

Consider the input file data.csv:

```
a1,b1,explanatory text of a1.b1,10
a1,b2,explanatory text of a1.b2,20
a2,b1,explanatory text of a2.b1,30
a2,b2,explanatory text of a2.b2,40
```

The set elements are stored in the first and second column, the explanatory text is stored in the third column and there is a fourth column containing some values. Suppose we want to read a two-dimensional set with explanatory text. By default, CSV2GDX does not read explanatory text, i.e. by running the following command:

```
csv2gdx data.csv id=abOnlyUELs index=1,2 colCount=4
```

CSV2GDX creates a GDX file containing the two-dimensional set abOnlyUELs without explanatory text (needless to say, we did not specify a column to get the explanatory text from). By specifying the third column within the index option, i.e. index=1..3, the result will be a three-dimensional set (without explanatory text, too). Using the value option is not suitable, as the data type will be a parameter instead of a set (in addition, CSV2GDX expects numeric data, potentially leading to undefined values). We can read the explanatory text easily by specifying column three within the text option:

```
csv2gdx data.csv id=abWithExpText index=1,2 text=3 colCount=4
```

The figure shows the set abOnlyUELs on the left without explanatory text and the set abWithExpText on the right:

Note that the text and value(s) options cannot be used at the same time (instead, use multiple CSV2GDX calls in scenarios when you wish to read set elements with explanatory text and parameters from a single data set).

6.7 Eigenvalue

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This calculates the Eigenvalues of a symmetric positive definite matrix. The matrix `A` is indexed over `A(i,i)`. `AVal(i)` is indexed over `i`.

6.7.1 Usage

Command line:

```
gamstool [linalg.]Eigenvalue i A AVal gdxIn=fileIn.gdx gdxOut=fileOut.gdx
```

Compile time:

```
$callTool [linalg.]Eigenvalue i A AVal [gdxIn=fileIn.gdx] [gdxOut=fileOut.gdx]
```

Execution time:

```
executeTool '[linalg.]Eigenvalue i A AVal [gdxIn=fileIn.gdx] [gdxOut=fileOut.gdx]';
```

Where:

Argument	Description
`i`	Name of set used in matrix `i(*)`.
`A`	Name of two-dimensional matrix parameter `A(i,i)`.
`AVal`	Name of one-dimensional parameter to store the Eigenvalues `AVal(i)`.

The following parameters are available:

Parameter	Description
`gdxIn=fileIn.gdx`	Name of GDX file that contains symbols `i` and `A`. Mandatory if called from the command line, otherwise optional.
`gdxOut=fileOut.gdx`	Name of GDX file that contains symbol `AVal` after execution. Mandatory if called from the command line, otherwise optional.

6.7.2 Example

```
set i /i1*i3/;
alias (i,j);

table a(i,j)
         i1   i2   i3
   i1     9    1    1
   i2     1    9    1
   i3     1    1    9

;

parameter e(i) 'eigenvalues';

execute_unload 'a.gdx', i, a;
executeTool.checkErrorLevel 'linalg.eigenvalue i a e gdxin=a.gdx gdxout=b.gdx';
execute_load 'b.gdx', e;
abort$(abs(e('i1')-8)>1e-3 or abs(e('i2')-8)>1e-3 or abs(e('i3')-11)>1e-3) 'Wrong Eingenvalues', e;

option clear=e;
executeTool.checkErrorLevel 'linalg.eigenvalue i a e';
abort$(abs(e('i1')-8)>1e-3 or abs(e('i2')-8)>1e-3 or abs(e('i3')-11)>1e-3) 'Wrong Eingenvalues', e;
```

6.8 Eigenvector

Note

> This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This calculates the Eigenvalues and Eigenvectors of a symmetric positive definite matrix. The matrices `A` and `AVec` are indexed over `(i,i)`. `AVal(i)` is indexed over `i`.

6.8.1 Usage

Command line:

```
gamstool [linalg.]Eigenvector i A AVal AVec gdxIn=fileIn.gdx gdxOut=fileOut.gdx
```

Compile time:

```
$callTool [linalg.]Eigenvector i A AVal AVec [gdxIn=fileIn.gdx] [gdxOut=fileOut.gdx]
```

Execution time:

```
executeTool '[linalg.]Eigenvector i A AVal AVec [gdxIn=fileIn.gdx] [gdxOut=fileOut.gdx]';
```

Where:

Argument	Description
i	Name of set used in matrix i(*).
A	Name of two-dimensional matrix parameter A(i,i).
AVal	Name of one-dimensional parameter to store the Eigenvalues AVal(i).
AVec	Name of two-dimensional matrix to store the Eigenvectors AVec(i,i).

The following parameters are available:

Parameter	Description
gdxIn=fileIn.gdx	Name of GDX file that contains symbols i and A. Mandatory if called from the command line, otherwise optional.
gdxOut=fileOut.gdx	Name of GDX file that contains symbols AVal and AVec after execution. Mandatory if called from the command line, otherwise optional.

6.8.2 Example

```
Set i / i1*i5 /;

Alias (i,j);

Table a(i,j)
        i1   i2   i3   i4   i5
   i1    1    2    4    7   11
   i2    2    3    5    8   12
   i3    4    5    6    9   13
   i4    7    8    9   10   14
   i5   11   12   13   14   15
;

Table expected(i,*)
              val          i1          i2          i3          i4          i5
   i1   -8.464425   0.5550905  -0.2642556   0.2892854   0.6748602   0.2879604
   i2   -1.116317   0.4820641  -0.2581518   0.2196341  -0.7349311   0.3355726
   i3   -0.512109   0.2865066   0.2159261  -0.8437897   0.0411896   0.3970041
   i4   -0.027481  -0.0992784   0.7711236   0.3943678   0.0055409   0.4898525
   i5   45.120332  -0.6062562  -0.4714561  -0.0238286   0.0520829   0.6378888
;

Parameter
   eval(i)   'Eigenvalues'
   evec(i,j) 'Eigenvectors';

execute_unload 'a.gdx', i, a;
executeTool.checkErrorLevel 'linalg.eigenvector i a eval evec gdxin=a.gdx gdxout=b.gdx';
```

```
execute_load 'b.gdx', eval, evec;
evec(i,j)$(evec('i1','i1')<0) = -evec(i,j);
abort$(sum(i$(abs(eval(i)-expected(i,'val'))>1e-4),1)) 'Wrong Eigenvalue', eval, expected;
abort$(sum((i,j)$(abs(evec(i,j)-expected(i,j))>1e-4),1)) 'Wrong Eigenvector', evec, expected;

option clear=eval, clear=evec;
executeTool.checkErrorLevel 'linalg.eigenvector i a eval evec';
evec(i,j)$(evec('i1','i1')<0) = -evec(i,j);
abort$(sum(i$(abs(eval(i)-expected(i,'val'))>1e-4),1)) 'Wrong Eigenvalue', eval, expected;
abort$(sum((i,j)$(abs(evec(i,j)-expected(i,j))>1e-4),1)) 'Wrong Eigenvector', evec, expected;
```

6.9 ENDECRYPT

A tool to encrypt and decrypt text files.

6.9.1 Usage

```
endecrypt option xfile [passwdfile]
```

option

-W

to encrypt standard input and write to **xfile**

-R

to decrypt **xfile** and write to standard output

xfile

Text file to encrypt or decrypt.

passwdfile (optional)

First line is used as password; will be deleted after use

6.10 ExcelDump

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This tool writes all worksheets of an Excel workbook to GAMS symbols.

6.10.1 Usage

Command line:

```
gamstool [data.]ExcelDump excelFile gdxOut=fileOut.gdx
```

Compile time:

```
$callTool [data.]ExcelDump excelFile [gdxOut=fileOut.gdx]
```

Execution time:

```
executeTool '[data.]ExcelDump excelFile [gdxOut=fileOut.gdx]';
```

Where:

Argument	Description
excelFile	Excel workbook filename

The following named parameters are available:

Parameter	Description
gdxOut=fileOut.gdx	Name of GDX file that contains symbols s, r, c, w, ws, vf, vs, and vu. Mandatory if called from the command line, otherwise optional.

6.10.2 example

```
Set s            'Workbook sheets'
    r            'Rows'
    c            'Columns'
    w            'Workbook sheets by name'
    l            'Labels'
    ws(s,w)      'Workbook map'
    vs(s,r,c)    'Cells with explanatory text'
    vu(s,r,c,l<) 'Cells with potential GAMS label';
Parameter
    vf(s,r,c)    'Cells with numerical value';

$callTool.checkErrorLevel data.ExcelDump input.xlsx;
display ws, vs, vu, vf;
```

6.11 ExcelMerge

Note

> This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

Merges the sheets of the source Excel workbook into the destination workbook.

6.11.1 Usage

Command line:

```
gamstool [win32.]ExcelMerge source destination
```

Compile time:

```
$callTool [win32.]ExcelMerge source destination
```

Execution time:

```
executeTool '[win32.]ExcelMerge source destination';
```

Argument	Description
source	Source workbook.
destination	Destination workbook.

6.11.2 Example

A model utilitzing ExcelMerge can be found in a GAMS Connect example.

6.12 ExcelTalk

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

Performs command on an Excel workbook specified by filename.

6.12.1 Usage

Command line:

```
gamstool [win32.]ExcelTalk command filename [quit=0|1]
```

Compile time:

```
$callTool [win32.]ExcelTalk [-quit=0|1] command filename
```

Execution time:

```
executeTool '[win32.]ExcelTalk [-quit=0|1] command filename';
```

The following commands are recognized:

Command	Description
close	Close workbook ignoring changes.
open	Opens workbook.
saveAndClose	Ask Excel to save & close the workbook.

The following named parameter is available:

Argument	Description
`quit=0 or 1`	Determines if Excel program should be terminated or not (default: 0).

6.12.2 Example: Save and close an Excel workbook

```
$callTool win32.ExcelTalk saveAndClose myFile.xlsx
```

6.13 FINDTHISGAMS

6.13.1 Introduction

The Windows command line tool `findthisgams` is used for modifying GAMS specific registry entries created by the GAMS installer. These registry entries are primarily used for Windows file associations for GAMS related files (`*.gms`, `*.gdx`, `*.gpr`) and for the Object-oriented GAMS APIs which use the registry key `gams.location`. for finding a valid GAMS system directory. This documentation uses the following registry hive abbreviations:

- HKCU: HKEY_CURRENT_USER
- HKLM: HKEY_LOCAL_MACHINE
- HKCR: HKEY_CLASSES_ROOT

Attention

> This tool should be used with caution and only when necessary, as it changes the state of the registry. Reading this documentation carefully before use is advised.

6.13.2 Usage

```
findthisgams.exe [-q] [ list | delete (HKLM|HKCU) | write (HKLM|HKCU) ] [ide]
```

Without arguments: When `findthisgams.exe` is run without arguments, the tool tries to register the current GAMS system directory by either writing to `HKCU` (which takes precedence over `HKLM`) or by deleting entries in `HKCU` that hide the correct entries (current system directory) in `HKLM`.

-q: Run quietly. No pop-ups.

list: Print a summary of GAMS related registry entries (old and new).

delete (HKLM|HKCU): Deletes the registry entries in either `HKLM` or `HKCU`. The tool tries to delete both the old and the new style registry entries.

write (HKLM|HKCU): Writes the registry entries for the current GAMS system directory either to `HKLM` or `HKCU`.

ide: Per default, when `findthisgams.exe` writes to the registry, it uses GAMS Studio for file associations. Specifying `ide` will register the GAMS IDE instead.

6.13.3 Registry Keys

The registry entries created by the Windows installer have changed with GAMS 32. When running `findthisgams` with command line parameter `list`, the registy is queried for both the old and the new style registry keys. This gives a good overview of the current state of the registry before initiating any actual modifications to the registry:

Old registry keys (up to GAMS 31):

```
[HKCR|HKCU]\.gms
[HKCR|HKCU]\.gdx
[HKCR|HKCU]\gamside.file
[HKCR|HKCU]\gamside.file\shell\open\command
[HKCR|HKCU]\.gpr
[HKCR|HKCU]\gamside.project
[HKCR|HKCU]\gamside.project\shell\open\command
[HKCR|HKCU]\gams.location
```

New registry keys (GAMS 32 and later):

```
[HKLM|HKCU]\Software\Classes\.gms
[HKLM|HKCU]\Software\Classes\.gdx
[HKLM|HKCU]\Software\Classes\gamside.file
[HKLM|HKCU]\Software\Classes\gamside.file\DefaultIcon
[HKLM|HKCU]\Software\Classes\gamside.file\shell\open\command
[HKLM|HKCU]\Software\Classes\.gpr
[HKLM|HKCU]\Software\Classes\gamside.project
[HKLM|HKCU]\Software\Classes\gamside.project\DefaultIcon
[HKLM|HKCU]\Software\Classes\gamside.project\shell\open\command
[HKLM|HKCU]\Software\Classes\gams.location
```

Additional registry keys (GAMS 37 and later):

```
[HKLM|HKCU]\Software\Classes\.gsp
[HKLM|HKCU]\Software\Classes\gamsstudio.project
[HKLM|HKCU]\Software\Classes\gamsstudio.project\DefaultIcon
[HKLM|HKCU]\Software\Classes\gamsstudio.project\shell\open\command
```

Note

In order to access registry entries in `HKLM`, `findthisgams` needs to be run with administrative privileges.

6.14 GAMS Studio

GAMS Studio is a completely new integrated development environment for GAMS, which is available for Windows, macOS, and Linux. It is based on C++ and Qt.

Note

In addition to this technical documentation, there is also a tutorial about the usage of GAMS Studio.

6.14.1 Motivation

The classic GAMS IDE has been shipped with the GAMS system for the last 20 years and is still the workhorse for many GAMS programmers. However, the existing IDE does not provide all the features we see in modern development environments. Due to its underlying software technology, implementing new features in the current IDE turned out to be a poor option, so work on a new development environment started from scratch. GAMS Studio is based on C++ and Qt, which makes it fast, reliable, and platform independent (Windows, macOS, and Linux). An overview about the differences between GAMS Studio and the classic GAMS IDE can be seen at the end of this chapter (for long-time GAMSIDE users, especially the difference between IDE projects and Studio projects might be interesting).

6.14.2 Central Widgets

6.14.2.1 Welcome Page

The Welcome page is the starting point of GAMS Studio. It is designed to give quick access to common actions and to offer helpful information. It is divided into four columns.

The first two columns labeled "Last Projects" and "Last Files" list all projects and files that have been opened recently in Studio with the most recent on top. A simple left click on an item in this list opens the corresponding file. The top left of each entry shows an "x" to remove it. The third column named "Getting Started" offers useful actions and links for new users. The upper half contains shortcuts to create new files in user defined locations, open the GAMS Model Library Explorer or load the Transport example. The lower half has a link to the Studio introduction video on YouTube and two further links that open the integrated help view showing either this Studio documentation or the page with the GAMS tutorial overview. The rightmost column "Further Help" contains a link to the recent changes of GAMS Studio, the latest GAMS release notes, the GAMS World Forum for support and information about how to contact GAMS.

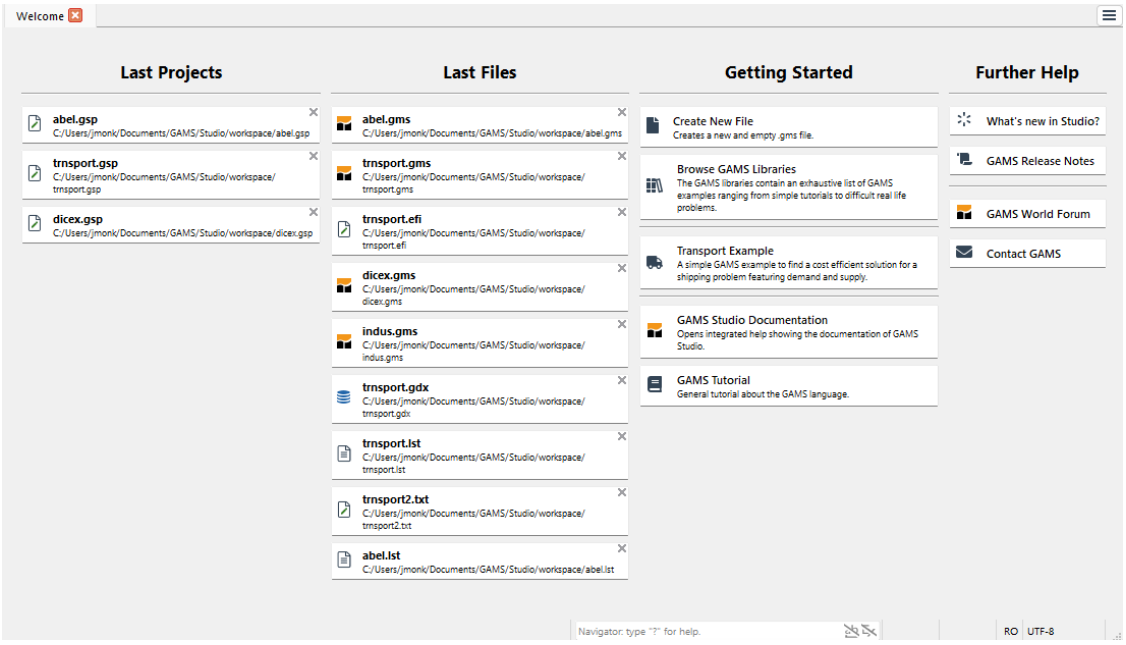


Figure 6.1 Welcome page

6.14.2.2 Code Editor

The Code Editor provides common functionality for editing with GAMS specific syntax highlighting. On the left side there is a special area displaying the line numbers and icons for links and errors generated by the compiler.

1. Basic Text Manipulation

Action	Shortcut	macOS	Description
Copy	**Ctrl - C**	**Command - C**	Copy selected text
Cut	**Ctrl - X**	**Command - X**	Cut selected text
Paste	**Ctrl - V**	**Command - V**	Paste text from clipboard
Select All	**Ctrl - A**	**Command - A**	Select whole document
Switch Comment	**Ctrl - ∗**	**Command - ∗**	(Un)Comment all lines in selection
Undo	**Ctrl - Z**	**Command - Z**	Reverts the latest text change
Redo	**Ctrl - Y** or **Ctrl - Shift - Z**	**Command - Shift - Z**	Restores a previous Undo

2. Advanced Text Manipulation

Action	Shortcut	macOS	Description
Remove Line	**Shift - Del**		Removes the current line
Duplicate Line	**Ctrl - D**	**Command - D**	Duplicates the current line(s)
Lowercase	**Alt - Shift - L**	**Option - Shift - L**	Toggle selection to lower case
Uppercase	**Alt - Shift - U**	**Option - Shift - U**	Toggle selection to upper case
Tab	**Tab**	**Tab**	Add spaces till next tab size
Untab	**Shift - Tab**	**Shift - Tab**	Remove spaces till previous tab size
Indent	**Ctrl - I**	**Command - I**	Indent complete line
Outdent	**Ctrl - Shift - I**	**Command - Shift - I**	Outdent complete line
Find	**Ctrl - F**	**Command - F**	Open Search & Replace window, filling search field with selected text if there is any.
Find Next	**F3**	**F3**	Jump to next search result if
Find Previous	**Shift - F3**	**Shift - F3**	Jump to previous search result

3. Navigation & Selection

Action	Shortcut	macOS	Description
Match Parentheses	**Ctrl - B**	**Command - B**	Jump to matching parenthesis
Select Parentheses	**Ctrl - Shift - B**	**Command - Shift - B**	Select to matching parenthesis
Toggle Bookmark	**Ctrl - M**	**Ctrl - M**	Toggle bookmark
Previous Bookmark	**Ctrl - ,**	**Ctrl - ,**	Go to previous bookmark
Next Bookmark	**Ctrl - .**	**Ctrl - .**	Go to next bookmark
Toggle Folding	**Alt - L**	**Alt - L**	Toggles the folding state of the current section
Fold all Sections	**Alt - O**	**Alt - O**	Folds all foldable sections
Fold DCO Sections	**Alt - I**	**Alt - I**	Folds all foldable DCO sections
Unfold all Sections	**Alt - Shift - O**	**Alt - Shift - O**	Unfolds all foldable sections

Action	Shortcut	macOS	Description
Open Code Completer	**Ctrl - Space**	**Ctrl - Space**	Opens the Code Completer
Jump to include	**Ctrl - Left Mouse Button** or **F2**	**Command - Left Mouse Button** or **F2**	Jumps to file referenced by $include
Scroll Up	**Ctrl - Up**	**Command - Up**	Scroll up (keeping cursor position)
Scroll Down	**Ctrl - Down**	**Command - Down**	Scroll down (keeping cursor position)
Scroll Page Up	**Ctrl - PageUp**	**Fn - Up**	Scroll page up (keeping cursor position)
Scroll Page Down	**Ctrl - PageDown**	**Fn - Down**	Scroll page down (keeping cursor position)
Start of Line	**Home**	**Command - Left**	Go to the start of line (or lines content)
End of Line	**End**	**Command - Right**	Go to the end of the line
Start of Document	**Ctrl - Home**	**Fn - Left**	Go to the start of the document
End of Document	**Ctrl - End**	**Fn - Right**	Go to the end of the document
Previous Word	**Ctrl - Left**	**Option - Left**	Go to previous start of word (or character class)
Next Word	**Ctrl - Right**	**Option - Right**	Go to next start of word (or character class)
Start/Change Block Edit	**Alt - Shift - Arrow Key**	**Option - Shift - Arrow Key**	Start/Change block edit selection
Start/Change Block Edit	**Alt - Left Mouse Button**	**Option - Shift - Left Mouse Button**	Draw block edit selection
Start/Change Block Edit	**Alt - Shift - Left Mouse Button**	**Option - Shift - Left Mouse Button**	Span block edit selection from text cursor to mouse click
Change Block Edit by Page	**Alt - PageUp / PageDown**	**Option - Fn - PageUp / PageDown**	Change block edit selection by a page step
Change Block Edit by Word	**Ctrl - Shift - Left / Right**	**Command - Shift - Left / Right**	Change block edit selection by word bounds
Change Block Edit by Word	**Ctrl - Left / Right**	**Command - Left / Right**	Change block edit position by word bounds (skip selection)
Change Block Edit Anchor	**Shift - Arrow Key**	**Shift - Arrow Key**	Change block edits anchor (start position)
Move Block Edit	**Arrow Key**	**Arrow Key**	Move block edit selection
Move Lines Up or Down	**Ctrl - Shift - Up / Down**	**Command - Shift - Up / Down**	Move line(s) up or down

The Code Editor looks for parenthesis to the right then to the left of the cursor and marks valid and invalid nesting. Hitting `Ctrl - B` moves the cursor to the matching parenthesis keeping the state inside/outside. To select the block to the matching parenthesis hit `Ctrl - Shift - B`.

To support navigating to files included by the gams source, place the mouse cursor over the file name and press the `Ctrl - Left Mousebutton` or place the cursor at the filename an hit `F2`. If found, Studio

will add the file to the current project and open it. Only plain text filenames can be detected this way, symbols are not resolved.

Code Folding

The Code Editor detects several kinds of sections in the source code that can be folded. These sections include all kinds of matching parentheses that extend over more than one line plus several $-control options that needs to come in pairs. To toggle the folding state of a section either the fold-marker can be clicked or the shortcut `Alt - L` can be pressed when the cursor is in a foldable line. To fold or unfold all foldable sections the main menu contains entries in the `View` menu. Additionally the shortcuts `Alt - O` to fold and `Alt - Shift - O` to unfold can be used.
To define individual blocks of folding the Dollar Control Options `$onFold` and `$offFold` can be added to the source.

List of $-control options that create a fold section:

- `$OnText` / `$OffText`
- `$OnEmbeddedCode[V/S]` / `$OffEmbeddedCode`
- `$OnEcho[V/S]` / `$OffEcho`
- `$OnPut[V/S]` / `$OffPut`
- `$OnExternalInput` / `$OffExternalInput`
- `$OnExternalOutput` / `$OffExternalOutput`
- `$ifThen[I/E]` / `$endIf`
- `$onFold` / `$offFold`

Syntax Highlighting

The GAMS Studio syntax highlighter recognizes different kinds of source code like

- **$control option** with some special treatments (like `$OffText`, `$OnText`, etc.)
- **comments**, e.g. `line comments` and `comment block`
- **keywords** like `declarations` or `loops`
- **description** as `descriptive text` for `identifiers`
- **data statements** for `identifiers` or `table data`
- ...

To get the best highlight result it is recommended to use strict GAMS syntax.

Code Completion

GAMS Studio offers a static code completion for fixed internal keywords known by the compiler. The code completer pops up after pressing `Ctrl - Space` and shows keywords that matches the current syntax and starts with the already typed characters of the word to the left of the cursor. Moving the cursor to the left shortens the matching pattern while moving the cursor to the right extends it. The keys `up`/`down`, `pageUp`/`pageDown`, and `home`/`end` are processed by the results list during the completer is visible. The selected word can be inserted with the keys `tab` or `enter`/`return` or by double clicking a word in the list. To hide the completer, hit `esc` or click outside of the completer window. Additionally the completer hides if there is no match for the current start of word.

Automatic closing of brackets and quote characters

This feature is meant to improve convenience for users typing source code. In GAMS, as in pretty much every other programming language, opening brackets or quote characters are followed at some point by a closing equivalent. So when a user types one of the following characters the matching closing character will also be inserted after the text cursor. This allows the user to easily type the content in between. When typing the closing character which should be right after the cursor the already existing character will be jumped over. So when a user types the closing character by force of habit the autoclose feature does not interfere or causes syntax errors. Also possible is to make a text selection and then type a bracket or quote character. The selected text will then be surrounded by the chosen character pair. In the GAMS Studio under **Settings** > **Editor** > **"Auto close opening brackets and quotes"** a switch is available to turn this auto-close feature off.

List of characters that trigger insertion of a closing character:

- (
- {
- [
- "
- '

Other editors also close the < character but in the GAMS language context it is used in a different way, making it unnecessary to be closed automatically (e.g. as a comparison operator). In some cases user might not like the closing character to be inserted automatically however, this is hard to predict. Other editors - like Qt Creator - check what the next character after the cursor is. Only a limited set of characters allows the automatic insertion of the closing character.

Following characters allow auto closing when being the next char after the text cursor:

- **Whitespaces**
- ,
- ;
-)
-]
- {
- }

All others will prevent the insertion of characters.

File Encoding

When opening a file Studio tries to assume the valid file-encoding. As it is not possible to safely determine the encoding the **Edit** > **Encoding** menu assists. To reload a file using another codec the **reload with ...** submenu can be used. With the **convert to ...** submenu the current file can be converted to another codec. A selection of encodings is preselected. This selection can be changed using the **Select encoding** submenu. It opens a dialog where the active encodings can be selected. One of them is marked as the `Default` encoding. Initially is is set to **utf-8**.

Bookmarks

The `Code Editor` provides the ability to set temporary `bookmarks` in the text files opened in main tabs. If Studio is closed they are dropped. Also if a file tab is closed the bookmarks for this file are dropped. A bookmark is placed at the current position of the current file using the `toggle bookmark` key. A line in the editor can only carry one bookmark. If this line has a bookmark already the existing one is removed. The keys for next/previous bookmark allow to navigate through the bookmarks beyond file bounds.

6.14.2.3 Listing Viewer

The Listing Viewer is used for displaying a GAMS output or listing file (`*.lst`) which is generated when running a GAMS (`.gms`) file. The Listing Viewer consists of two sides. On the left side, the content of the listing file is presented in a tree structure that allows for convenient navigation within the listing file. The right side shows the listing file itself.

By clicking on a specific item in the tree on the left hand side, the listing file will automatically jump to the corresponding location. Changes of the cursor position in the listing file are dynamically reflected in the tree by selecting the item that belongs to the current section of the listing file.

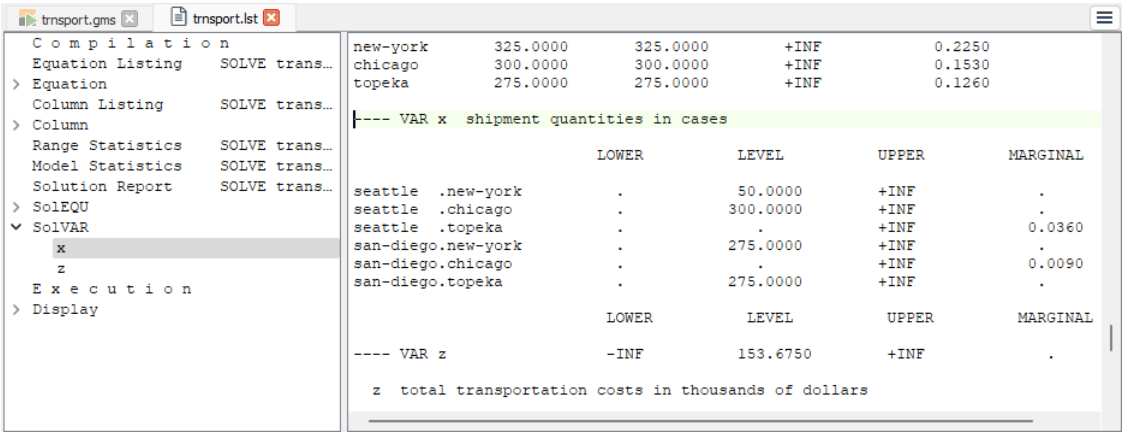


Figure 6.2 Listing Viewer showing trnsport.lst generated by running trnsport.gms from the GAMS Model Library

When the GAMS process returns with a compilation error the `Listing Viewer` contains links to the source file. If the source file doesn't exist anymore the link is still shown but striked out.

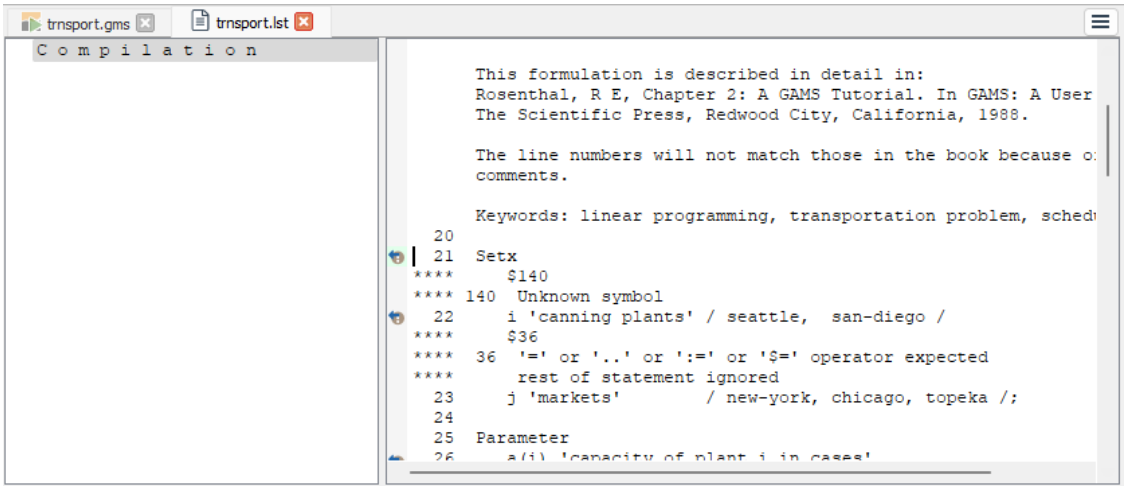
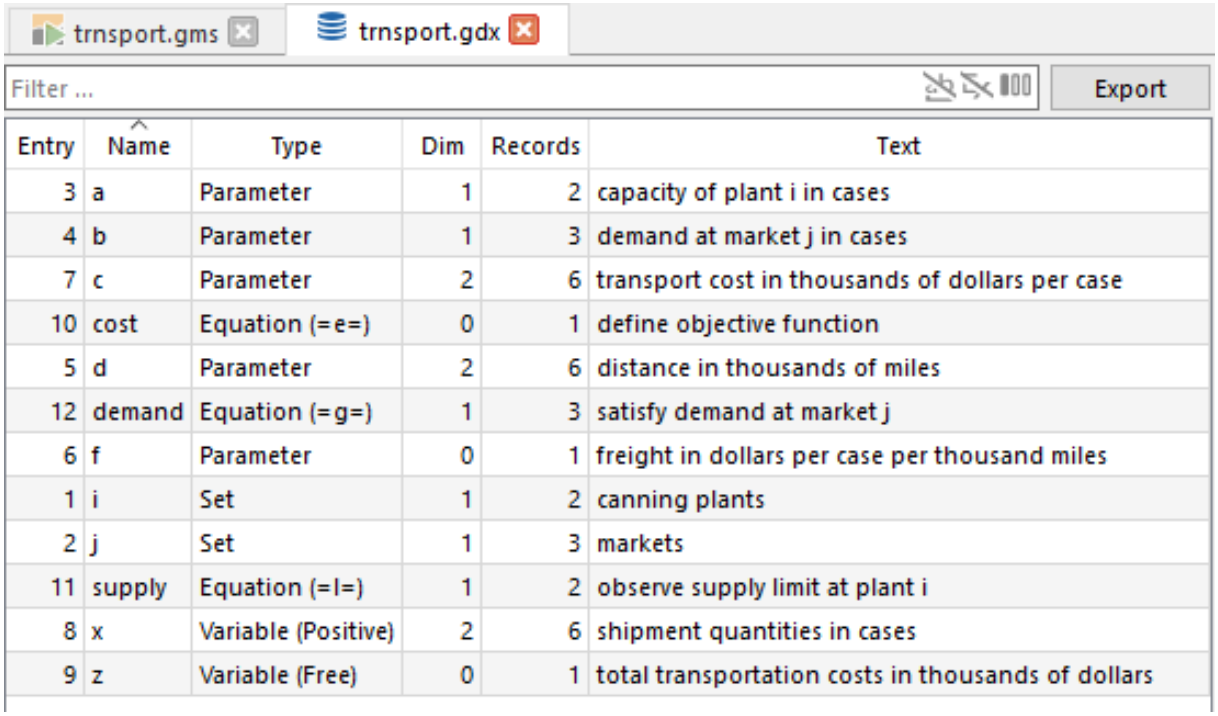


Figure 6.3 Listing Viewer showing trnsport.lst generated by running a modified trnsport.gms

6.14.2.4 GDX Viewer

The GDX Viewer is used to open and inspect GDX (GAMS Data eXchange) files. A GDX file contains GAMS symbols such as sets, parameters, variables, and equations in a binary format that serves as input and output to GAMS. The GDX Viewer component consists of two sides. The left side displays the symbol table of the open GDX file in a table format. Every entry represents a symbol and consists of the following information:

- **Entry:** The index of the symbol
- **Name:** The identifier of the symbol
- **Type:** The type of the symbol (Alias, Equation, Parameter, Set, or Variable) including the sub type if available
- **Dim:** The dimension of the symbol
- **Records:** The number of records of the symbol
- **Text:** The explanatory text of the symbol



Entry	Name	Type	Dim	Records	Text
3	a	Parameter	1	2	capacity of plant i in cases
4	b	Parameter	1	3	demand at market j in cases
7	c	Parameter	2	6	transport cost in thousands of dollars per case
10	cost	Equation (=e=)	0	1	define objective function
5	d	Parameter	2	6	distance in thousands of miles
12	demand	Equation (=g=)	1	3	satisfy demand at market j
6	f	Parameter	0	1	freight in dollars per case per thousand miles
1	i	Set	1	2	canning plants
2	j	Set	1	3	markets
11	supply	Equation (=l=)	1	2	observe supply limit at plant i
8	x	Variable (Positive)	2	6	shipment quantities in cases
9	z	Variable (Free)	0	1	total transportation costs in thousands of dollars

Figure 6.4 Symbol table showing trnsport.gdx generated by running trnsport.gms from the GAMS Model Library

A specific column can be sorted by clicking on the respective column header. Clicking again toggles the sorting direction. The `Filter` facility provides dynamic filtering of the list of symbols contained in the GDX file. By default, only the `Name` column is taken into account by the filter. Clicking on the `Export` button will open the export dialog. The right side of the GDX Viewer displays the actual data of the symbol that is selected in the symbol table. Data can be displayed in either the List View or the Table View which can be toggled using the corresponding button in the upper part of the GDX Viewer.

The GDX Viewer automatically saves and restores its state on closing and as soon as data is reloaded due to changes to the underlying GDX file. Changes to the symbol table like an applied filter or the currently

selected symbol are kept. Individual symbols keep applied filters as well as the view settings (e.g. the currently selected view (List View or Table View), preferences and the visible attributes for variables and equations) as long as the name, type, and dimension of the symbol did not change during the update of the GDX file. Resetting the state of a symbol can be achieved by clicking on the `Reset` button. For each GDX file known by the Project Explorer the state is stored in the Settings. The context menu of the Project Explorer provides an entry to reset the GDX Viewer state for that file. Removing the GDX file from the project also clears its state.

Note

The GDX Viewer currently supports a maximum number of ∼107 million records to be shown for an individual symbol. As soon as a specific symbol reaches this limit, a warning is added to the system log and records exceeding this limit will be truncated from the view. Also a note about truncated data is displayed in the GDX Viewer itself. Applying filters might reduce the number of records to be displayed and as soon as filtered data is shrunk enough, the whole (filtered) data can be displayed again and the note gets removed.

List View

The `List View` is the default representation when looking at data using the GDX Viewer. The data is presented in the form of a table in which each row represents a record of the symbol. Each record consists of key columns corresponding to the symbols dimension. The value columns vary according to symbol type:

- **Set/Alias**: Column `Text` contains the explanatory text of a record, `Y` if no explanatory text is available.
- **Parameter:** Column `Value` contains the numerical value of a record.
- **Variable/Equation**: Five numerical columns: (`Level`, `Marginal`, `Lower`, `Upper`, `Scale`).

i [1]	j [2]	Value
seattle	new-york	2.5
seattle	chicago	1.7
seattle	topeka	1.8
san-diego	new-york	2.5
san-diego	chicago	1.8
san-diego	topeka	1.4

Figure 6.5 Right side of GDX Viewer showing the parameter `d` from the `trnsport` model

The header displays the individual domains as well as the original index position using superscript notation which is especially useful if the column order has been changed. Furthermore it allows to manipulate the displayed data in several ways. By left clicking on a column header, the corresponding column is sorted using a stable sort mechanism that does not change the order of equal entries. Clicking again changes the sorting direction. Due to the Extended Range Arithmetic used by GAMS, columns containing numerical data can contain special values. Those values are treated in a special way when being sorted:

- **-INF:** Smallest numerical value.
- **EPS:** Treated as value very close but different from 0.
- **+INF:** Largest numerical value.
- **NA:** Treated as first non-numerical value (sorted as being greater than +INF).
- **UNDEF:** Sorted as being greater than `NA`.
- **ACRONYMS:** Sorted as being greater than `UNDEF`. Relative order of acronyms is by their internal number.

The order of columns can be changed by dragging a column header and dropping it at a different position. Dragging the border between column headers allows for resizing the widths of the involved columns. The columns in the `List View` offer a filter facility which can be opened either by right clicking on the column header or by left clicking on the filter icon. The `Reset` button in the top right corner can be used to reset the view to defaults. This removes all filters, resets the applied sorting as well as the applied numerical format and switches back to the `List View` in case the `Table View` was active.

The button `Preferences` opens a menu containing different settings for the current symbol. `Squeeze Defaults` can be used on variables and equations in order to hide all columns that have the default value of the respective variable or equation type. `Squeeze Trailing Zeroes` allows to turn on/off the truncation of trailing zeroes. The displaying format of numerical values can be specified using the `Format` drop down menu and the `Precision` spin box.

The available options for the `Format` are as follows:

- **g-format**: The display format is chosen automatically: `f-format` for numbers closer to one and `e-format` otherwise. The value in the `Precision` spin box specifies the number of significant digits. When precision is set to `Full`, the number of digits used is the least possible such that the displayed value would convert back to the value stored in GDX. Trailing zeros do not exist when `precision=Full`.

desk [1]	Level	Marginal	Lower	Upper	Scale
d1	1333.33	0	0	+INF	1
d2	0	-6.66667	0	+INF	1
d3	0	-3.33333	0	+INF	1
d4	66.6667	0	0	+INF	1

Figure 6.6 Variable `mix` from the `prodmix` model using `g-format` with `precision=6`

desk [1]	Level	Marginal	Lower	Upper	Scale
d1	1333.3333333333333	0	0	+INF	1
d2	0	-6.666666666666665	0	+INF	1
d3	0	-3.3333333333333317	0	+INF	1
d4	66.66666666666666	0	0	+INF	1

Figure 6.7 Variable `mix` from the `prodmix` model using `g-format` with `precision=Full`

- **f-format**: Values are displayed in fixed format as long as appropriate. Large numbers are still displayed in scientific format. The value in the `Precision` spin box specifies the number of decimals.

desk [1]	Level	Marginal	Lower	Upper	Scale
d1	1333.333333	0	0	+INF	1
d2	0	-6.666667	0	+INF	1
d3	0	-3.333333	0	+INF	1
d4	66.666667	0	0	+INF	1

Figure 6.8 Variable `mix` from the `prodmix` model using `f-format` with `precision=6`

- **e-format**: Values are displayed in scientific format. The value in the `Precision` spin box specifies the number of significant digits. When precision is set to `Full`, the number of digits used is the least possible such that the displayed value would convert back to the value stored in GDX. Trailing zeros do not exist when `precision=Full`.

desk [1]	Level	Marginal	Lower	Upper	Scale
d1	1.33333e+03	0e+00	0e+00	+INF	1e+00
d2	0e+00	-6.66667e+00	0e+00	+INF	1e+00
d3	0e+00	-3.33333e+00	0e+00	+INF	1e+00
d4	6.66667e+01	0e+00	0e+00	+INF	1e+00

Figure 6.9 Variable `mix` from the `prodmix` model using `e-format` with `precision=6`

The value in the `Precision` spin box specifies the number of decimals or the number of significant digits depending on the chosen format:

- **g-format**: Significant digits [1..17, Full]
- **f-format**: Decimals [0..14]
- **e-format**: Significant digits [1..17, Full]

Next to Preferences is the `Attributes` button. This button opens a menu that allows to hide certain attribute columns of variables and equations explicitly. This is especially useful when using the `Table View`.

Table View

Switching to the `Table View` is possible for symbols that have at least two dimensions. This mode consists of two parts: A separate header view that shows the domains and attributes of the symbol as well as the actual data. The header view also allows to access the filter facility and highlights the positions of specific domains by using visual separators (from left to right: rows, columns, attributes). The data itself is reshaped into a format where the labels of the last dimension are moved into the column header while

the remaining dimensions go into the row header. The entries in the table are the numerical values (or explanatory text in case of GAMS sets) of the record with the corresponding labels of the headers. The following image shows the `List View` of a GAMS Parameter on the left side and its representation in the `Table View` on the right side:

i [1]	j [2]	Value
seattle	new-york	2.5
seattle	chicago	1.7
seattle	topeka	1.8
san-diego	new-york	2.5
san-diego	chicago	1.8
san-diego	topeka	1.4

i [1]	j [2]	Value

	new-york	chicago	topeka
seattle	2.5	1.7	1.8
san-diego	2.5	1.8	1.4

Figure 6.10 Two dimensional parameter as List View and Table View

In case of GAMS variables and equations, a virtual dimension for the five numerical values (Level, Marginal, Lower Bound, Upper Bound, Scale) is introduced and displayed in the column header. When looking at the data of variables or equations it is most of the time useful to visualize only a specific attribute like the variable levels. This can be achieved by using the `Attributes` menu in the upper part of the GDX Viewer. The menu allows to hide specific attributes explicitly by disabling the corresponding checkboxes.

Both column and row header dimensions can be moved into different positions in order to reshape the table view. This can be done by performing a drag-and-drop operation that starts by clicking (and holding) the left mouse button while the mouse cursor is located on the index that should be moved. Moving the mouse to an arbitrary location will bring up a drop indicator (in the form of a thin line) that previews the position in which the index will be dropped as soon as the mouse button is released. In general all index positions can be moved to arbitrary new locations, but there are a few restrictions:

- The (virtual) numerical dimension of variables and equations (Level, Marginal, Lower Bound, Upper Bound, Scale) can not be moved and is always the last dimension in the column header.

- If a header (column or row) runs out of dimensions, a dummy header (Value, Text) is introduced which can not be moved but can be used as a drop target for further drag-and-drop operations.

s 4	t 3	z 2	m 6	w 5	c 1	Value						
			jan									
			standard									
			rab-fod	gram		mus+rap	sc-mill	sc-gur	wheat	orchard		
standard	bullock	nwfp	0.2	0.1			0.19	0.19	0.22	0.08		
		pmw	0.2	0.05		0.2	0.14	0.14	0.22	0.1		
		pcw	0.2	0.1		0.2	0.09	0.09	0.23	0.11		
		psw	0.2	0.1		0.2	0.09	0.09	0.23	0.08		
		prw	0.2	0.1		0.2	0.14	0.14	0.23	0.08		
		scwn	0.21	0.11		0.21	0.25	0.25	0.2	0.17		
		srwn	0.2	0.1		0.2	0.23	0.23	0.19	0.15		
		scws	0.27	0.11		0.11	0.25	0.25	0.25	0.18		
		srws	0.27	0.1		0.2	0.25	0.25	0.2	0.19		
	semi-mech	nwfp	0.2	0.1			0.19	0.19	0.22	0.08		
		pmw	0.2	0.05		0.2	0.14	0.14	0.22	0.1		
		pcw	0.2	0.1		0.2	0.09	0.09	0.23	0.11		
		psw	0.2	0.1		0.2	0.09	0.09	0.23	0.08		

Figure 6.11 The table view representation of a GAMS parameter during a drag-and-drop operation that will move the dragged dimension into the position indicated by the drop indicator (thin black line)

While filters that have been applied in the `List View` will also be applied to the `Table View` and vice versa, sorting does not have any effect. The labels are sorted using the internal order which can not be changed by the user. For large data it is recommended to reduce the amount of visible records by applying filters in the `List View` and by disabling certain value columns in case of variables and equations. Once the data has been reduced, switching to the `Table View` allows to look at the data in a compact format.

Filter Facility

Filters can be applied in both the List View and the Table View through the header view that contains the domains as well as the (numerical) attributes. In the List View, the header is part of the data view itself, while the Table View offers a separate header view above the actual data. The filter dialog can be opened either by right clicking on the header or by left clicking on the filter icon. Depending on the type of the column (key or value), the appropriate filter dialog for labels or numerical values is opened.

The filter dialog for a specific key column shows all occurring labels of that column in a list. Per default all labels in all columns are visible. Shrinking the displayed data can be achieved by unchecking one or more labels. The filter is applied by clicking the `Apply` button. Instead of selecting labels manually one can also use the filter in the upper part of the filter dialog. More about the filter can be found in the Filter Section. The filter is automatically applied to the list of labels. Matching labels will be selected while all others will be deselected. The `Hide unselected items` checkbox can be used to automatically hide all deselected labels. This is especially useful for larger amounts of labels in combination with the filter as only the matching labels will remain in the list while typing. The `Select All`, `Invert`, and `Deselect All` buttons speed up getting the desired selection. Using `middle-click` or `Ctrl` - `left-click`(`Command` - `left-click` on macOS) allows to quickly select one specific label while all others are deselected. The filter gets applied automatically and the dialog closes without the need to click the `Apply` button. Sometimes it is convenient to select/deselect a range of labels. This is possible by selecting a certain label first and using `Shift` -

`left-click` on another label afterwards. This will change the selection state of all labels in the specific range to the same state. If no label has been selected in the first step, the range starts at the first label by default.

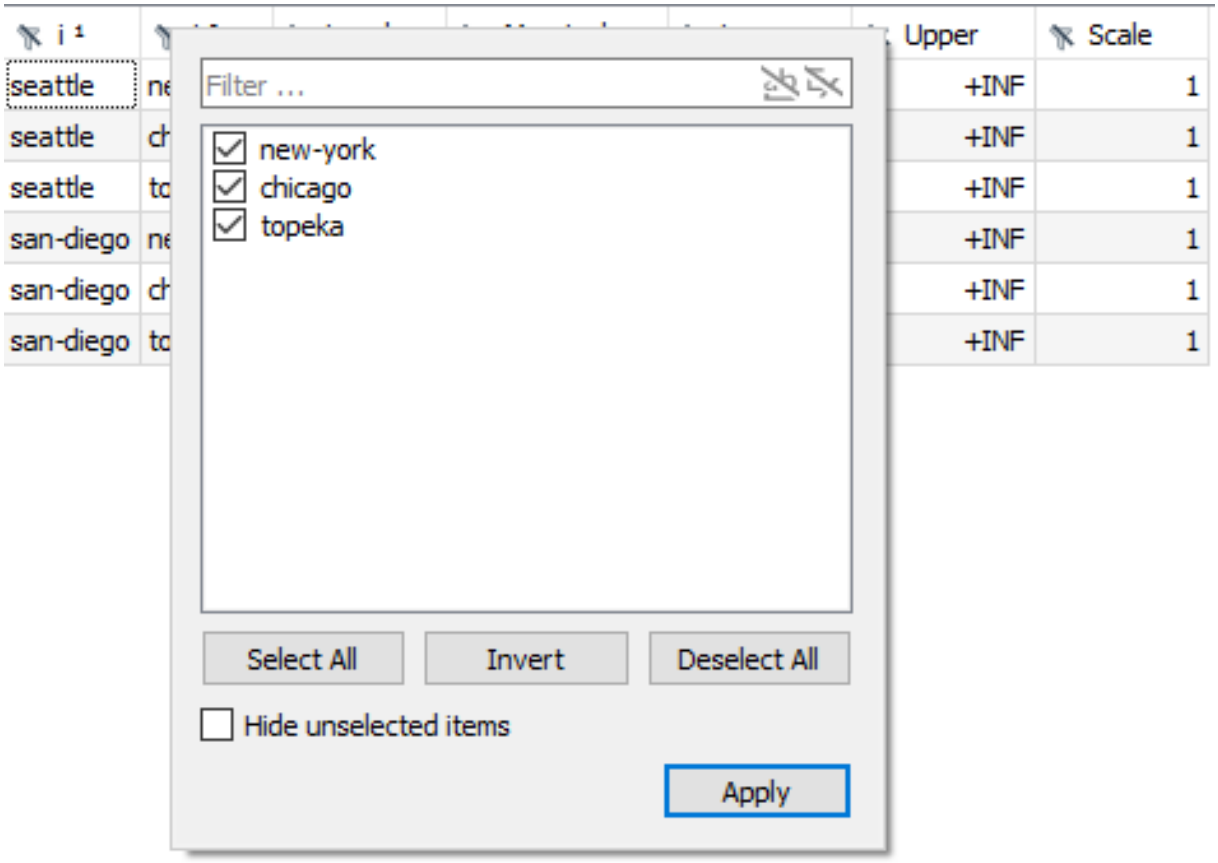


Figure 6.12 Filter dialog on column `j` for variable `x` in trnsport.gdx

Numerical columns offer a different filter type that allows to specify an interval for values that should be included or excluded in the view for this specific column. The `Min` value is the lowest and the `Max` value is the largest one still included in the view. When the `Exclude` check box is activated, the specified range is excluded from the view instead of included. The lower part of the filter contains check boxes for special values like `+INF` or `EPS`. Checking/unchecking will show/hide the corresponding values in the view. Clicking the `Apply` button adds the filter to the column and refreshes the view accordingly. The `Reset` button removes an active filter.

Note

When a filter on a numerical column is applied, the values used for comparison are the actual values from the GDX file. Since the displayed values are influenced by the `Format` and the `Precision` setting, they can differ from the internal ones from the GDX file. Here is an example: Let's assume we have a parameter `p(i)` with `p("i1")=1.34` and `p("i2")=2`. With format set to `g-format` and precision set to `2`, the values are displayed as `1.3` and `2`. When applying a filter with the bounds (`Min`, `Max`) set to `1.33` and `2` both records are still shown even though `1.33` is greater than `1.3` (number shown in the GDX Viewer) since the internal GDX value used for comparison is actually `1.34`.

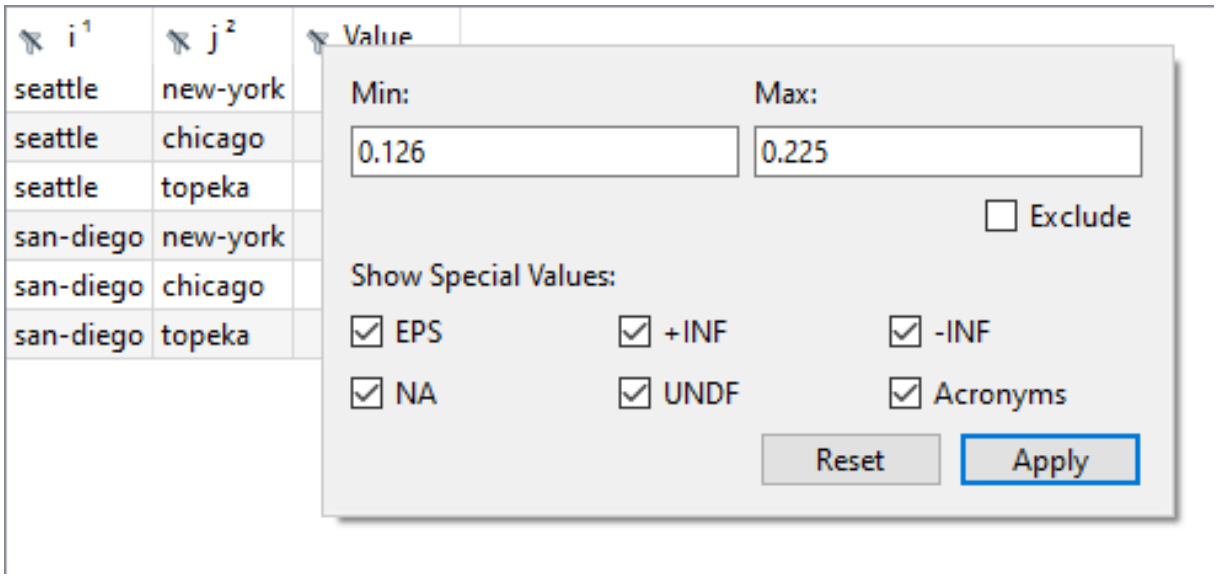


Figure 6.13 Filter dialog on numerical column `Value` for parameter `c` in trnsport.gdx

Search Facility

Both the List View and the Table View can be searched by adding a term into the search field which is located above the Data View. The search behavior can be customized using the buttons on the right hand side of the line edit. See Filter in Tables and Trees for more information. As soon as the content of the search field changes all matches in the Data View are highlighted automatically. The `Search Forward` (**F3**) and `Search Backward` (**Shift** - **F3**) buttons allow to select the next/previous element that matches the search criteria. Search is continued at the beginning of the Data View as soon as the end is reached and vice versa. In the List View, all cells except for the header are considered while in the table view all cells and all header entries are considered with the exception of symbol type specific extra headers which is `Value` (parameters), `Text` (sets/aliases), `Level`, `Marginal`, `Lower`, `Upper`, `Scale` (variables/equations).

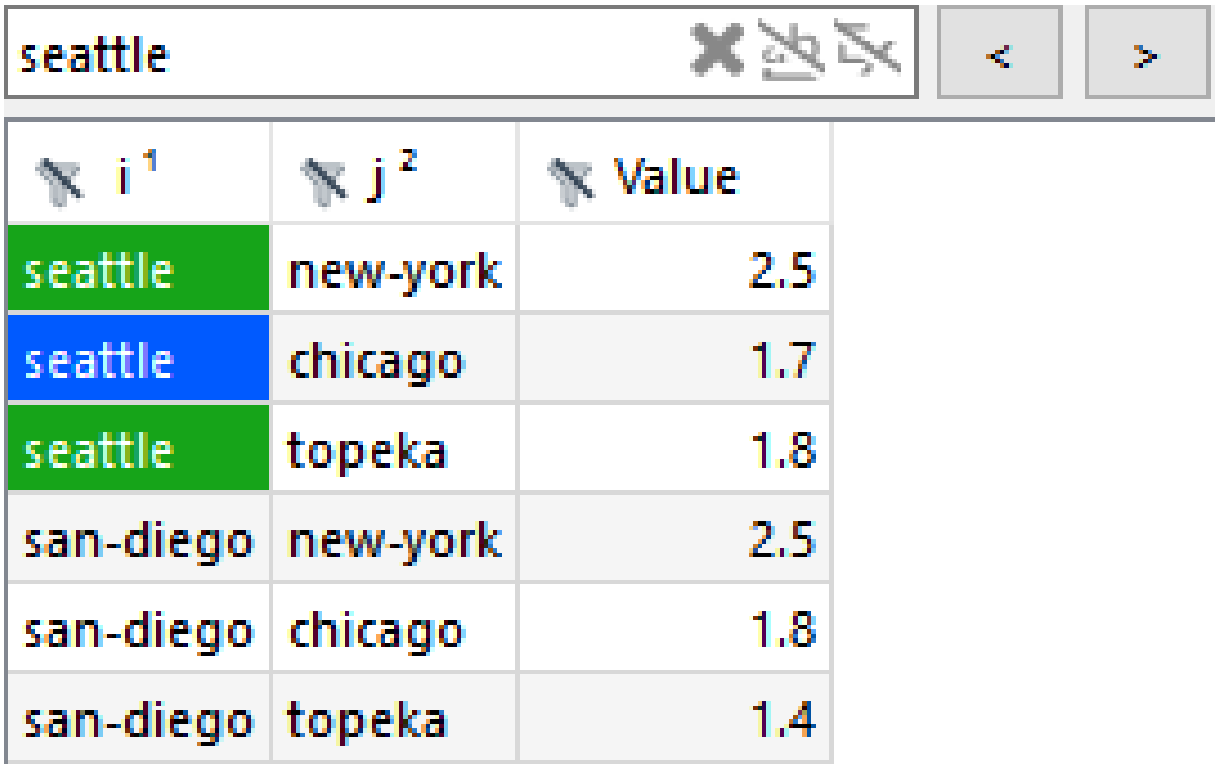


Figure 6.14 Searching the List View of a parameter by 'seattle'

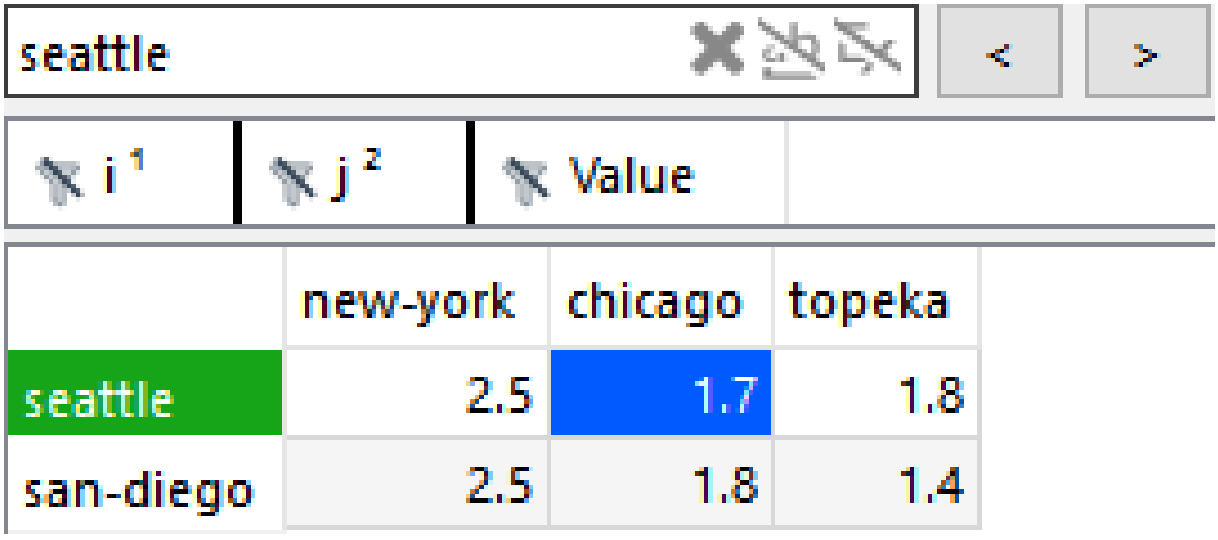


Figure 6.15 Searching the Table View of a parameter by 'seattle'

Export Dialog

The GDX Viewer offers an export functionality that can be accessed by clicking on the `Export` button above the symbol table on the left side of the GDX Viewer. The export dialog uses GAMS Connect in order to export data from a GDX file into another data format.

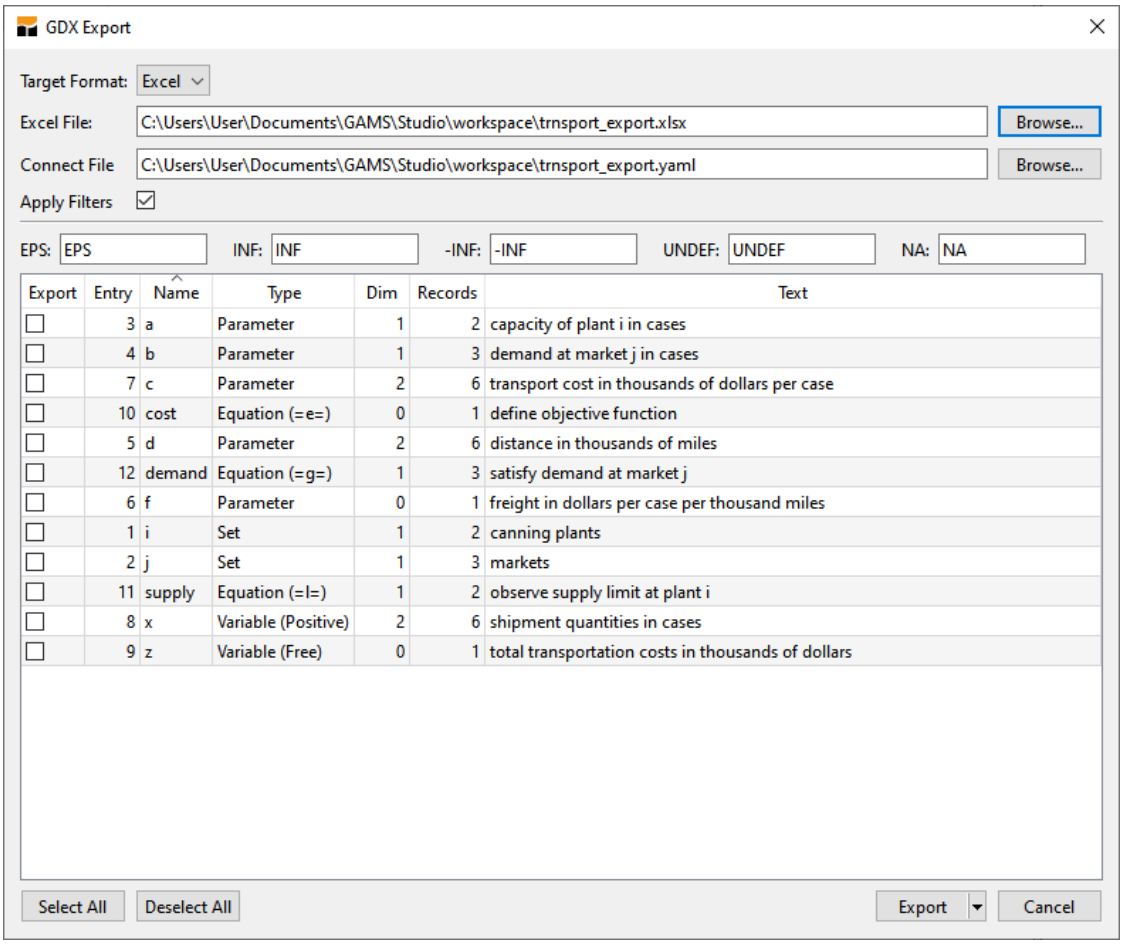


Figure 6.16 GDX Export dialog opened from trnsport.gdx

The `Target Format` drop down menu allows to select the format in which the data is exported (currently only Excel is available as target format). The upper part of the dialog also allows to specify the Excel output file as well as the GAMS Connect instructions file in YAML format. The `Apply Filters` checkbox controls if active filters in the GDX Viewer are applied to the exported data or not. The input fields for the representation of GAMS special values allow to customize those. A common use case is the export of GAMS special value `EPS` as `0` instead of the string representation `EPS` (default) which can be achieved by changing the corresponding input field to `0`. Symbols can be selected for export using the checkboxes in the `Export` column of the symbol table. The button `Select All/Deselect All` can be used in order to select/deselect all symbols for export. The `Export` button generates the GAMS Connect instructions file and executes it. Choosing `Save` instead will only generate the GAMS Connect instructions file but won't run it. As long as the export process is running the dialog will be disabled. Clicking the `Cancel` button closes the dialog and stops the export process that has not been finished yet.

The export mechanism uses the underlying GDX file that is opened in the GDX Viewer. If a symbol is currently in `table view` mode, the order and location of individual dimensions are reflected in the Excel output.

Summary of Actions and Shortcuts

Action	Shortcut	macOS	Description
Jump to Symbol Search/Data View Search	**Ctrl - F**	**Command - F**	Moves the focus either to the Symbol Search input field located above the symbol table or the Search input field of the Data View, depending on the current focus.
Find next	**F3**	**F3**	Jumps to the next match when searching the Data View
Find previous	**Shift - F3**	**Shift - F3**	Jumps to the previous match when searching the Data View
Select All	**Ctrl - A**	**Command - A**	Selects all data (right side only)
Copy (comma-separated)	**Ctrl - C**	**Command - C**	Copy selection to clipboard using comma as separator
Copy (tab-separated)	**Ctrl - Shift - C**	**Command - Shift - C**	Copy selection to clipboard using tab as separator (right side only)
Copy Without Labels (comma-separated)	context menu only	context menu only	Copy selection without labels to clipboard using comma as separator (Table View only)
Copy Without Labels (tab-separated)	context menu only	context menu only	Copy selection without labels to clipboard using tab as separator (Table View only)
Auto Resize Columns	**Ctrl - R**	**Command - R**	Resizes all columns to content (right side only)

6.14.2.5 Reference File Viewer

The Reference File Viewer is a useful tool to navigate the source code of GAMS models via a reference file, especially when multiple files are involved. A reference file contains all symbol references of GAMS model and is created using the rf parameter when running the model.

Figure 6.17 Parameter to create a symbol reference file when running the model

When the model is compiled, the reference file with ".ref" extension is created and can be opened either using the menu: `File` > `Open` or double-clicking the "RefFile" entry in the Process Log:

```
--- Starting compilation
--- trnsport.gms(66) 3 Mb
--- RefFile /home/jeed/Documents/GAMS/Studio/workspace/trnsport.ref
--- Starting execution: elapsed 0:00:00.002
--- trnsport.gms(43) 4 Mb
--- Generating LP model transport
--- trnsport.gms(64) 4 Mb
```

Figure 6.18 Process Log showing clickable entry for the reference file

The Reference Viewer is categorized by a number of tabs:

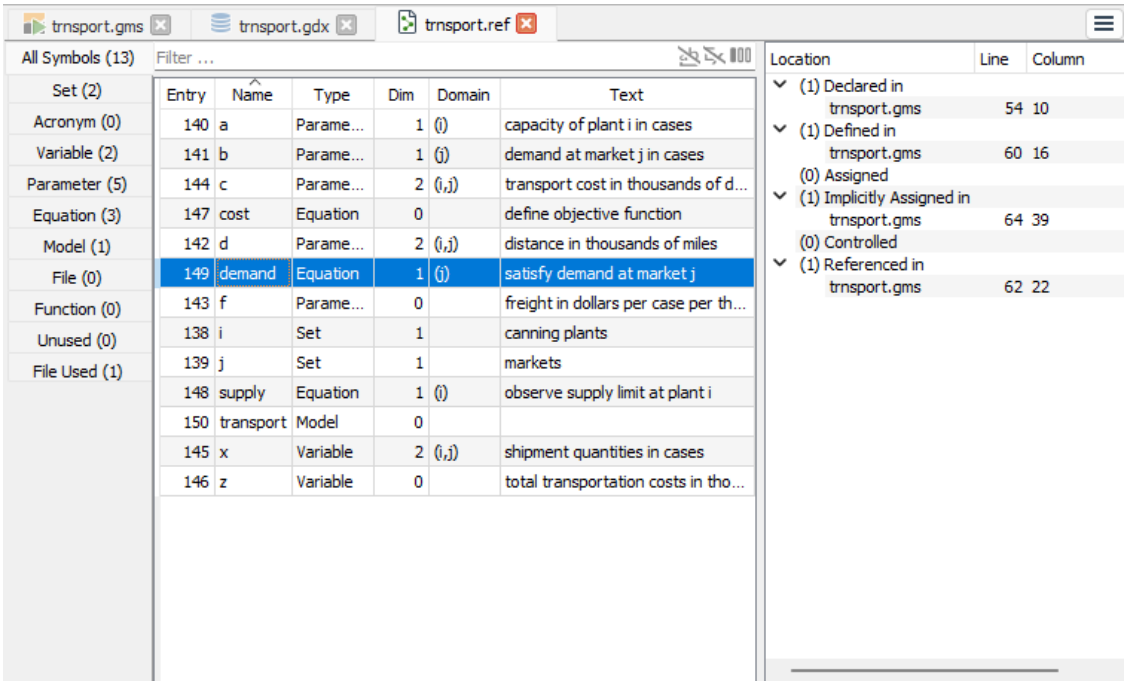


Figure 6.19 Reference File Viewer showing tabs and reference file information

- **All Symbols** shows an alphabetical listing of all symbols used in the model.
- **Set, Acronym, Variable, Parameter, Equation, and Model** shows an alphabetical listing of symbols of the same type. See Data Types and Definitions for more details on GAMS data types.
- **File** shows an alphabetical listing of all file statements used in the model.
- **Function** shows a list of GAMS built-in functions that has been used in the model.
- **Unused** shows an alphabetical listing of all symbols that have been declared in the model but are not used anywhere.
- **File used** shows an alphabetical listing of all files used in the model including the full file path. A double-click on an file entry of the table will jump to the beginning of the reference file.

The number next to the tab name indicates how many symbols in the category, where the number of All Symbols tab sums up the number of symbols in all other tabs except Unused tab and File used tab. A click on a symbol entry of the table in every tab (except File used tab) will show the detailed reference list containing the reference location in the right hand side of the reference viewer. A mouse-over on an entry of the reference list will show the tooltip of file location and position with the file that has been referenced. A double-click on an entry of the reference list will jump to the position in the corresponding file that has been referenced.

The reference list is organized by the following reference types:

Reference Type	Description
`declared`	Declaration of the symbol.
`defined`	Defined using a data statement.
`assigned`	When the symbol appears on the left side of an assignment statement.
`implicitly assigned`	Implicit assignment like a variable in a model.
`control`	When the set is used as a control set.
`reference`	Referenced in a statement.

The detailed description of the reference types and their shorthand symbols can be found in Reference Types in GAMS Output.

Sort and Filter referenced symbols

- **Sort** : A click on a table column header in the reference table will sort the symbol in the table by either Entry, Name, Type, Dim(ension), Domain, or Text.
- **Filter** : An input in the `Filter box` will filter the symbols in the reference table by `Name`. More about the filter can be found in the Filter Section.

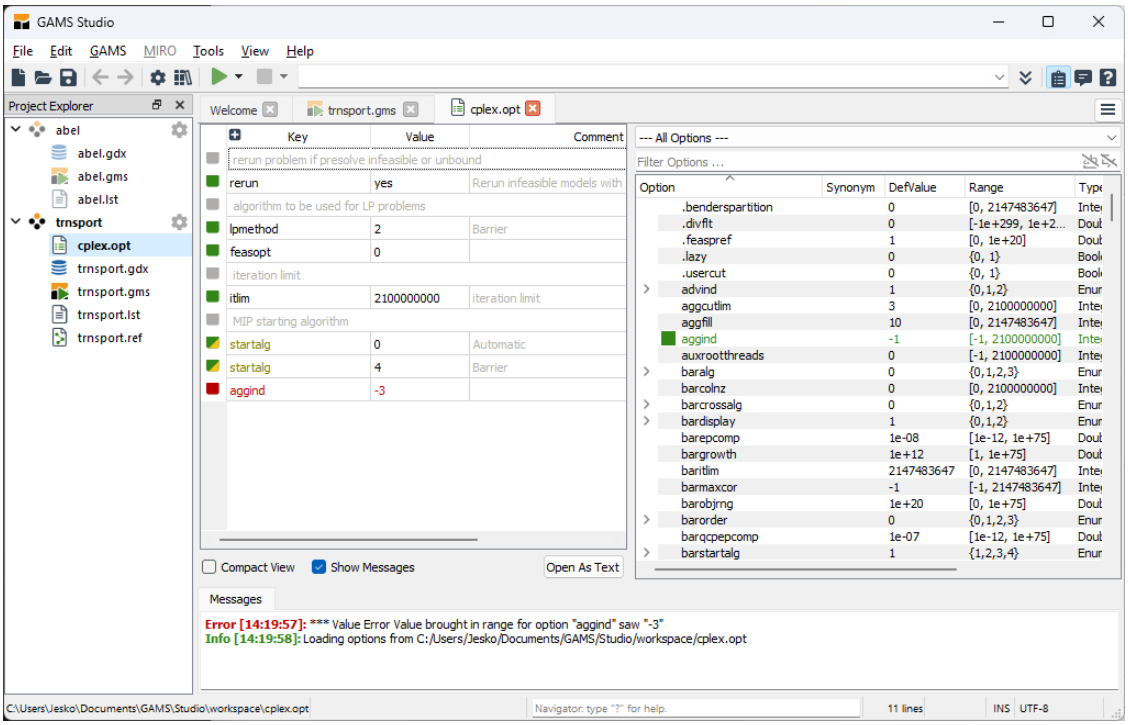
Summary of Actions and Shortcuts

Action	Shortcut	macOS	Description
Symbol Filter	**Ctrl - F**	**Command - F**	Jump to the Symbol Filter input field located above the symbol table
Auto Resize Columns	**Ctrl - R**	**Command - R**	Resize all columns to content (right side only)

6.14.2.6 Solver Option Editor

The Solver Option Editor is used to view and edit a solver-specific option file for controlling solver and interpreting results. See usage of The Solver Options File in Solver Usage and how to set optFile parameter to instruct gams to read an option file in The GAMS Call and Command Line Parameters. The Solver Option Editor provides an editor to edit the file contents in a table form where a row entry corresponds to a line in the solver option file. The Solver Option Editor also provides a browser of solver option definitions which can be displayed in group or be filtered by filter-term and where the option can be directly added into or deleted from the editor.

The Solver Option Editor consists of the contents editor in the left pane, option definition browser in the right pane, as well as message and configuration tab in the bottom pane.



- The left pane shows the contents editor of solver option file in a table form. Each row entry is either an option entry containing the option key, option value, as well as end-of-line comment (shown if there is definition of end-of-line comment characters for a solver option file) or a comment entry. Below the editor shows `Compact View` checkbox and `Open As Text` button
  - An option key and value are shown in green color when defined correctly according to the definition, otherwise in red color when there is an error. An end-of-line comment (if defined) and a comment entry are shown in grey color. (see section Toggle comment/option selection on how to turn an option entry into a comment entry and vice versa)
  - A comment row entry is shown in a merged cell as differences between option key van value are not identified in a comment line of a solver option file.
  - The `Compact View` checkbox enables the editor to display the file contents without comments. Note that some editing actions are suppressed when the editor is in compact view (see section Compact View for more details).
  - The `Show Messages` checkbox allows to display or hide the messages log at the bottom pane.
  - The `Open As Text` button allows to reopen the file in the text editor once contents of the file has been saved (see Open an existing solver option file and Save a solver option file in section Basic Operations for more details).
- The right pane shows a browser of solver option definitions.
  - The lower part of the browser lists all option definitions, each contains Option `Name`, `Synonym`, `Default Value`, `Range`, `Type`, and `Description`. The list can be sorted alphabetically in ascending or descending order by clicking the Option `Name` header. A checkbox in front of an option entry indicates whether there is an option of this definition entry defined in the left pane editor. An entry with enumeration type (`EnumStr` or `EnumInt`) can be expanded to show all enumerated values of the option by clicking the bullet in front of the option entry, clicking the bullet again hides all enumerated values of the option. By double clicking the entry or dragging the entry and dropping in the left pane editor adds the option key with the default value defined by this option definition in the left pane editor (See section Summary of Actions and Shortcuts on how to add or insert an option).
  - Above the list of all option definitions is a group of options and a filter box. Option groups filter the list of option definitions by group name. By default all option definitions are displayed. Filter box with placeholder text `Filter Options ...` allows to filter the option definitions. As a filter term has been typed into the filter box the results are displayed in the list of option definitions below. More about the filter can be found in the Filter Section.
- The bottom pane shows the message log which reports the operations that have been carried out such as the contents have been loaded and saved as well as errors resulting from editing the contents. By default the message log is shown unless the `Show Messages` checkbox in the left pane has been unmarked to hide the message log.

Basic Operations

- **Create a solver option file**: via either the Studio menu or the project context menu in Project Explorer.
  - From the Studio menu, choose `File` > `New...` then enter a valid solver option file name and suffix into the file dialog. choosing `Add new file` by right clicking on project name in Project Explorer brings up the same file dialog for entering a valid solver option file name and suffix (See how to set different suffix values for solver option file from optFile parameter).
  - From the project context menu, right clicking on project name in Project Explorer and choosing `Add new solver option file` brings up a list of solver names. Selecting a solver name from the list brings up a file dialog with the selected solver name as file name and the default option file suffix name `opt` (See how to set different suffix values for solver option file from optFile parameter).

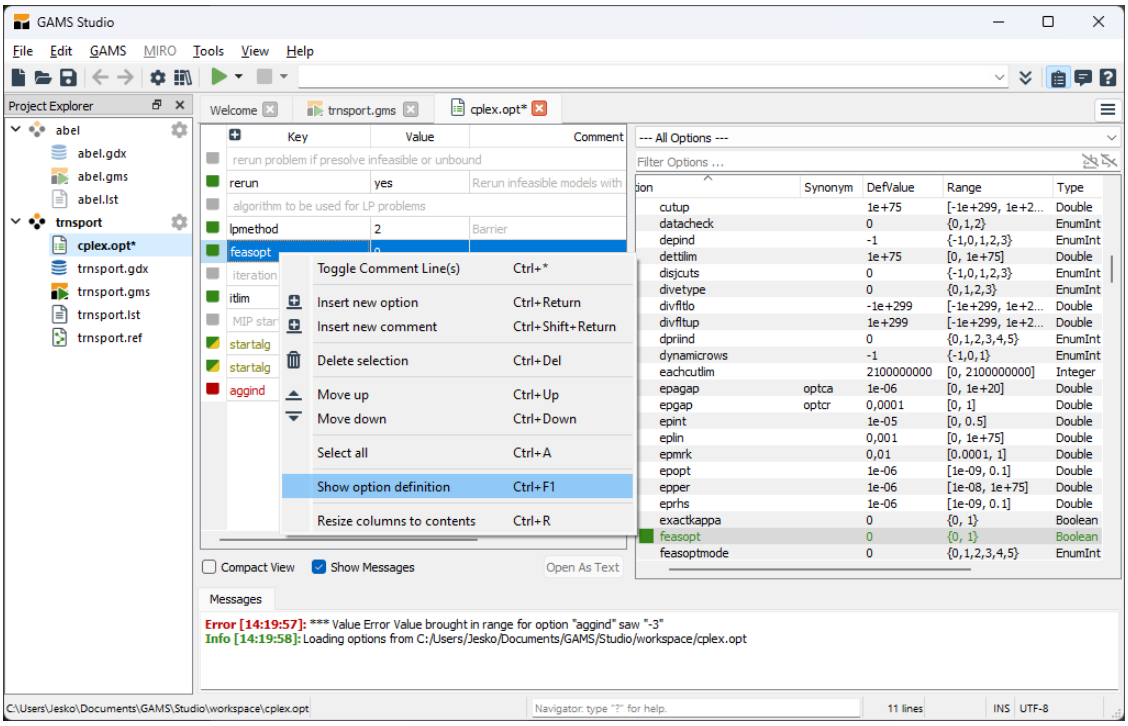
- **Open an existing solver option file**: via either the Studio menu or project context menu in Project Explorer.
  - From the Studio menu, choose `File` > `Open...` and select `Option Files` or `All Files` in the file dialog and then select a solver option file.
  - From the project context menu, right click on project name then choose `Add Existing file` then choose `Option Files` or `All Files` in the open file dialog and then select a solver option file.
  - In case a solver option file is already opened with solver option editor, the file can be reopened in text editor by choosing `Reopen File As Text` from the file context menu in Project Explorer.
  - It is possible to open the solver option file in either solver option editor or text editor.
    * In case a solver option file has already been opened in solver option editor, choosing `Reopen File As Text` from the file context menu in Project Explorer closes the solver option editor and reopens the file in the text editor. Note that `Open As Text` button in the left pane editor performs the same function.
    * In case a solver option file is already opened in text editor, choosing `Reopen File using Solver Option Editor` from the file context menu in Project Explorer closes the text editor and reopens the file in the solver option editor.
    * In case a file is already listed under the project entry but not yet opened, choosing `Open File` from the project context menu opens the file in the solver option editor; however, choosing `Open File As Text` from the project context menu opens the file in the text editor.
- **Save a solver option file**: activated by choosing either `File` > `Save` menu or `Ctrl - S` shortcut. Choosing `File` > `Save As` menu or `Ctrl - S` shortcut brings up a file dialog to choose a file with different name to be saved as (See also how to set different suffix values for solver option file from optFile parameter).

Navigating the contents

When a component of the solver option editor in focus, it is possible to perform further actions (see section Summary of Actions and Shortcuts for the list of actions and shortcuts). Clicking the component area will bring the component into focus or in selection. In addition to using mouse to navigate the solver option editor components, it is also possible using keyboard. Press `Tab` key navigates the components of the solver option editor. For example, when left pane editor is in focus, pressing `Tab` navigates from the left pane editor, to the right pane option group box, to the right pane Filter box, and to the right pane definition browser in the describing order. When the left pane editor is in focus, press `Up`, `Down`, `Left`, and `Right` keys to navigate the left pane editor table. When the right pane browser is in focus, press `Up` and `Down` to navigate the definition entries, press `Right` to expand/show the enumerated entries of the definition, and press `Left` to collapse/hide the enumerated entries of the definition.

There is a connection between an option entry in the left pane editor and a definition entry in the right pane browser. When navigating the solver option editor, it is possible to identify this connection from both the left pane editor and the right pane browser.

- From the left pane editor, right click on the selected entry and choose `show option definition` from the context menu. The entry in the right pane browser that contains the definition of the selected entry is highlighted and selected. When a cell or a row is already selected the shortcut `Ctrl - F1` delivers the same behavior.



- From the right pane browser, click or select on an entry. All entries in the left pane editor will be highlighted and selected if the definition has already been added. Otherwise there will be no selection in the left pane editor.

More editing actions can be performed on both the left pane editor and the right pane browser (See Editing the Contents for details).

Editing the Contents

The followings describe actions that can be performed when editing the contents:

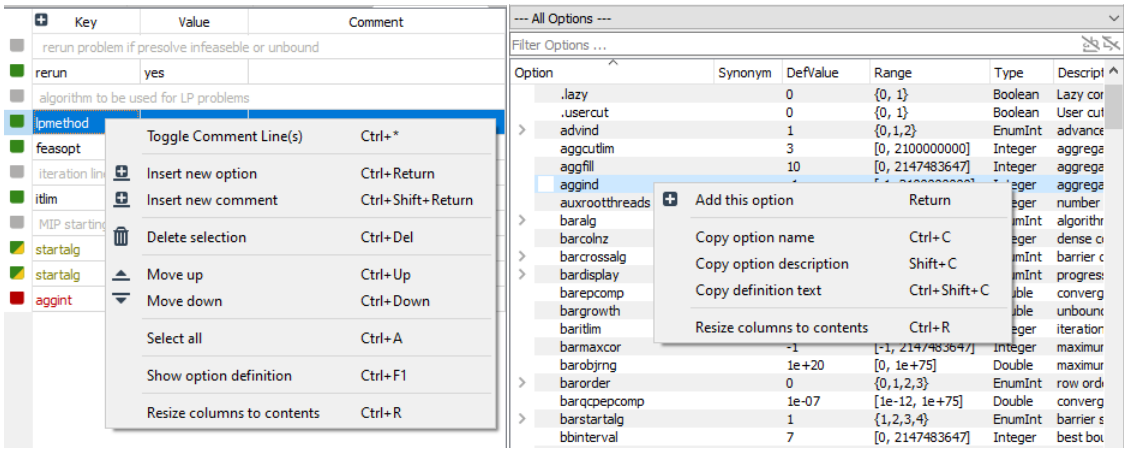


Figure 6.20 actions to be performed via the left pane editor and the right pane editor

- **Edit option key, value, and comment**: This action performs in the left pane editor.

– when the cell in the left pane editor is selected, double click on the cell or press platform-dependent edit key (eg. `F2`) in order to edit option key, value, or comment (if available). A drop-down list suggests possible option keys and option values as when possible. Press `Enter` to confirm the edit and press `Esc` to cancel the edit.

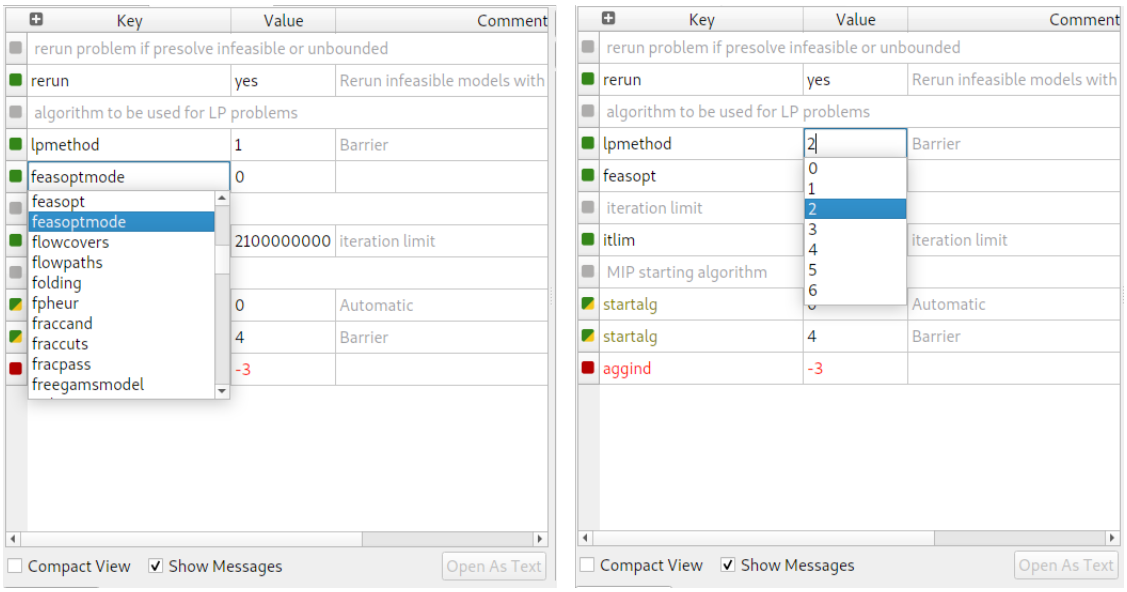


Figure 6.21 Drop-down lists suggest possible keys (left) and values (right)

- **Add new option**: This action appends a new option entry as the last entry of the left pane editor. There are several ways to add a new option entry.
  - click on the plus icon in the header of the left pane editor, a new option entry with dummy option key `[KEY]`, option value `[VALUE]`, and option comment `[comment]` (if available) is appended as the last entry.
  - it is also possible to add new option from the the right pane browser. Select the definition entry in the right pane browser that has not been added/inserted into the left pane editor (a solid checkbox in front of the entry is not marked), double click or right click on the selection then choose `add this option`. A new option entry ith the option key and default value of this definition will be added as the last entry in the left pane editor. and dummy option value `[VALUE]` is appended as the last entry.
- **Insert new option and comment**: It is possible to perform this action from the left pane editor and from the the right pane browser and there are several ways to insert a new option or a comment entry.
  - right click on the selected entry in the left pane editor and choose `insert new option` from the context menu. A new option entry with dummy option key `[KEY]`, option value `[VALUE]`, and option comment `[comment]` (if available) will be inserted before the selected entry. When a cell or a row is already selected the shortcut `Ctrl - Return` delivers the same behavior.
  - right click on the selected entry in left pane editor and choose `insert new comment` from the context menu. A new option entry with dummy text `[COMMENT]` will be inserted before the selected entry. When a cell or a row is already selected the shortcut `Ctrl - Shift - Return` delivers the same behavior.
  - double click on a definition entry in right pane browser.
    - In case this definition has not yet been added/inserted (a checkbox in front of the entry is not marked), a new option entry with the option key, default value, and comment (if available) of this definition will be added as the last entry in the left pane editor. When there is a selection on a cell or a row the shortcut `Return` delivers the same behavior.

    * In case this definition has already been added/inserted (a checkbox in front of the entry is marked), by default the studio option editor will ask for overriding existing option if there is an option from the same definition that has already been added/inserted into the left pane editor. A pop-up message box appears and offers three alternatives: either to `replace existing entry`, or to `add new entry`, or to `abort`. `Replace existing` entry will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition. `Add new entry` will add a new option entry with the option key and default value of this definition into the left pane editor. `Abort` will cancel the action. See section override existing option on how to suppress this default behavior.

  When double clicking an enumerated value entry of the definition a new option entry will be added with the option key of the parent entry and the selected enumerated value.

- drag a definition entry from the right pane browser and drop in the left pane editor.
    * In case this definition has not yet been added/inserted (a checkbox in front of the entry is not marked), a new option entry with the option key, default value, and comment (if available) of this definition will be dropped before the position of selected entry in the left pane editor.
    * In case this definition has already been added/inserted (a checkbox in front of the entry is marked), by default the studio option editor will ask for overriding existing option if there is an option from the same definition that has already been added/inserted into the left pane editor. A pop-up message box appears and offers three alternatives: either to `replace existing entry`, or to `add new entry`, or to `abort`. `Replace existing` entry will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition. `Add new entry` will insert a new option entry with the option key and value of this definition before the position of selected entry in the left pane editor. `Abort` will cancel the action. See section override existing option on how to suppress this default behavior.

  See also section Settings on how to insert a new option together with a comment from the definition.

- **Toggle comment/option**: This action performs in the left pane editor, turning an option entry into a comment entry and turning a comment entry into an option entry.
    - select the option entry and right click on the selection, then choose `toggle comment/option selection` from the context menu. In case of an option entry this action turns the entry into a comment entry (shown in grey color). In case of a comment entry this action turns the entry into an option entry entry (shown in either green or red color depending on whether or not there is an error). When a cell or a row is already selected the shortcut `Ctrl -` * delivers the same behavior. Clicking on the color box in front of the entry row also delivers the same behavior, even without a selection.

- **Move up and down**: These two actions perform in the left pane editor, changing the order of the option and comment entries.
    - to move an entry up in the left pane editor: select the option entry and right click on the selection, then choose `move up` from the context menu. The selected option entry will be moved one position up the option entry table. When a cell or a row is already selected the shortcut `Ctrl - Up` delivers the same behavior.
    - to move an entry up in the left pane editor: select the option entry and right click on the selection, then choose `move down` from the context menu. The selected option entry will be moved one position down the option entry table. When a cell or a row is already selected the shortcut `Ctrl - Down` delivers the same behavior.

- **Deleted option**: It is possible to perform this action from the left pane editor and from the the right pane browser.
    - from the left pane editor, select the option entry and right click on the selected, then choose `delete selection`. The selected option entry will be deleted from the option entry table. When there is a selection on a cell or a row the shortcut `Ctrl - Delete` delivers the same behavior.

– from the right pane browser, select the definition entry that has already been add/inserted into the left pane editor (a checkbox in front of the entry is marked) and right click on the selection then choose `remove this option`. All entries in the left pane editor defined by this definition will be deleted from the option entry table. When there is already a selection on a definition row the shortcut `Delete` delivers the same behavior.

- **Show option definition**: This action performs in the left pane editor. Right click on the left pane of the option editor and choose `Show option definition`. The option definition entry on the right pane will be highlighted. The shortcut `Ctrl - F1` delivers the same behavior.

- **Show all options of the same definition**: This action performs in the left pane editor. Right click on the left pane of the extended editor and choose `Show all options of the same definition`. All option entries on the left pane will be highlighted. The shortcut `Shift - F1` delivers the same behavior.

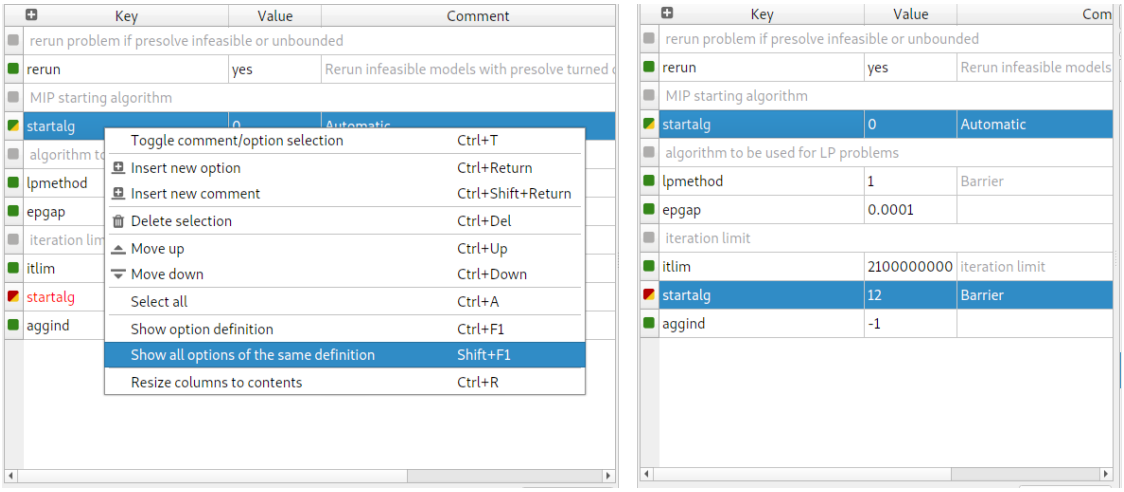


Figure 6.22 All entries of the same option are highlighted

Compact View

Compact View of the solver option editor allows to view and edit solver option without comments. Comments in a solver option file are not interpreted by either GAMS or the solver but used for documentation purpose. As the contents of the solver option file grows larger with several comment lines, it can be difficult to see which options eventually will be interpreted. To this end, the compact view becomes useful to hide all comments and show only non-comment lines. In the right pane editor, mark `Compact View` checkbox to activate the compact view and unmark the checkbox to deactivate the compact view. The comment entry is shown again once the compact view is deactivated.

Note that the result of some actions are not visible when in compact view. For example, action **toggle comment/option selection** when performed on an option row entry, turning the entry into a comment entry and therefore hidden from the compact view. Action **delete selection** when performed on an option row entry, removing the content entry and therefore no longer visible hidden from the compact view.

Some editing actions that can change the order of the contents are suppressed. These actions are **insert new option**, **insert new comment**, **move up**, and **move down**. Nevertheless adding or inserting option from the right pane are allowed, but a comment that has been inserted with the option (if set) are not visible in compact view. See also section Settings on how to insert a new option together with a comment from the definition.

Settings

The setting tab allows to configure the behavior when inserting new option from definition and deleting option. The setting can be accessed by opening the Settings dialog (`File` > `Settings`) and switching to the `Misc.` tab. These configurations are:

- **Override existing option**. This behavior allows Studio to override existing option when an option entry has been added or inserted from the right pane browser. In case there are more than one entry of the same option, Studio will pop-up a message box to offer three alternatives: either to `replace existing entry`, or to `add new entry`, or to `abort`.
  - *Replace existing* entry will remove all other entries of this definition key but the first entry from the left pane editor and replace the option value of the entry by the default value defined by the definition.
  - *Add new entry* will add or insert a new option entry with the option key and default value of this definition into the left pane editor.
  - *Abort* will cancel the action.

  By default this behavior has been set. In the Settings dialog, unmark the checkbox in front of the text *overriding existing option* to suppress the behavior and mark to enable the behavior.
- **Add option description as comment above**. This behavior allows Studio to add option description as additional comment entry above an option entry that has been added or inserted from the right pane browser. The description has been taken from the option definition in the right pane browser. By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text *add option description as comment above* to enable the behavior and unmark the checkbox to suppress the behavior.
- **add option description as end of line comment** This behavior allows Studio to add option description as additional end-of-line comment of an option entry that has been added or inserted from the right pane browser. The end of line comment is only available when the solver defines valid end of line characters. The end-of-line comment column will be shown in the solver option editor only when available (there is definition of end-of-line comment characters for a solver option file). By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text *add option description as end of line comment* to enable this behavior and unmark the checkbox to suppress the behavior.
- **delete all immediate comments above** This behavior allows Studio to delete all immediate comments (if there is any) above when deleting an option. By default this behavior has not been set. In the Settings dialog, mark the checkbox in front of the text *delete all immediate comments above* to enable this behavior and unmark the checkbox to suppress the behavior.

Summary of Actions and Shortcuts

Actions and their shortcuts that can be performed via the left pane of solver option editor are:

Action	Shortcut	macOS	Description
Toggle option/comment selection	**Ctrl** - ∗	**Command** - ∗	Toggle between option and comment
Insert new option	**Ctrl** - **Return**	**Command** - **Return**	Insert a new option
Insert new comment	**Ctrl** - **Shift** - **Return**	**Ctrl** - **Shift** - **Return**	Insert a new comment
Delete selection	**Ctrl** - **Delete**		Delete the selected option/comment
Move up	**Ctrl** - **Up**	**Command** - **Up**	Move the selected option/comment up for 1 row

Action	Shortcut	macOS	Description
Move down	**Ctrl - Down**	**Command - Down**	Move the selected option/comment down for 1 row
Select all	**Ctrl - A**	**Command - A**	Select all options
Show option definition	**Ctrl - F1**	**Command - F1**	Show definition of this selected option in the right pane
Show all options of the same definition	**Shift - F1**		Show all options of the same definition defined in the right pane
Resize columns to contents	**Ctrl - R**	**Command - R**	Resize the columns in the left pane to contents

Actions and their shortcuts that can be performed via the right pane of solver option editor are:

Action	Shortcut	macOS	Description
Option Filter	**Ctrl - F**	**Command - F**	Jump to the focus to the Option Filter input field
Add this option	**Return**	**Return**	Add option in the left pane from the selected definition
Remove this option	**Delete**		Remove option defined by this definition from the left pane
Copy option name	**Ctrl - C**	**Command - C**	Copy option key from this selected definition
Copy option description	**Shift - C**	**Shift - C**	Copy option description from this selected definition
Copy definition text	**Ctrl - Shift - C**	**Command - Shift - C**	Copy option text from this selected definition
Resize columns to contents	**Ctrl - R**	**Command - R**	Resize the columns in the right pane to contents

6.14.2.7 GAMS Configuration Editor

The GAMS Configuration Editor is used to view and edit a GAMS Configuration File in YAML Format. The editor shows the two configuration sections: command line parameters (`commandLineParameters`), and operating system environment variables (`environmentVariables`), in different tabs of the editor.

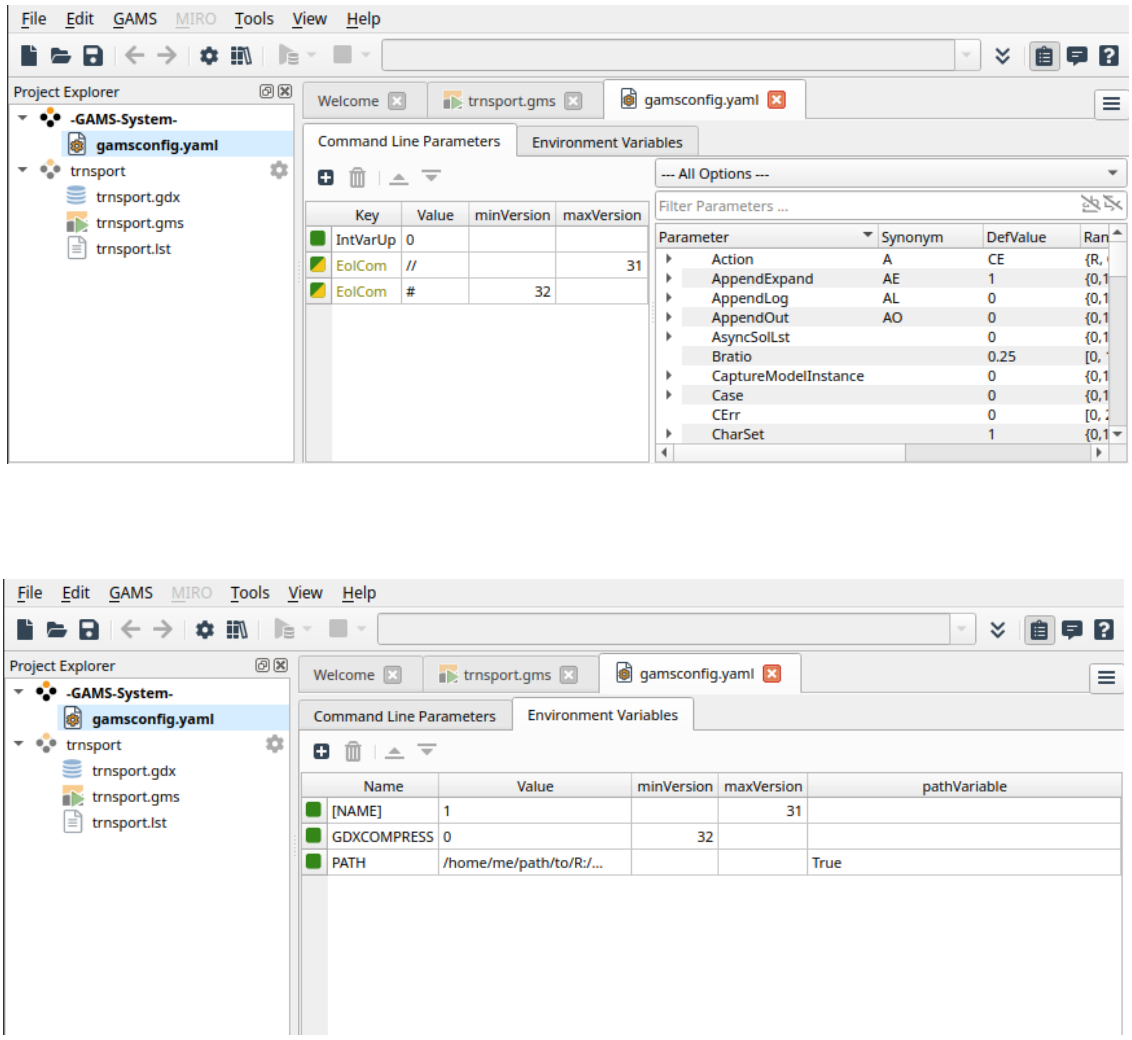


Figure 6.23 Contents of gamsconfig.yaml shown in GAMS Configuration Editor

Note that GAMS Configuration editor does not show the contents of the third section: external solver configuration (`solverConfig`) if exists. However, the contents in this section will be written out to the file when saving the file via the editor. It is possible to use `Reopen File as Text` from Project Explorer context menu to reopen and edit the file contents in text editor when the file is already opened in the GAMS Configuration Editor and use `Open File as Text` from Project Explorer context menu to open the file in text editor when the file is not yet opened in any editor. See the syntax of `gamsyconfig.yaml` file from GAMS Configuration File in YAML Format.

GAMS will process a sequence of `gamsconfig.yaml` files from different locations (see the location of GAMS Configruation file from The GAMS Configuration in YAML Format and the order of precedence for command line parameters from Order of Precedence for Options). Studio can create a new `gamsconfig.yaml` file at the default user-specific location and open the file in the editor when the file at the location does not exist. This can be done by choosing `GAMS > Default GAMS Configuration`. In case the file already exists at the default location, choosing `GAMS > Default GAMS Configuration` will not override the `gamsconfig.yaml` in the default user-specific location but open the file in the editor.

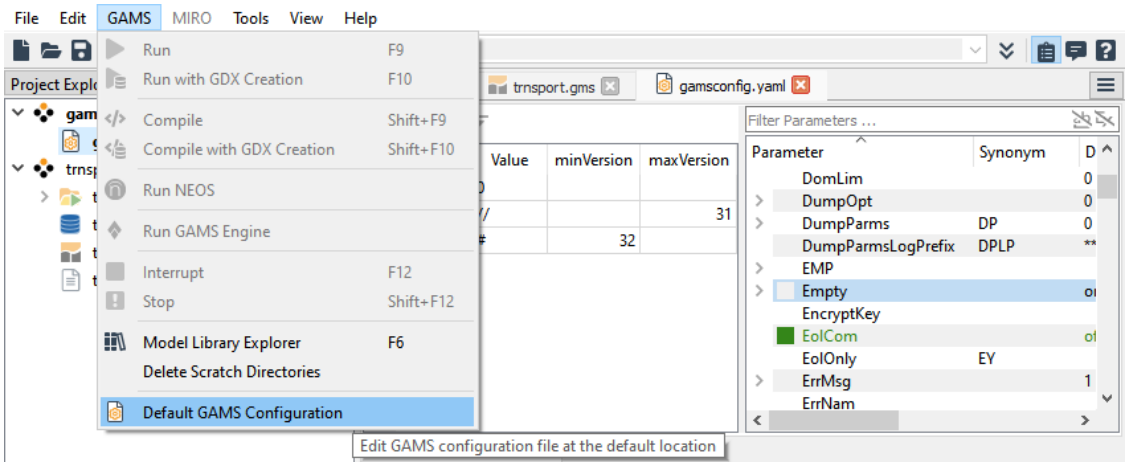


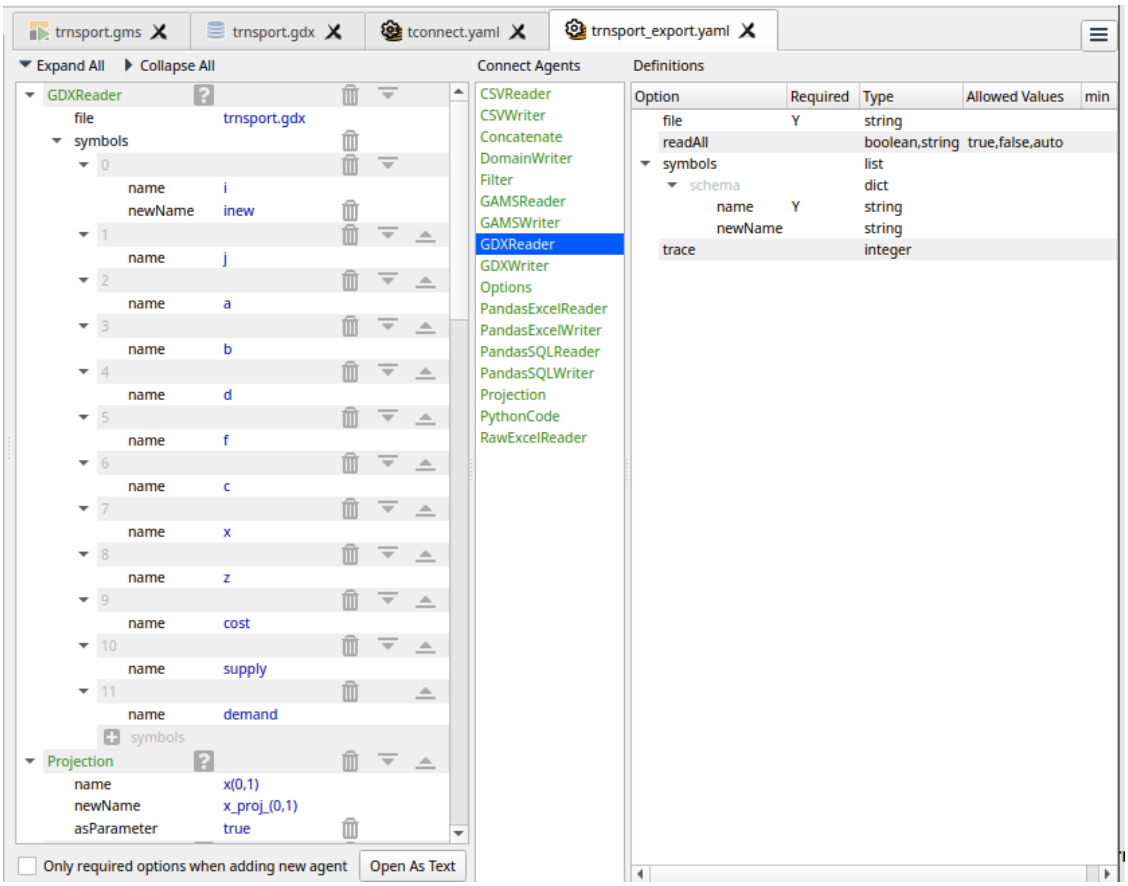
Figure 6.24 Menu entry point to create/open gamsconfig.yaml file from default location

It is possible to save the `gamsconfig.yaml` file into different location using `File` > `Save As...`.

6.14.2.8 Connect Editor

Connect Editor provides fundamental functionalities for creating and editing GAMS Connect file, containing instructions in YAML syntax which will be processed by the GAMS Connect interfaces. The Connect Editor consists of three main sections:

- the left section displays instructions from the file in tree structure. The top level entries are Connect Agents names with option entries as its children.
- the middle section lists all available Connect Agents. Double click or drag-and-drop an agent name from this section into the left section will add or insert instructions for the selected agent in the left section.
- the right section shows Definition of the Connect Agents selected in the middle section. Clicking an agent name in the middle section will display definition of the selected agent in this section.



A new connect file can be created by selecting **Add new file** from the **File** menu or the project context menu in Project Explorer. An existing connect file can be edited by selecting **Open**... from the **File** menu or **Open in Current Project...** or **Add existing file** the project context menu in Project Explorer. In both cases, a file dialog opens to select the project folder, choose **Gams Connect Yaml File (∗.yaml)** as a type of files.

A top level entry of instructions is a connect agent name with option entries as its children. Each `option entry` is represented as either a map of `option key` to `option value` or a list of either `option values` or option `entries` depending on the definition of the agent. Only an `option value` can be edited. An edited `option value` that does not conform to the definition (eg. wrong type or unallowed value) will be highlighted in red and so is its agent name. By clicking `Help` icon next to the agent name at top level entry, the agent name in the middle section will be selected and the definition of the agent will be shown in the right section.

New instruction for an agent can be added by either double clicking an agent name in the middle section or dragging an agent name in the middle section and dropping it in the left section at the desired position. Double clicking an agent name will append instruction for the selected agent as the last agent entry in the left section, while dragging an agent name will either append instruction in the left section as the last agent entry or insert instruction for the selected agent below the drop position. When **Only required option when adding new agent** checkbox is marked, only the required options of the agent will be added. An option missing from the agent name entry can be inserted or appended by dragging the option from the definition in the right section and dropping it at the desired position in the left section. Dropping the option onto the agent name entry will append the option as the last child entry of the agent name. The missing option can still be added even when it is not required by definition and **Only required option when adding new agent** checkbox is marked.

A new child entry of a list entry can be added using the `plus` icon at the last child entry of the list. An option entry and all its children can be `removed`, `moved up`, and `moved down` using `delete` icon, `move up` icon, and `move down` icon accordingly at the end of each entry. An option entry with children can

be either `expanded` or `collapsed` by clicking an icon in front of the entry. The tree structure can be completely expanded by clicking **Expand All** and collapsed by clicking **Collapse All**.

The content of connect file can also be displayed and edited in plain yaml format using a text editor. Switching from connect editor to text editor can be done by either clicking **Open As Text** button or selecting **Reopen File as Text** from the context menu of the file in the Project Explorer.

- **Open As Text** button will be enabled once the content of the file has been saved, allowing a switch to the text editor.
- Selecting **Reopen File as Text** from the context menu, when the content of the file has been modified, will popup a dialog to report modified content and offer options to `discard the content` or to `save the content` or to `cancel the operation`.
  - In case of discarding the content, Studio will switch to the text editor to display the current file content before modifying.
  - In case of saving the content, Studio will save the content into a file before switching to the text editor.
  - Cancel operation will bring back to the current modifying state of connect editor.

6.14.2.9 Parameter File

To support fast switching between parameter sets, Studio contains an editor for "GAMS Parameter Files". To create a Parameter File, open the context menu of a project and select "Add New Gams Parameter File". The project settings contains a box to switch between different parameter files or to deselect it for the project.

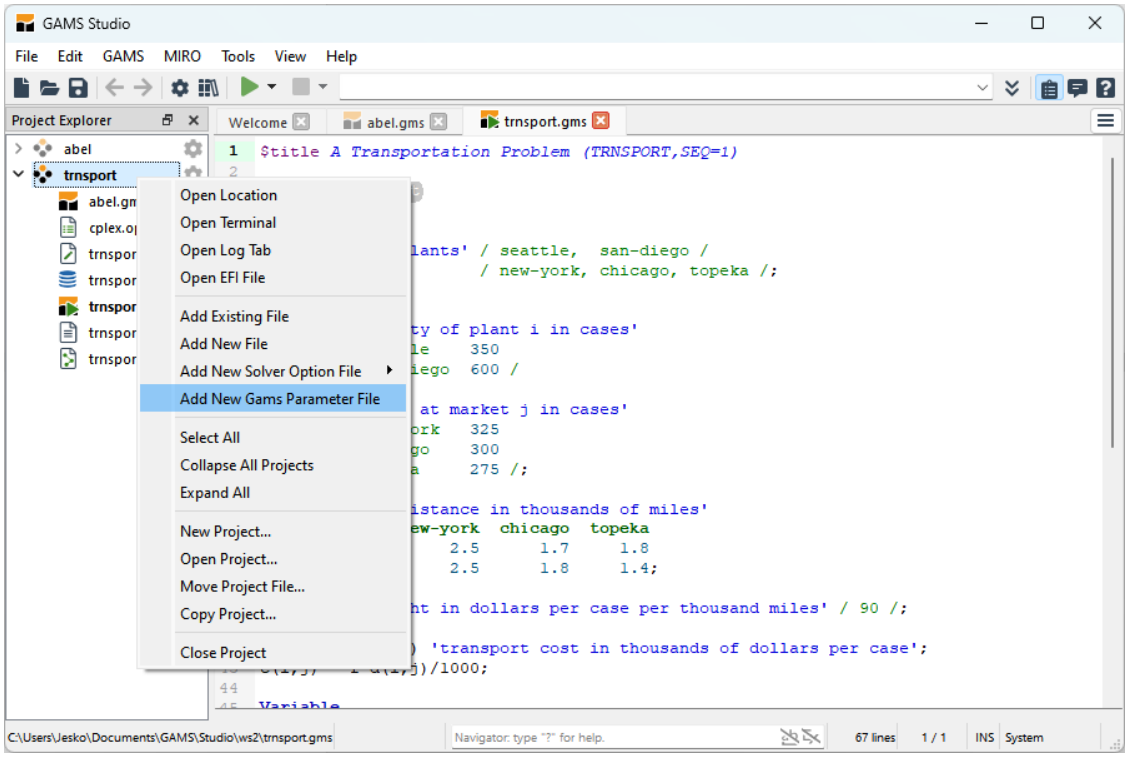


Figure 6.25 Pin View

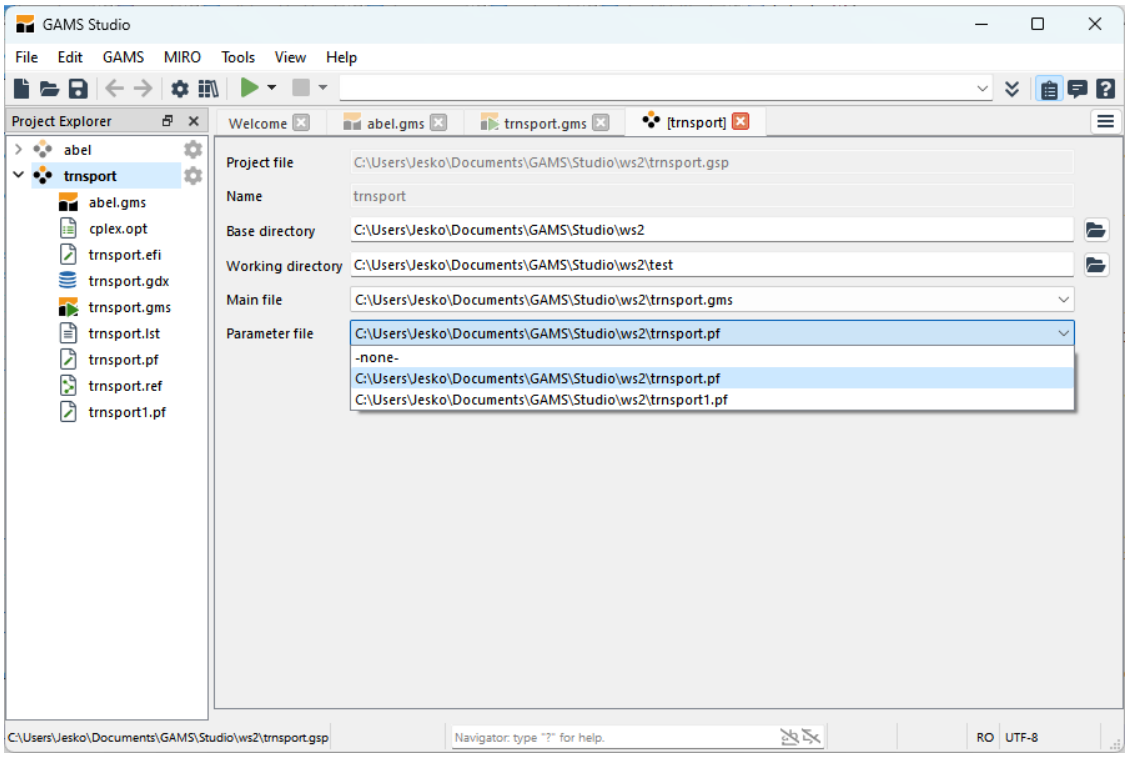


Figure 6.26 Pin View

Parameter defined in the parameter file will overlay parameters defined in the GAMS Configuration File. The usage of the editor is equivalent to the Solver Option Editor.

6.14.2.10 Tab Browser (Deprecated)

Deprecation Notice: This feature will be removed in a future version of GAMS as it is replaced by the Navigator.
The Tab Browser is accessible with a button next to the main tab bar or via the shortcut `Ctrl - 4`. It opens a list of all open files and features a search input field. When opening the tab browser, the search field is already focused. Typing updates the list in real time. The `Enter` key selects the first item in the list and opens it. It is also possible to use arrow up/down to navigate the list and select an item with `Enter`. The filter supports wildcard syntax.

6.14.2.11 Navigator

The Navigator allows users to quickly navigate between files and can be opened with the shortcut `Ctrl - K` or by clicking into the Navigator input field located in the Status Bar in the bottom right. The default view shows a list of all files from the Project Explorer with three columns. The first column contains the file name, the second the path relative to the current project's directory. If it is the same path the field is empty. And the third column contains additional information, by default which kind of result it is. Typing something into the input field filters this list. Using the arrow keys a result can be selected and confirmed with the `Enter` key. The filter also supports wildcards: to e.g. find all ".gdx" files one can input `*.gdx`. To narrow a search down even more the Navigator supports different modes which are accessed with prefixes in front of the search term. To see a list of all available modes just type "?" into the input field. This allows, for example, to only show files that are currently opened in a tab (`t`), associated with the current project (`p`), or all open log tabs (`l`). Additionally using the `:`prefix followed by a number users can jump to a certain line in the current file. There is the possibility to jump to files that are not yet known

to Studio: using the `f` prefix allows users to navigate the filesystem of the operating system, starting with the current project's workspace. Also, with the quick action prefix (`x`) it is possible to call often used Studio functions like "Open Model Library Explorer", "Clean Scratch Directories", "Run Engine", and many more. For a complete list just input `x` into the Navigator.

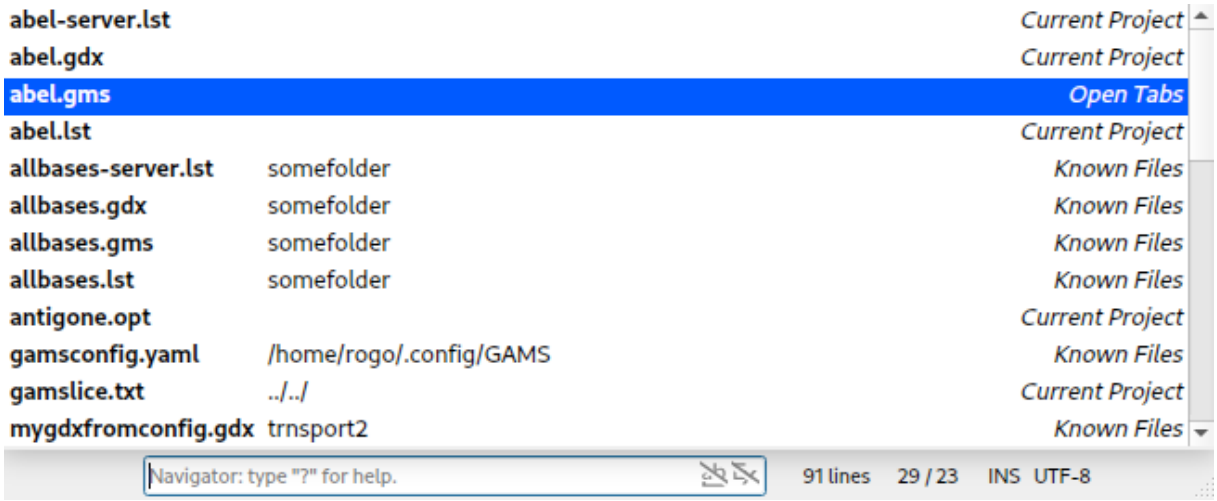


Figure 6.27 Navigator showing some example files

6.14.2.12 Pin View

Studio provides a Pin View to carry a clone of a selected tab. This allows to have two file contents visible at the same time. To pin a tab there are two entries in the tabbar: "Pin right" (shortcut `Ctrl - Click`) splits the edit area horizontally and opens the Pin View right to the main tabs. "Pin below" (shortcut `Ctrl - Shift - Click`) splits the edit area vertically and opens the Pin View below the main tabs. The state and size of the Pin View is restored on restarting Studio.

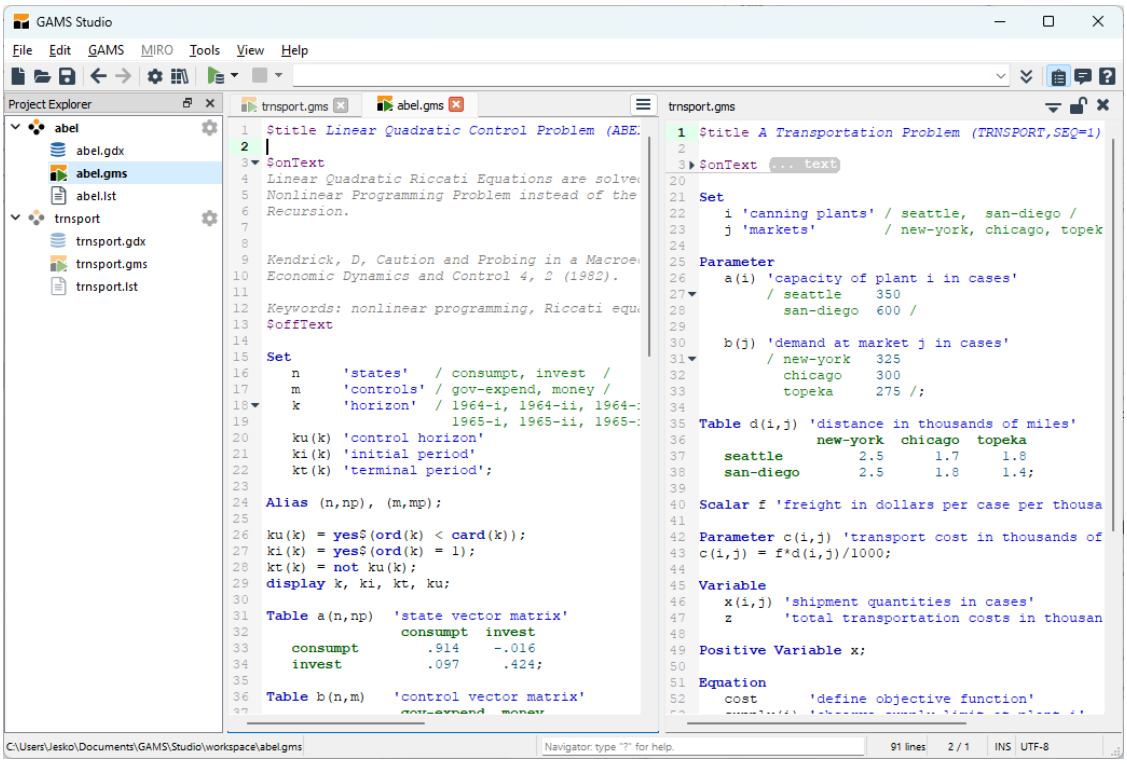


Figure 6.28 Pin View

The Pin View provides additional buttons. The first button switches the split orientation from right to below and vice versa. The second button allows to activate synchronized scrolling for the two visible editors. Synchronizing is always passed from the active editor containing the focus to the passive editor, which means that scrolling in the passive editor with the mouse wheel isn't passed back to the active editor allowing adjustments without switching the function off and on. The last button closes the Pin View.

6.14.2.13 Navigation History

The Navigation History records a navigation history event each time the cursor is moved in some kind of text file opened in the Studio central widget. This allows the user to quickly jump back (and forward again) to text passages that were recently edited. To reduce clutter some events are set to overwrite the previous event. This happens when the cursor is moved by only a single position in either direction, so that when a user jumps through text or types something only the latest position will be put into the history. The idea behind this is to make it easier and more convenient to jump to positions that actually matter. Using the navigation history is not limited to a single file. It is easy to jump between two or more files to go back and review recent changes. If a file was closed in the meantime it will be reopened when navigating through the history. This includes even files that were completely removed from Studio. When such a file is reopened through the Navigation History feature, due to technical limitations, the affiliation of the file to its previous project is lost. Please note that after navigating back in the history all "future" history events will be deleted and overwritten if the user creates new history events.

Action	Shortcut	macOS	Alternative Shortcut
Go Back	**Alt - Left Arrow**	**Ctrl - [**	**Mouse Button 4**
Go Forward	**Alt - Right Arrow**	**Ctrl -]**	**Mouse Button 5**

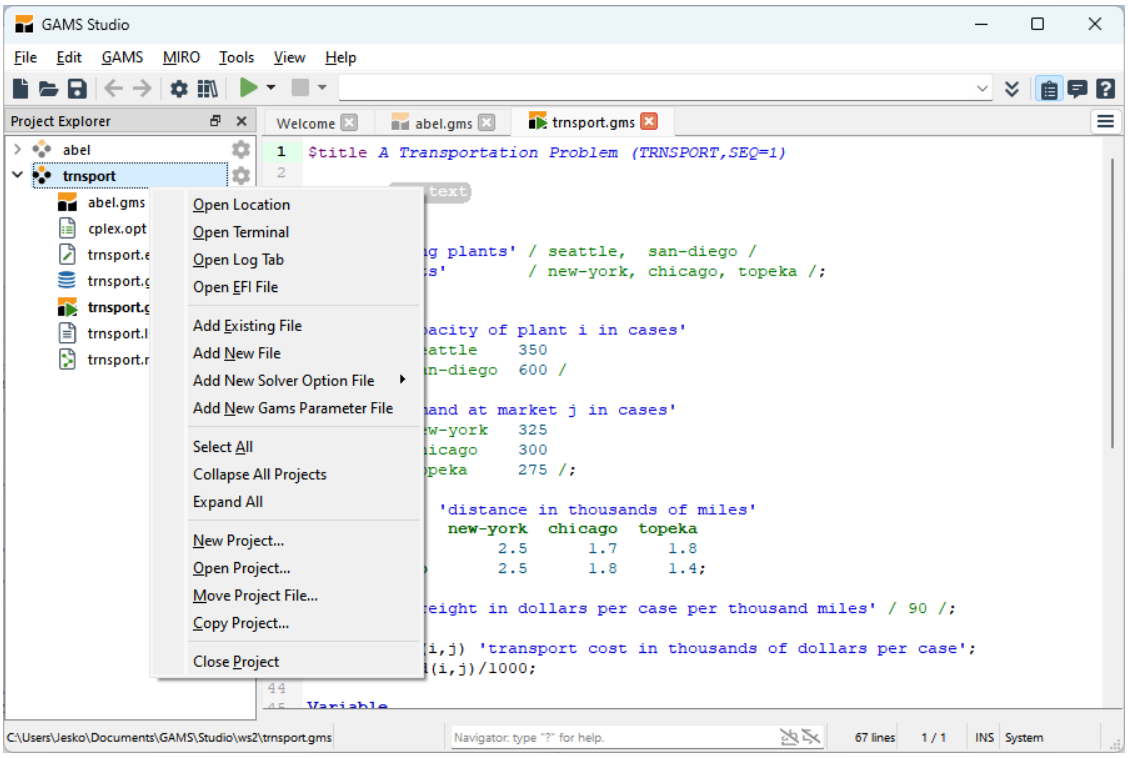
6.14.3 Further Studio Widgets

6.14.3.1 Project Explorer

The Project Explorer organizes the files opened in Studio into projects. Each project has its own base directory. All files associated to the project are organized in a hierarchical tree structure of folders relative to this base directory. A projects tooltip tells the location of the project in the file system, so it shows the location of the project file (the `.gsp` file - `gsp` standing for GamsStudioProject) . To support faster detection there are several icons for different file types:

- The runnable GAMS files (currently supported `.gms` and `.inc`) are listed with GAMS icon,
- the main GAMS file assigned to the project additionally contains a green play symbol,
- GDX files (.gdx) have an database icon,
- Reference files (.ref) are shown with a reference nodes on a sheet,
- editable files (e.g. `.txt`) contain a pen on a sheet,
- read-only files (e.g. `.log`) contain lines on a sheet.

The tooltip of a file or folder entry tells the location in the file system. The file location, the project location, and the folder location can be explored using the default file system explorer via the context menu `"Open Location"`. Selecting `"Open Terminal"` opens the systems terminal in the location of the selected file, project, or folder. The context menu of a `.gdx` file provides a special entry `"Open in GDX Diff"` that opens the specific GDX Diff Dialog.



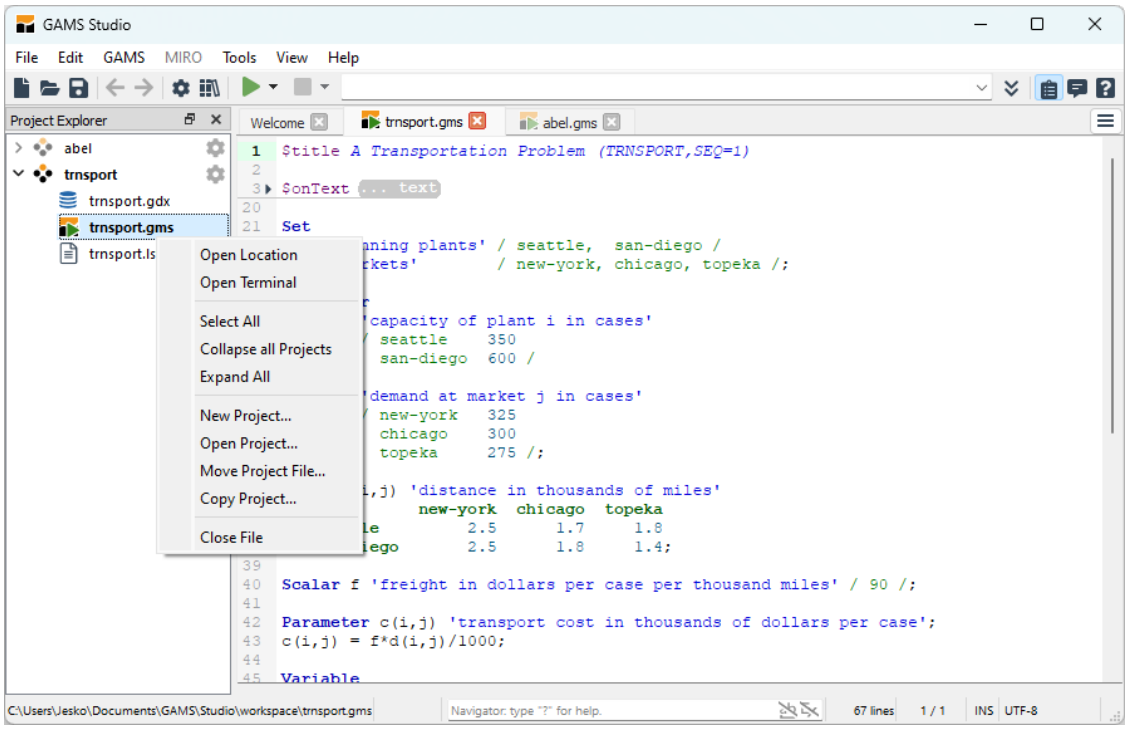


Figure 6.29 Project Explorer and its context menu

An empty project can be created by selecting `"New Project..."` from the `"File"` menu or the context menu in the Project Explorer. A file dialog opens to select the project file name. If a project with that name already exists, `Studio` looks for a number to append until the projects name is unique. Additionally a project is created automatically when opening a file. The projects name and location is determined by the name and location of the opened file. It is possible to add two projects of the same name to Studio. To help keeping them apart, a gray number is added to the project name of conflicting names.
One project may contain multiple runnable GAMS files but only one runnable GAMS file is set as the main file, marked with a green arrow overlay. By default the first runnable GAMS file in a project is considered the main file. The tooltip of a project also tells the location of the folder and the name of the current main file. Additionally the name of the recently created output file (.lst) is shown after the first run. The main file can be changed to a different runnable GAMS file in a project via the context menu `"Set as Main File"`. The project base directory remains unchanged if the runnable GAMS file is changed.

The **main file** of the active project will be executed when pressing `F9` or clicking the execute button (see how to set GAMS parameters and execute the main file in Quick Access Toolbar and GAMS Parameter Editor). To better recognize the active project, the current file and its project are in bold text and the icons of other projects are slightly dimmed. After executing the main file, the generated listing file (.lst) is added to its project and the output view (see Process Log) is opened. A clickable log line in the process log may add the corresponding file into the project and open the file in the Central Widgets area. Users can manually add more files to a project by right-clicking and choosing either `"Add Existing File"` or `"Add New File"` from the context menu. Additionally Drag and Drop from the systems file explorer is supported.

It is possible to select multiple items in the Project Explorer. A selection always contains items of one kind, files or projects never both depending on the first selected item. Folders in this case count as multiple file selection. The context menu then addresses all selected items.

Files can be removed from a project and projects can be removed from Project Explorer via the two actions `"Close File"` and `"Close Project"` from the context menu. The action `"Close Project"` closes the project and all files in the project if opened as well as removes all file entries and the project entry from

the Project Explorer without actually touching the files on the file system. The action "Close Files in This Folder" closes all files in this folder and it's subfolders. The action "Close File" closes the file if opened and removes the file entry from the project. In case the closed file is the only entry in a folder, the folder is also closed accordingly. In contrast a project can be empty during the current session but an empty project isn't restored on Studio restart. In case there are unsaved changes a message popup will appear, asking the user how to proceed.

Each project is stored in its unique file. Related paths are stored relatively to the location of the project file. The relative paths allows to share the project along with its other files. You can use "Move Project File" to rename a project or to move it to another folder. In this case the relative paths are adjusted to reach the original referenced files. The option "Copy Project" allows to create a copy of the project file and all of its referenced files.

Selected files can be dragged to another project. By dropping them they are moved to the destination project. If a folder is empty after this action its representation is removed from the project. Pressing the Ctrl key creates a copy of the file-reference in that project (no file copy). If the project already contains a specific file it is just ignored.

Next to each project there is a cog wheel button that opens the **Project options** page. It contains these actions:

- Project file: shows the location of the project file
- Name: shows the name of the project
- Base directory: defines the directory from where the folders to each file of the project are generated. Changing this will update the visible folder structure to all files of this project
- Working directory: defines the directory where the main GAMS file associated to the project will be executed
- Main file: informs about the current main GMS file

The changed Project Options can be applied by selecting "File > Save" or pressing Ctrl - S.

Actions like rename, move, or delete from file system are currently not supported but will be added in the future.

Note

> For long-time users of the classic GAMS IDE the section about the differences between IDE projects and Studio projects might be of particular interest.

6.14.3.2 Process Log

After starting a GAMS process the LogViewer is opened to view the process **Log**. Usually the **Log** stays at the bottom to keep tracking the latest log lines. The tracking can be paused by clicking on the up arrow of the vertical scrollbar. To continue tracking drag the scrollbar to the bottom.

The **Log** usually contains links of different kinds:

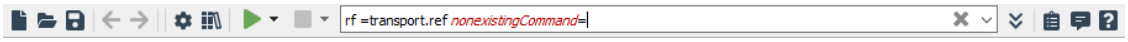
- compilation errors (red): Links to the gms file where the error occurred. Additionally contains a link to the listing file (.lst) at the beginning of the line (blue).
- listing links: Links to a specific line in the listing file
- file link (green): Links to a file without line numbers such as .gdx or .ref

Clicking on a link takes the focus to that file (and line). Double-clicking in the **Log** looks for the nearest link to the listing file and takes the focus there. Files that were not part of the current project are added when they are opened by link.

For general messages there is the **System Log** stacked together with all **Process Logs**. It keeps track of **Studio** related messages not originated from a specific GAMS run.

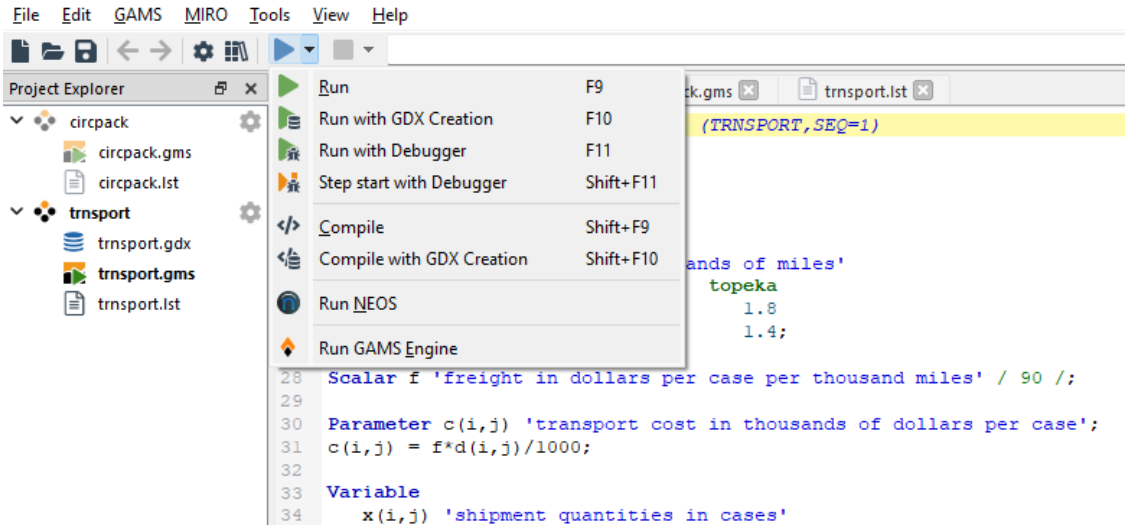
6.14.3.3 Quick Access Toolbar and GAMS Parameter Editor

The toolbar contains two parts: a quick access section and GAMS Parameter editor to customize how GAMS is executed the model (see The GAMS Call and Command Line Parameters).



The quick access section contains icons for performing common `File` actions like *New*, *Open*, and *Save* , as well as accessing *Settings* dialog, *Model Library Explorer* dialog, *Project Explorer* view, *Process* Log, and *Help* view.

The GAMS Parameter Editor provides a way to control and customize the execution of a GAMS model. The editor displays the parameters of the project of the file that is currently opened in Central Widgets (see also Project Explorer for project and its runnable files). The parameters can be typed into a combobox and the execution of a GAMS model can be carried out using the following pre-customized execution commands:



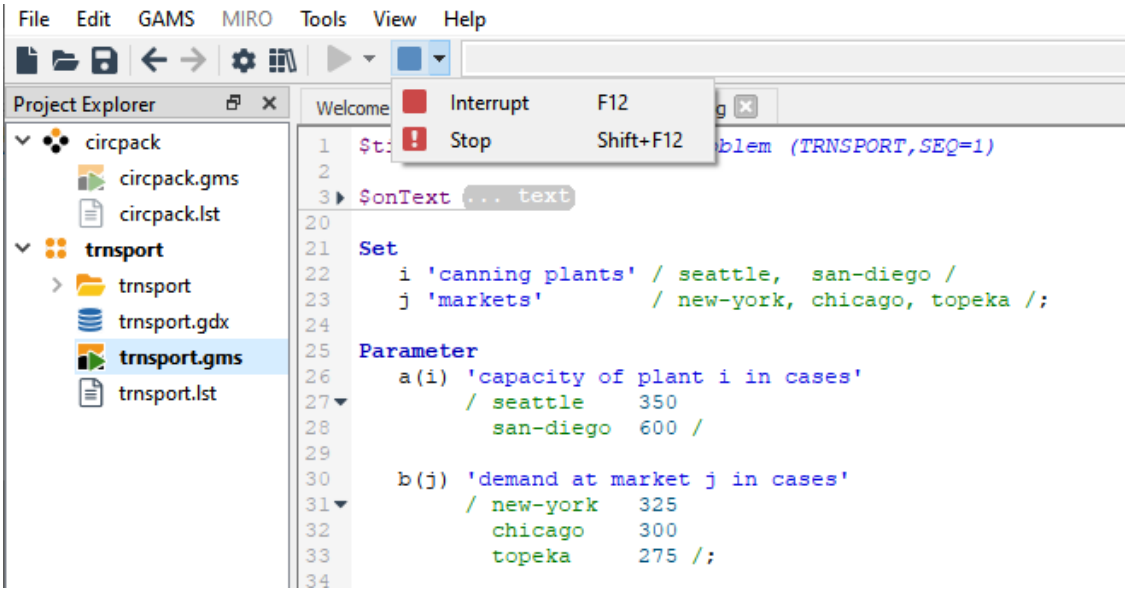
- **Run**: Choose `Run` or press `F9` to compile and execute GAMS statements in the main runnable file. This execution command is equivalent to running GAMS with the default parameter action=CE (Compile and Execution).
- **Run with GDX Creation**: Choose `Run with GDX Creation` or press `F10` to compile and execute GAMS statements in the main runnable file and create a GDX file with the name of the main runnable file and a gdx extension. This execution command is equivalent to running GAMS with the combination of the two parameters action=CE and gdx=default.
- **Run with Debugger**: Choose `Run with Debugger` or press `F11` (macOS `Meta - F11`) to compile and execute GAMS statements in the main runnable file with the debugger enabled.
- **Step start with Debugger**: Choose `Step start with Debugger` or press `Shift - F11` to compile and execute GAMS statements in the main runnable file with the debugger enabled and pause the process at the first possible line.
- **Compile**: Choose `Compile` or press `Shift - F9` to compile GAMS statements in the main runnable file. This execution command is equivalent to running GAMS with the parameter action=C (CompileOnly).

- **Compile with GDX Creation**: Choose `Compile with GDX Creation` or press `Shift - F10` to compile GAMS statements in the main runnable file and create a GDX file with the name of the main runnable file and a gdx extension. This execution command is equivalent to running GAMS with the combination of the two parameters action=C and gdx=default.
- **Run NEOS**: Choose `Run NEOS` to compile GAMS statements in the main runnable file locally and pass the execution to the NEOS server.
- **Run GAMS Engine**: Choose `Run GAMS Engine` to compile GAMS statements in the main runnable file locally and pass the execution to the GAMS Engine server.

The pre-customized execution command will operate on the main file (marked with a little green arrow over the regular icon) of the active project (see how to manage the main file in section Project Explorer). In case GAMS parameters have been set they will be appended to the pre-customized commands. Just like the GAMS Terminal, when there are identical parameters with different values the last one "wins" and overwrites the previous ones. Thus it is possible for a user to change and override GAMS Studio default parameters, which can possibly lead to problems if done incorrectly. In this case Studio prints out a warning message, informing users about potential problems. To debug what Studio does the full GAMS call parameters will be printed to the System Log. After execution, the parameters will be added into the project history. All files in a project share one parameter history. The previous GAMS parameters can be recalled from the project history via the drop-down menu of the combobox. Switching the file opened in Central Widgets will activate the different project history only when the project of the newly opened file is different from the project of the file before the switch.

The parameters `curDir` and `workDir` behave slightly differently in Studio compared to the terminal. When using one of these parameters on the command line GAMS expects the gms file to be in the given folder. In Studio the path to the input file is always given as an absolute one. Therefore, there is no need to use the `inputDir` parameter to make the gms file accessible to GAMS if it is not in the specified working directory.

The GAMS Parameter Editor provides a way to either interrupt or stop the currently running job when an execution of a GAMS model is in progress (indicating with animated icon over the project folder icon in Project Explorer) :



- **Interrupt**: Click the `Interrupt` button or press `F12` to send an interrupt request to the running job in order to perform a graceful stop and collect an incumbent result back from the execution if the solver supports this feature. The command is enabled when there is a job that is currently running in the project and disabled when there is no currently running job in the project.

- **Stop**: Click the `Stop` button or press `Shift - F12` to send a request to stop the running job immediately. The command is enabled when there is a job that is currently running in the project and disabled when there is no currently running job in the project.

The Extended GAMS Parameter Editor allows to configure the GAMS parameters. The extended parameter editor is shown when the Show button next the parameter combobox is clicked or with the shortcut `Ctrl - Alt - 3`. The editor can be hidden when the button is clicked again. When shown, the GAMS parameters from the parameter combobox will appear as a list of entries in the left pane of the extended parameter editor, each entry contains the `Key` and `Value`.

Note that the GAMS parameter combobox will be disabled when the extended parameter editor is shown and all editing has to be done in the extended parameter editor.

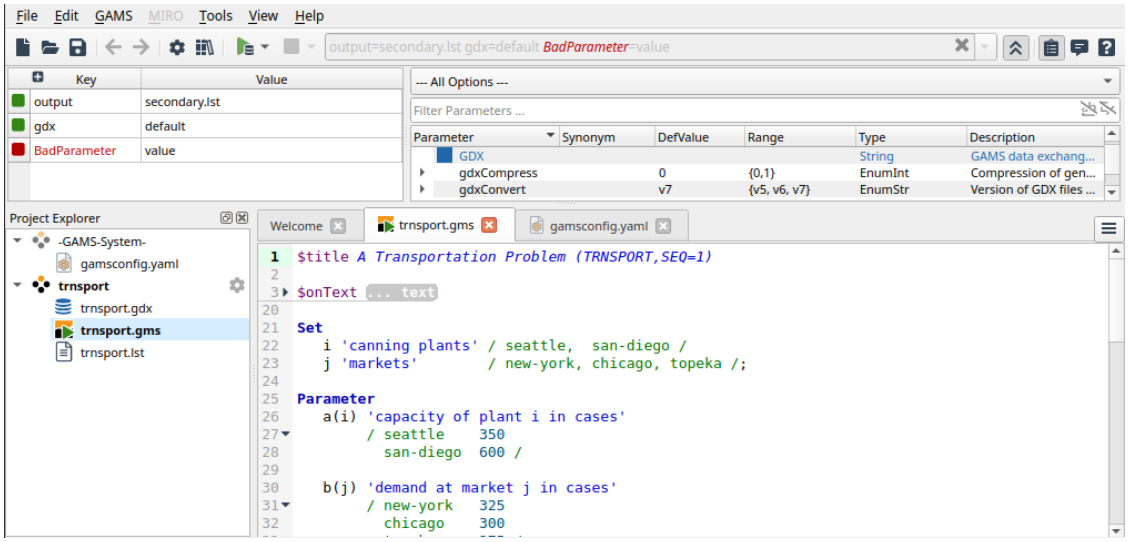


Figure 6.30 Extended GAMS Parameter Editor is shown when the button next to parameter combobox is clicked

The right pane of the extended editor displays the list of all valid parameter definitions, each contains `Parameter Name`, `Synonym`, `Default Value`, `Type`, and `Description`. The parameter definition entry with enumeration type (`EnumStr` or `EnumInt`) can be expanded to show all enumerated values of the parameter by clicking the bullet in front of the parameter entry, clicking the bullet again will hide all enumerated values of the parameter. Above the list of all valid parameter definitions is a filter box (with placeholder text `Filter Parameter...`) to allow to filter all parameter definitions. As a filter term has been typed into the search box the results will be displayed in the list below. More about the filter can be found in the Filter Section.

The parameter editor displays potential parameter errors in red color both in the combobox and in the left pane of the extended editor. When hovering a mouse over the error key or value, a tooltip with more detailed explanation of the error appears.

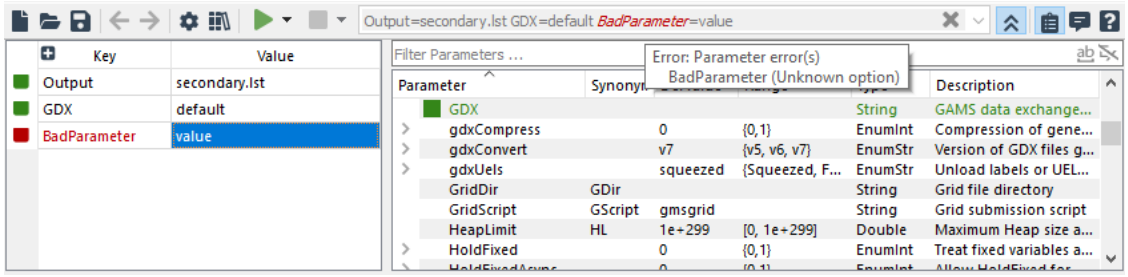


Figure 6.31 Parameter key turns red with pop-up tooltip when there is a potential error

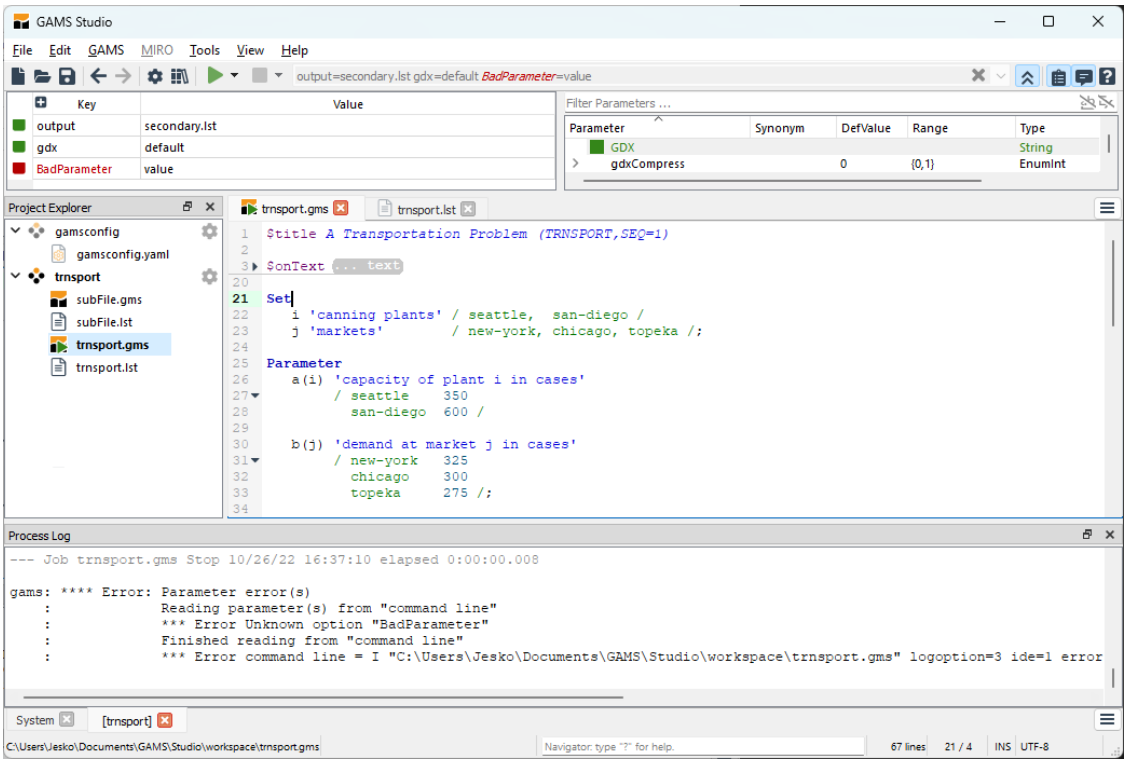


Figure 6.32 Process Log reports parameter errors when running the model with an unknown option parameter

When there are several entries of the same parameter, all other entries except the last entry will be ignored when gams executes the model. In such case, the combobox of parameter editor displays a warning in yellow and the icon in front of the parameter in the extended editor turns half-yellow in combination of another color, red or green, depending of whether or not there is a potential error.

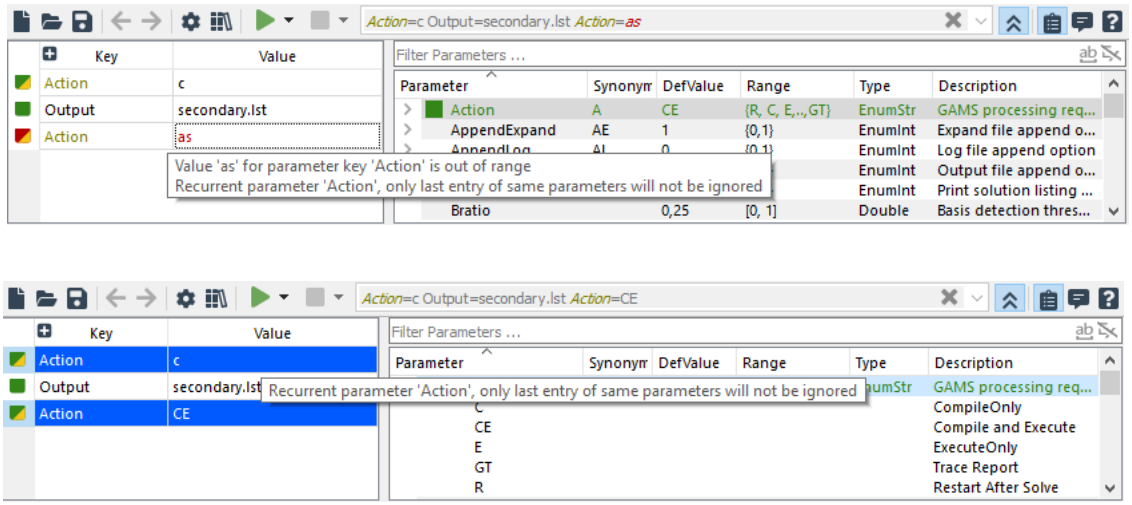


Figure 6.33 Parameter key turns yellow and the pop-up tooltip indicating a warning

See edit parameter key and value on how to edit parameter key and value that contains an error or a warning. See show all parameters of the same definition on how to locate all entries of the same parameter.

The followings describes all editing actions that can be performed via the extended parameter editor:

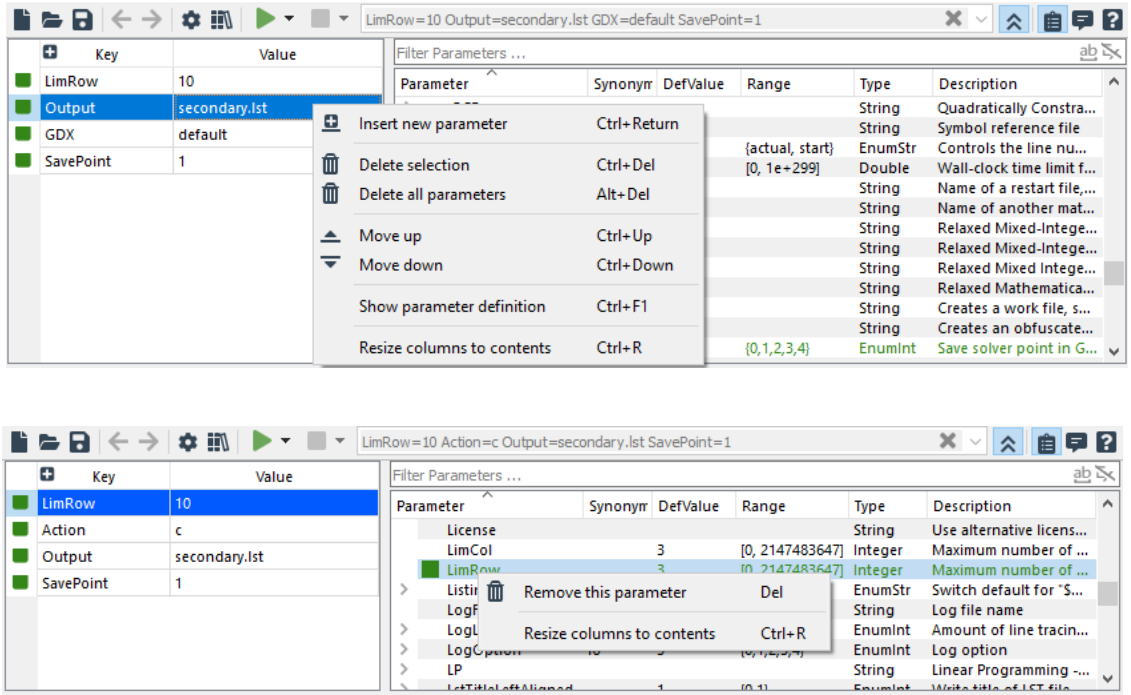


Figure 6.34 A number of actions that can be performed via the context menus of Extended Parameter Editor

- **Edit parameter key and value**: double click on key cell to edit `Key` and on value cell to edit `Value` of the entry. A drop-down list will suggest possible keys and values when possible. Press `Enter` to confirm the edit and press `Esc` to cancel the edit.

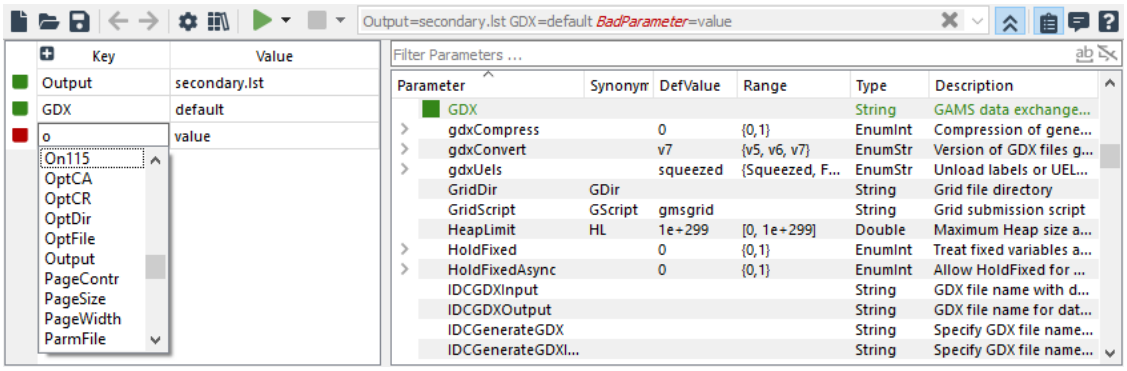


Figure 6.35 GAMS Parameter editor shows a drop-down list of all possible parameters started with 'o'

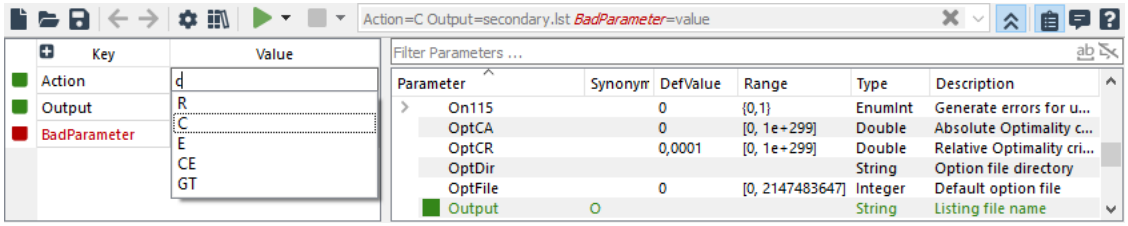


Figure 6.36 GAMS Parameter editor shows a drop-down list of all enumerated values of parameter 'action'

- **Add new parameter**: there are several ways to add a new parameter via extended parameter editor.
  - right click on the left pane of the parameter editor and choose `add new parameter` from the pop-up context menu
  - right click on the selected parameter entry of the parameter table and choose `add new parameter` from the pop-up context menu
  - click on the add new parameter button next to `Key` table header in the left pane

  A new entry will be added at the end of parameter entry table with dummy parameter key `[KEY]` and dummy parameter value `[VALUE]`. See edit parameter key and value on how to edit parameter key and Value.

  It is also possible to add a new parameter from the right pane of the parameter editor by double clicking at the definition entry in the right pane. The selected definition entry will be added as a new parameter entry at the end of table in the left pane. In case of double clicking an enumerated value entry of a parameter definition the definition entry will be added as a new parameter with the selected enumerated value. Otherwise the default value of the entry will be added.

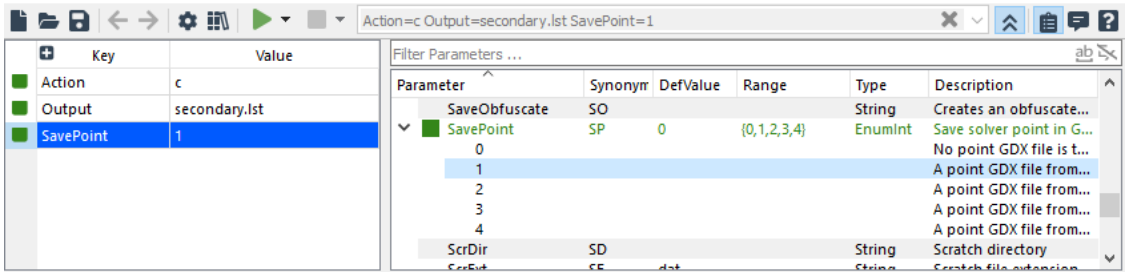


Figure 6.37 Parameter with enumerated value added by double clicking at the selected entry in the right pane

- **Insert new parameter**: right click on the selected entry and choose `insert new parameter`. A new entry will be inserted before the selected entry with dummy parameter key `[KEY]` and dummy parameter value `[VALUE]`. See edit parameter key and value on how to edit parameter key and value.

- **Move up**: right click on the selected entry and choose `Move Up`. The selected entry will be moved one position up the table. This action will change the order of parameters.

- **Move down**: right click on the selected entry and choose `Move Down`. The selected entry will be moved one position down the table. This action will change the order of parameters.

- **Delete selection**: right click on the selected entry and choose `Delete Selection`. The selected entry will be deleted from the table.

- **Delete all parameters**: right click on the left pane of the extended editor and choose `Delete all Parameters`. All entries will be deleted from the table.

- **Show parameter definition**: right click on the left pane of the extended editor and choose `Show parameter definition`. The parameter definition entry on the right pane will be highlighted.

- **Show all parameters of the same definition**: right click on the left pane of the extended editor and choose `Show all parameters of the same definition`. All parameter entries on the left pane will be highlighted.

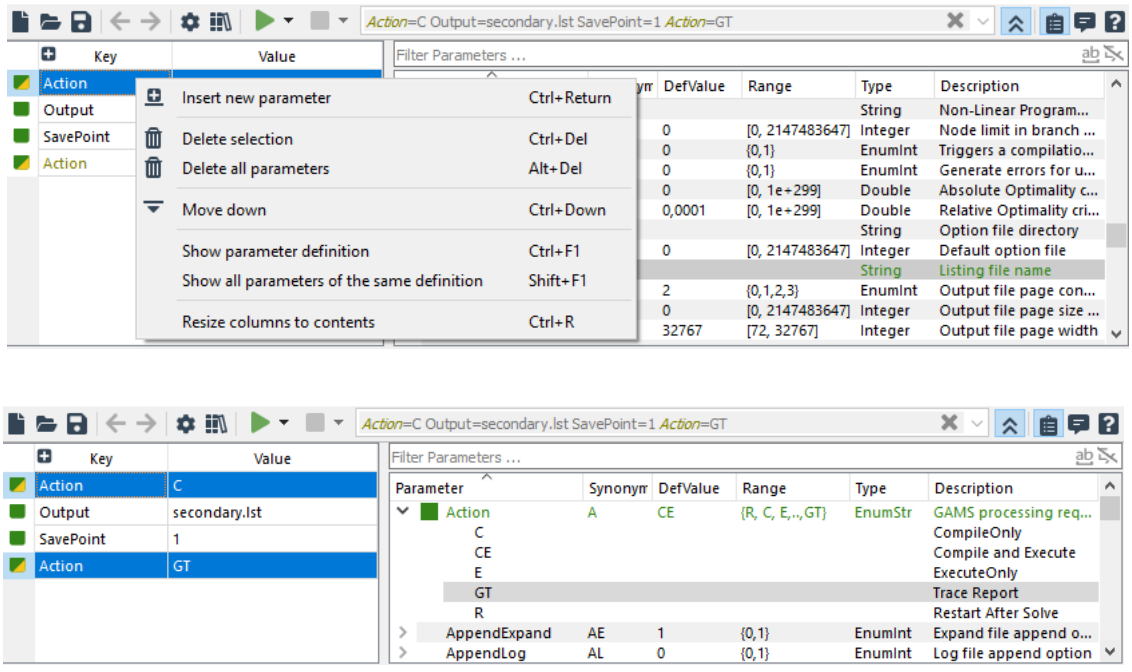


Figure 6.38 All entries of the same parameter are highlighted

- **Resize columns to contents**: right click on the left or right pane of the extended editor and choose `Resize columns to contents`. All columns will be resized according to their contents.
- **Add this parameter**: right click on the right pane of the extended editor and choose `Add this parameter`. The parameter with default value (if defined) of the selected definition will be added as the last entry on the left pane of the extended parameter editor. This action in the context menu is enabled only when there is no entry of this parameter on the left pane. Note that a double click on the definition will also perform add this parameter action. Performing the action on the parameter definition that has already been added will pop-up a dialog indicating that the parameter exists on the left pane and prompting for one of the followings:
  - replace existing entry If there is only one entry or replace the first entry and delete other entries if there is more than one entry, or
  - add new entry of the same definition, or
  - show details of all entries of the same parameter, or
  - abort the action

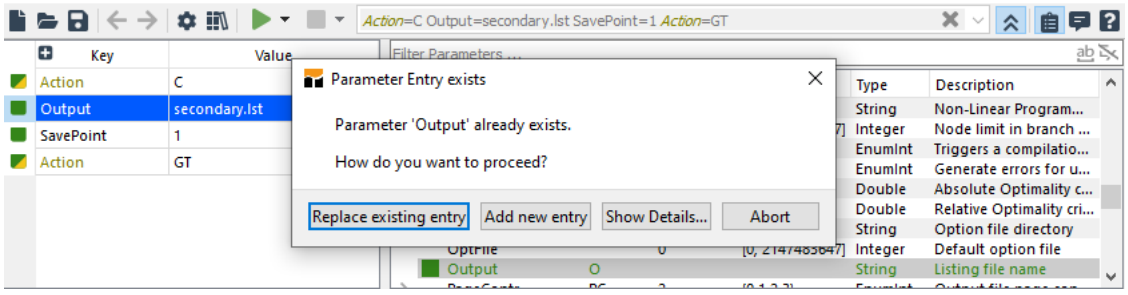


Figure 6.39 A pop-up dialog when adding the definition parameter that has already been added

- **Remove this parameter**: right click on the right pane of the extended editor and choose `Remove this parameter`. The parameter of the selected definition will be deleted from the left pane of

the extended parameter editor. This action in the context menu is enabled only when there is an entry of this parameter definition on the left pane. In case there are multiple entries of the same parameter, all entries on the left pane will be deleted.

Press F1 on a parameter entry in the extended parameter editor will activate the The GAMS Call and Command Line Par help page containing the detailed description of the GAMS parameter.

Summary of Actions and Shortcuts

Actions and their shortcuts that can be performed via Toolbar and GAMS parameter editor are:

Action	Shortcut	macOS	Description
New	**Ctrl - N**	**Command - N**	Open new file dialog
Open	**Ctrl - O**	**Command - O**	Open file open dialog
Save	**Ctrl - S**	**Command - S**	Save the current file. If the file does not exist on disk open the save file dialog
Settings	**F7**	**Command - ,**	Open the Settings dialog
Model Library	**F6**	**F6**	Open the Model Library Explorer dialog
Run	**F9**	**F9**	Run main file of current project with GAMS
Run with GDX Creation	**F10**	**F10**	Run main file of current project with GAMS and create a GDX file
Compile	**Shift - F9**	**Shift - F9**	Compile main file of current project with GAMS without execution
Compile with GDX Creation	**Shift - F10**	**Shift - F10**	Compile main file of current project with GAMS and create a GDX file without execution
Interrupt	**F12**	**F12**	(Gracefully) Interrupt currently running GAMS job
Stop	**Shift - F12**	**Shift - F12**	Stop currently running GAMS job
Show extended parameter	**Shift - Ctrl - 3**	**Control - Option - 3**	Open the extended GAMS parameter editor
Project Explorer	**Ctrl - 1**	**Command - 1**	Open Project Explorer
Process Log	**Ctrl - 5**	**Command - 5**	Open Process Log
Help	**F1**	**F1**	Open the GAMS Help.

Actions and their shortcuts that can be performed via the left pane of the extended GAMS parameter editor are:

Action	Shortcut	macOS	Description
Insert new parameter	**Ctrl - Return**	**Ctrl - Return**	Insert a new parameter
Delete selection	**Ctrl - Delete**		Delete the selected parameter
Delete all parameters	**Alt - Delete**		Delete all parameters

Action	Shortcut	macOS	Description
Move up	**Ctrl - Up**	**Command - Up**	Move the selected parameter up for 1 row
Move down	**Ctrl - Down**	**Command - Down**	Move the selected parameter down for 1 row
Select all	**Ctrl - A**	**Command - A**	Select all parameters
Show parameter definition	**Ctrl - F1**		Show definition of this selected parameter in the right pane
Show all parameters of the same definition	**Shift - F1**		Show all options of the same definition defined in the right pane
Resize columns to contents	**Ctrl - R**	**Command - R**	Resize the columns in the left pane to contents

Actions and their shortcuts that can be performed via the right pane of the extended GAMS parameter editor are:

Action	Shortcut	macOS	Description
Parameter Filter	**Ctrl - F**	**Command - F**	Jump to the focus to the Parameter Filter input field
Add this parameter	**Return**	**Return**	Add option in the left pane from the selected definition
Remove this parameter	**Delete**		Remove option defined by this definition from the left pane
Resize columns to contents	**Ctrl - R**	**Command - R**	Resize the columns in the right pane to contents

6.14.3.4 Integrated Help

The Help View is designed to integrate the navigation of the GAMS documentation into GAMS Studio. Press `F1` or choose `Help` > `GAMS Documentation` or check `View` > `Help` to start the Help View and browse the documentation. The start page of the Help View is the main document page available in the GAMS distribution that has been used to run GAMS Studio. See Home on how to always navigate back to the start page of the document. Alternatively, choose `Help` > `Studio Documentation` to directly jump the to start of the GAMS Studio documenation. Click Close button or uncheck `View` > `Help` to dismiss the Help View. GAMS Studio will remember the last viewed page along with its browsing history until GAMS Studio is restarted.

The Help View starts in docking state for the first time and can be docked around the editor in the central widgets area by either dragging the view to the desired location. Studio will remember last state of the Help View before it was closed.

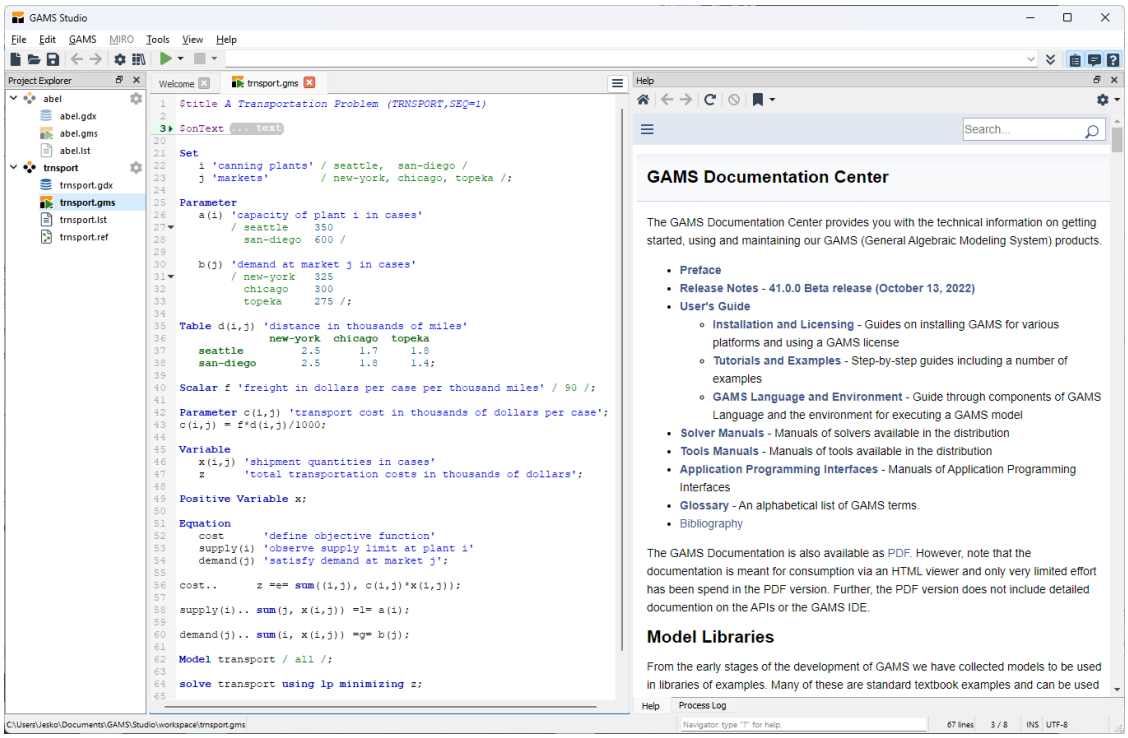


Figure 6.40 Help View when docked to the right of Code Editor

The Help View can also be floated by double clicking the title part of the Help View. Dragging or double clicking the title of the Help View again will dock the Help View.

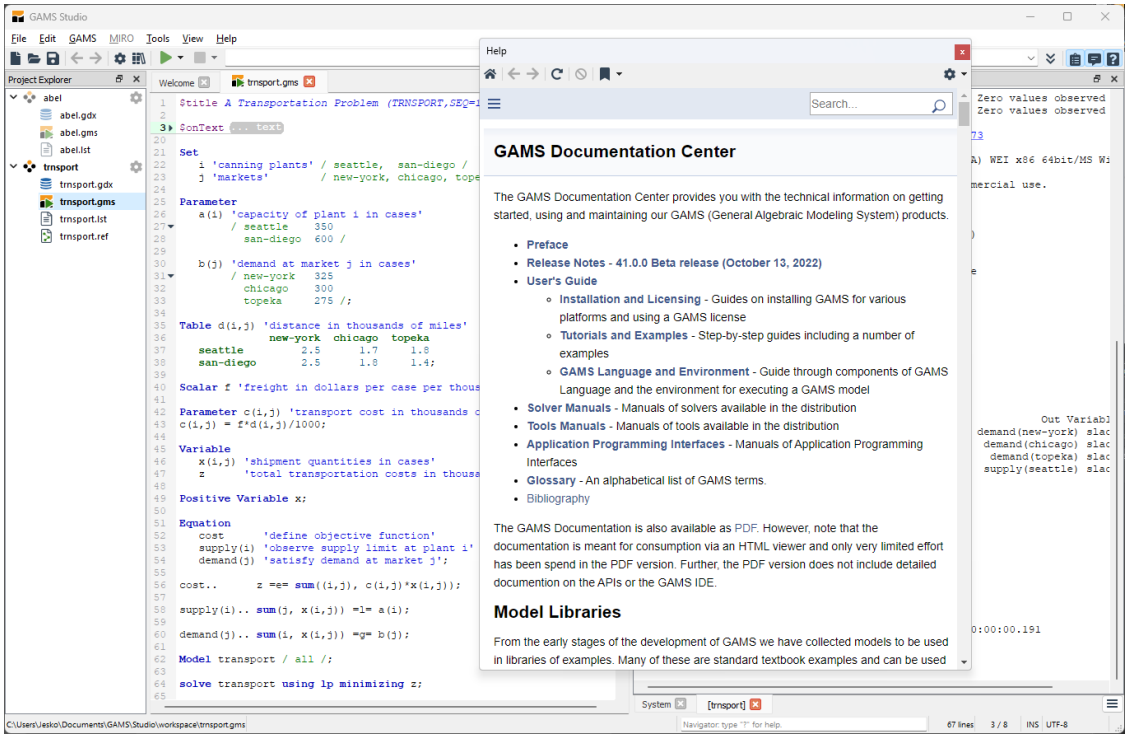


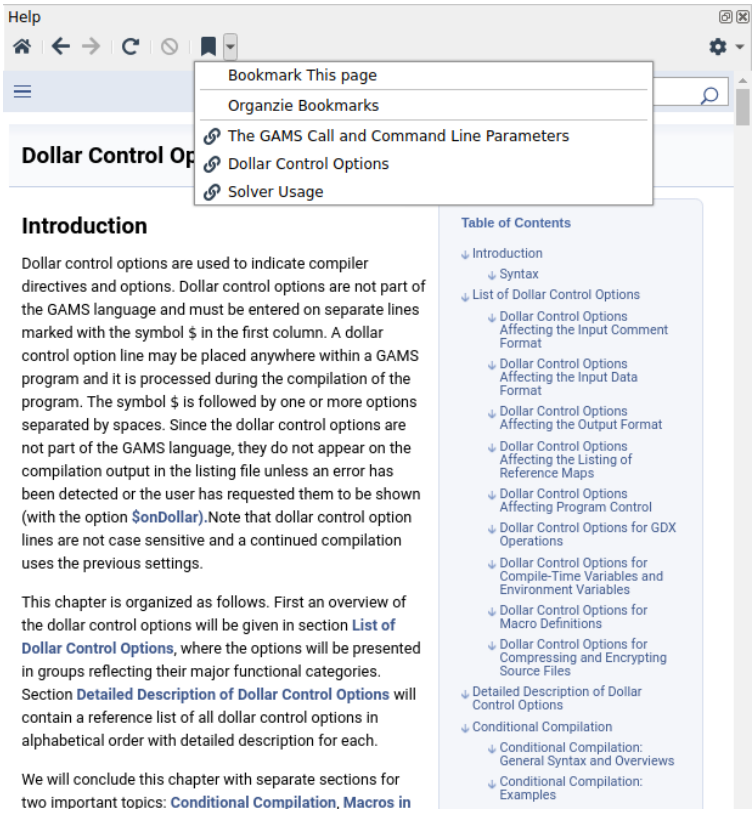
Figure 6.41 Help View when floated

Note that the Help View does not offer the full browsing features of a web browser. Though the help view

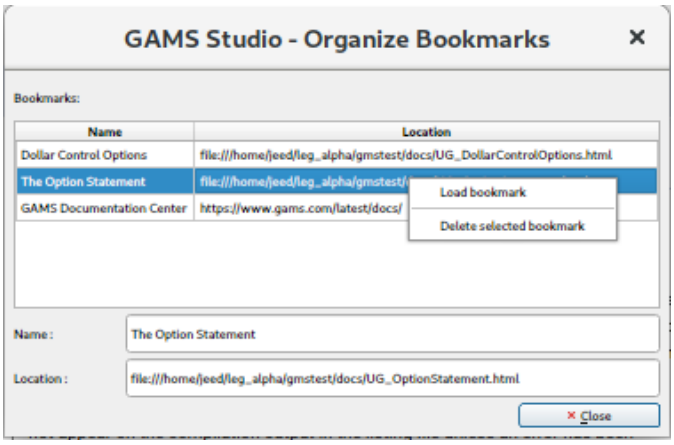
offers a Open this page in Default Web Browser button to use the full features of a web browser.

An overview of Help features:

- **Home:** Start the start help page `[GAMSDir]/docs/index.html`
- **Back:** Back to previous page
- **Forward:** Go to next page
- **Reload:** Reload the content of this page
- **Stop:** Stop loading the content of this page
- **Bookmarks:**
  - **Bookmark this page**: Store the title of the page as the name of bookmark together with its location. The entry of the bookmark will appear below the Organize Bookmarks section. Click on an entry to jump to the page that has been bookmarked.



  - **Organize Bookmarks**: Open the bookmark dialog to edit or delete a bookmark entry. Right click on the selected entry in the bookmark table, then either load the selected bookmark in the Help View, or delete the selected bookmark. The selected bookmark Name and Location can be edited from the lower part of the bookmark dialog and press `Enter`. Press Esc to cancel the edit. Click close button to dismiss the bookmark dialog.



- **Zoom In**: Zoom in the page to increase the font size. Press `Ctrl - +`, or `Ctrl - Mousewheel Up` or choose `View > Zoom In` menu to zoom in the page.

- **Zoom Out**: Zoom out the page to reduce the font size. Press `Ctrl - -`, or `Ctrl - Mousewheel Down` or choose `View > Zoom Out` menu to zoom out the page.

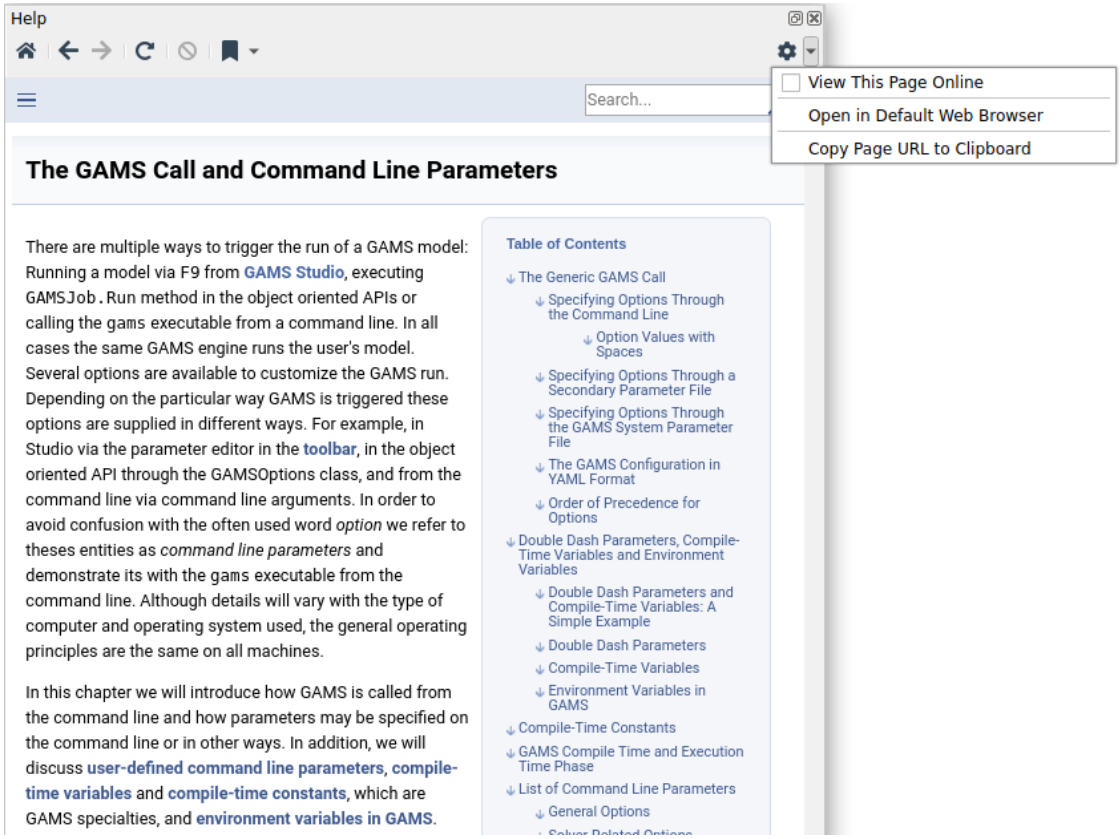
- **Reset Zoom**: Reset the font size of the page to its original size. Press `Ctrl - 0` or choose `View > Reset Zoom` menu to reset zoom in the page.

- **Help Option** :

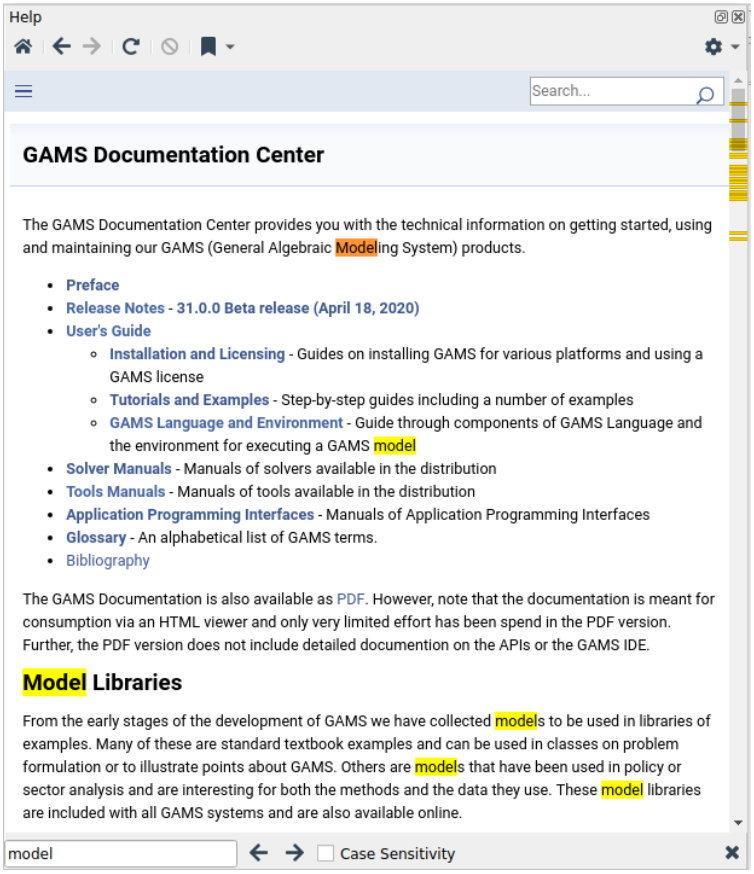
  - **View This Page Online** : Browse the same documentation on GAMS website. This, for example, allows to use more sophisticated search functionalities across all GAMS documentation of version 46.4 or to browse different version of documentation, or to browse the GAMS website from the Help View.

  - **Open in Default Web Browser** : Browse the same document in default web browser. As the help view does not offer the full features of a web browser, this allows to browse the documentation using the full web browser features in default web browser.

  - **Copy page URL to Clipboard** : Copy of the current page's URL to the clipboard. This, for example, help to see the URL of the currently viewed page of the document.



- **Find in page...** : Press `Ctrl - F` or choose `Edit > Search` menu to activate a search at the bottom area of the Help View. Type in a word to be found in the page. The word found in the page will be highlighted as it has been typed in and the number of found occurrences will be highlighted in the scrollbar of the Help View. Click `previous button` to find the previous occurrence, and click `next button` or press `Enter` to find the next occurrence. Check `Case Sensitivity box` to find a word case sensitively. Highlighting the word stays as the document is navigated from page to page until the keyword is clear or the search is dismissed or the Help View is closed or invisible. Click the close button or press Esc to dismiss the search.



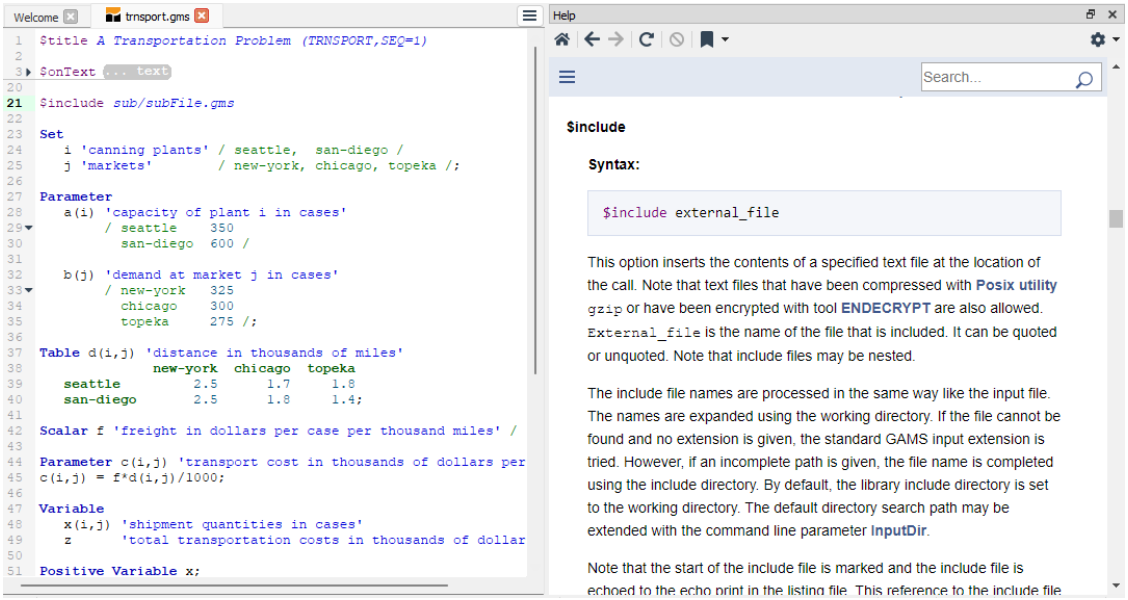
Help on Specific widgets

It is possible to jump directly from directly from different widgets (Welcome Page, Code Editor, Listing Viewer, GDX Viewer, Reference File Viewer, and Solver Option Editor) to the related part of the document.

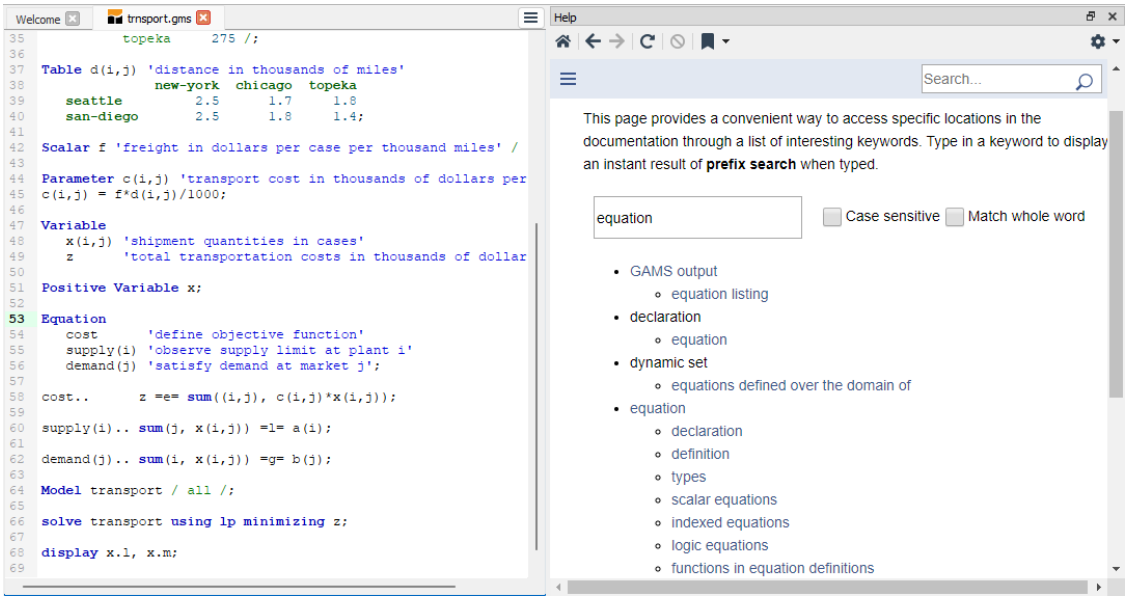
From Welcome Page, Listing Viewer, GDX Viewer, and Reference File Viewer, press `F1` will activate the GAMS Studio page and jump to the section corresponding to the widget. From GAMS Parameter Editor, press **F1** on a parameter entry in the extended parameter editor will activate the The GAMS Call and Command Line Parameters help page containing the detailed description of the GAMS parameter. From Solver Option Editor, press **F1** on an option entry in the editor will activate the related solver page containing the detailed description of the option.

From Code Editor, it is possible to jump to the document page that describes Dollar Control Options or to the index page that lists entries related to Data Types and Definitions and Language Items in Code Editor, as well as from the parameter editor to the GAMS parameter described in The GAMS Call and Command Line Parameters :

- Press `F1` on a Dollar Control Option within the editor will activate the page Dollar Control Options in the Help View and jump to the corresponding dollar control option described in Dollar Control Options chapter.



- Press `F1` on a keyword in Data Types and Definitions or in Language Items within the editor to activate the Index Page in the Help View that lists the index entries related to the keyword.



- Press `F1` on a parameter entry within the extended parameter editor to activate the help page displaying the detailed description of the GAMS parameters in The GAMS Call and Command Line Parameters.

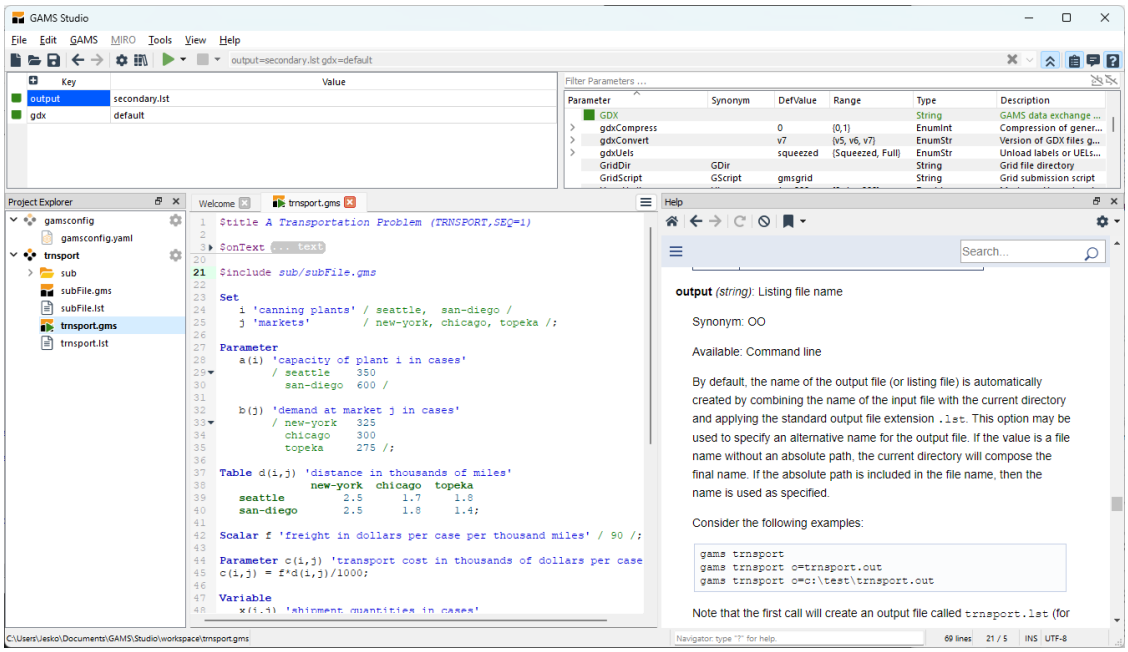


Figure 6.42 Help View when press F1 on parameter entry 'output' in extended parameter editor

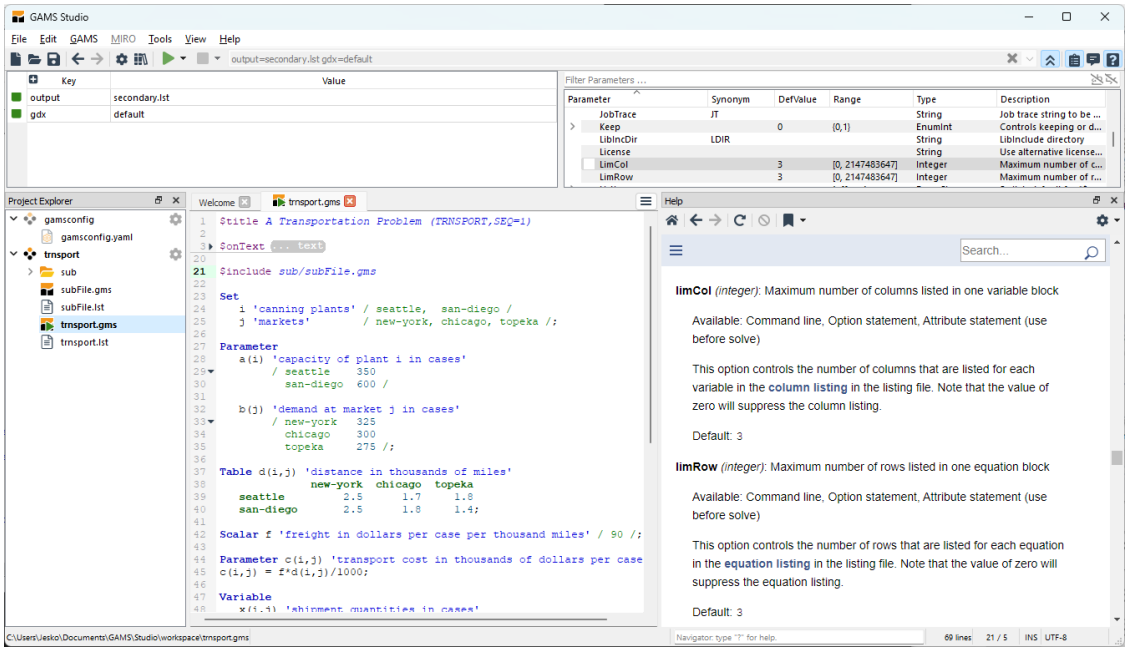


Figure 6.43 Help View when press F1 on parameter definition entry 'LimRow' in extended parameter editor

Summary of Actions and Shortcuts

Action	Shortcut	macOS	Description
Zoom In	**Ctrl- +** or **Ctrl - Wheel Up**	**Command - +** or **Command - Wheel Up**	Zoom in the page to increase the font size
Zoom Out	**Ctrl- -** or **Ctrl - Wheel Down**	**Command - -** or **Command - Wheel Down**	Zoom out the page to reduce the font size

Action	Shortcut	macOS	Description
Reset Zoom	**Ctrl - 0**	**Command - 0**	Reset the font size of the page to its original size
Search	**Ctrl - F**	**Command - F**	Jump to the help search box to search for a keyword in the current page

6.14.4 Debugger

GAMS Studio provides a debugger to detect errors. It is started by selecting `Run with Debugger` or `Step start with Debugger` from the `GAMS` menu or the Quick Access Toolbar. When pausing the process, GAMS Studio marks the line in the source code that will be processed next, and opens a temporary GDX file in the Pin View containing the current data state. It is possible to have more than one active debug sessions.

To pause a debug process you can

- set a **breakpoint**
- use **step**
- use **pause**

Breakpoints can be set to any line in the source code, although not all lines are valid spots where the process can be paused. GAMS can pause on lines that contains an operation. If a breakpoint is set to another kind of line (e.g. empty line, comment, or delcaration) it will be moved to the next possible line, or the last possible line in this file. The original line keeps a mark until the end of the debug session hinting its new line number.

Step can be triggered initially by selecting the menu `GAMS` > `Step start with Debugger` or by using the shortcut **F11** (**Meta - F11** on macOS). During the debug session **step** can be triggered in the debug pane below the editor and also using the shortcut.

Pause can be triggered when a debug session is running and not paused. This is available in the debug pane below the editor.

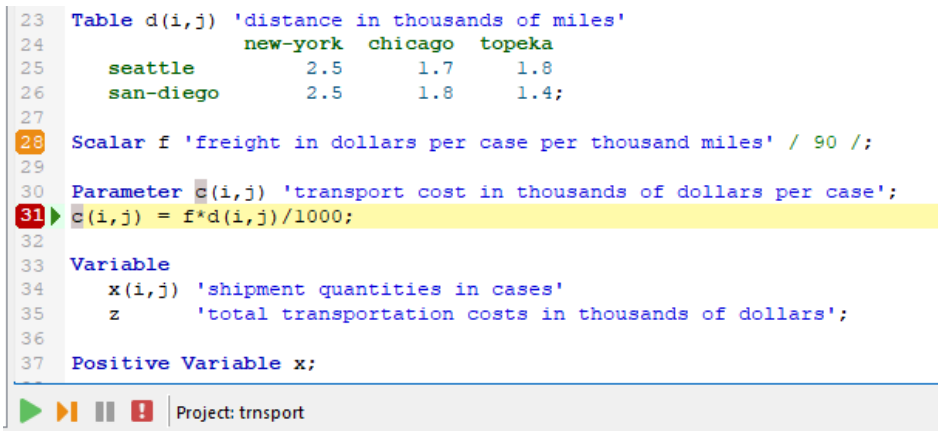


Figure 6.44 Debug pane and breakpoints

During a debug session the **Debug Pane** below the editor allows to control the flow:

- **Continue** - continues until the next breakpoint or the end of the program is reached
- **Step** - continues until the next possible pause line or the end of the program is reached
- **Pause** - pauses the process when the next possible pause line is reached
- **Stop** - stops the execution of this process
- the name of the active project follows right after the buttons

6.14.4.1 Debugger Shortcuts

Action	Shortcut	macOS	Description
Run Debugger	**F11**	**Meta - F11**	Run main file of current project in debug mode
Step Run Debugger	**Shift - F11**	**Shift - F11**	Run main file of current project in debug mode and pause at the first possible line

6.14.5 MIRO

GAMS Studio can be used to develop models for `GAMS MIRO`. Please go to the `GAMS MIRO documentation` to learn more about the features and workflow of the MIRO integration.

6.14.5.1 GAMS MIRO Shortcuts

Action	Shortcut	macOS	Description
Run base mode	**F8**	**F8**	Run main file of current project with MIRO in Base Mode
Run configuration mode	**Shift - F7**	**Shift - F7**	Run MIRO configuration mode

6.14.6 NEOS

GAMS Studio can remotely execute models on the `NEOS server`. The model is compiled locally before the compilation object is send to NEOS. Note that a GDX file can be created but always has the name `out.gdx`. On running a dialog pops up to enable further setup:

- "Email" - The email to submit a job. It can be predefined by setting NEOS_EMAIL in the GAMS User Configuration or by defining the environment variable NEOS_EMAIL.
- "Create a GDX file" - Create a GDX file even if the Command line parameter is not set.
- "Short priority" - The execution may start earlier and provides intermediate log output, but it is limited to an execution time of five minutes.
- "Long priority" - The execution may start later and provides no intermediate log output, but it is not limited in execution time.

The selections of the dialog are stored to the settings. To reveal the Terms of Use after checked `Don't show Terms of Use again` there is a checkbox in `Settings` > `Misc`.

The results of the remotely executed model are placed into a sub-folder named by the base name of the model. The log output of the remote execution has a different background color which can be configured in the settings dialog (Default Text, 3rd color). Links in the remote execution log output are replaced by their local counterpart.

6.14.7 GAMS Engine

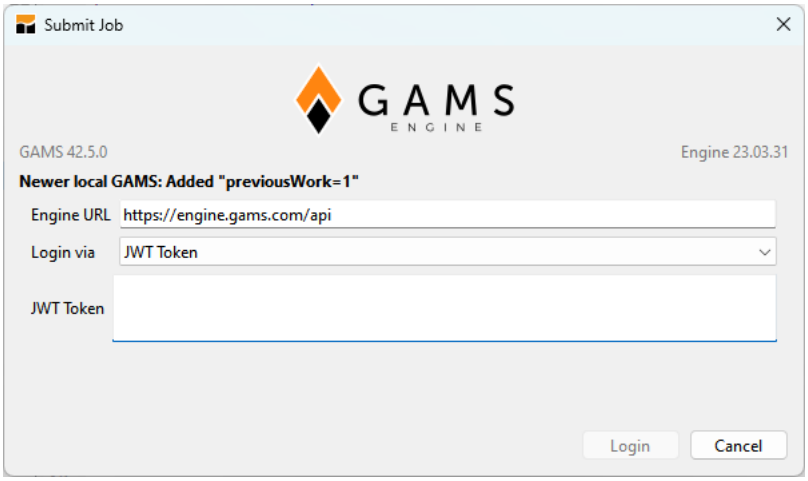
GAMS Studio can remotely execute models on your own `GAMS Engine server`. The model is compiled locally before the compilation object is sent to GAMS Engine. Additional files can be sent to the server by creating a file with the name of the model and the extension ".efi" (for "External FIles"). Each line in this file addresses an additional file. Files defined in the EFI file that should be actualized locally by changes from the GAMS Engine server can be marked by adding a < separated by a space.

Example: if an EFI file named `trnsport.efi` contains these lines:
`inc/trnsport.inc`
`inc/data.txt <`
and the model `trnsport.gms` is executed, both files will be transferred to GAMS Engine. After solving, the file `inc/trnsport.inc` will stay untouched but `inc/data.txt` will be replaced by the version returned from the server.

GAMS Studio provides an EFI editor to select the files. The context menu of a project in the Project Explorer allows to create or open the EFI file matching the current runnable GAMS model. With this editor files can be selected and selected files can be marked to be actualized locally. You can switch an open EFI editor to a text editor by selecting `Reopen as text` after changes have been saved. All operations (check, uncheck, select all, and clear) work on the visible files. In contrast the check state of a directory shows the select state of all files (visible and hidden by filter).

On running a dialog pops up asking the user to log in. If already logged in before and the login is still valid this first page is skipped. The login page offers several methods to authorize:

- *Login page*
  - "Url" - The `URL` to your GAMS Engine server
  - "Login via" - A list of available methods to authorize
    - "User" and "Password" - The username and password to login
    - "JWT Token" - A token can be used to login
    - "SSO" - An authorization provider
    - Additional predefined authorization providers (depending on the GAMS Engine installation)



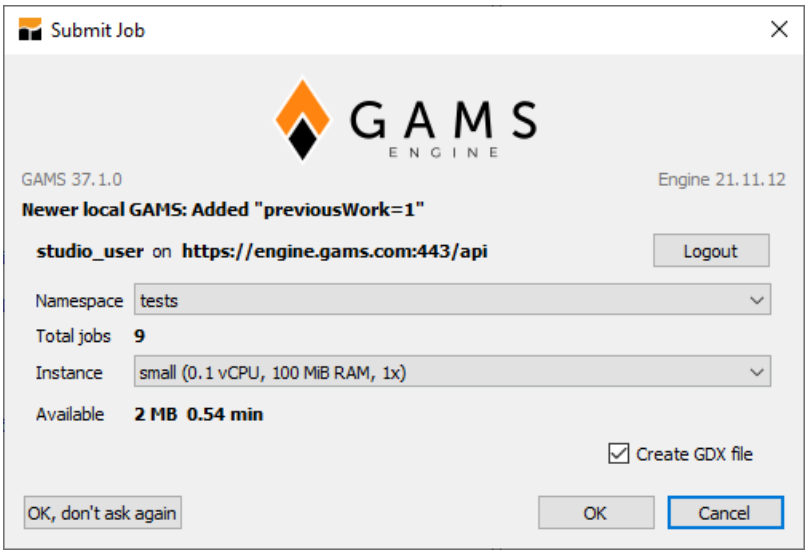
While changing the URL, `Studio` tries directly to access the server and requests version information. On success it gets the GAMS Engine version placed to the right and the version of the GAMS installation on the server to the left. If the local GAMS version is newer the command line parameter "previousWork=1"

will be added automatically if no "previousWork" is already present. The entries of the dialog (excluding the password) are stored to the settings. Further configuration can be done in the settings dialog.

To login via an authorization provider a SSO name can be entered or one of the predefined authorization providers can be selected. After selecting the "Login" button Studio contacts the provider and offers a code and a link to the provider where the code can be entered to verify the identity. For convenience the button "Copy and Visit" copies the code to the clipboard and opens the link in the systems default browser. Here the code can be entered using paste `Ctrl + V`.

Being logged in, the submit page allows to setup the remote job:

- *Submit page*
  - "Namespace" - The `namespace` to be used on the server
  - "Instance" - Allows to specify the user instance for GAMS Engine SaaS
  - "Create a GDX file" - Creates a GDX file even if the command line parameter is not set



To get back to the login page click the "Logout" Button. The job is submitted to the GAMS Engine server on clicking "OK". To always run the same setup, the "OK, don't ask" can be selected. For this Studio session the job is always submitted directly without showing this dialog. To see the dialog again, use `Ctrl-Click` on the Engine icon, or click on "Settings > Remote > Reactivate Dialog".

The selection of an user instance and the information of available space and time quota is only visible on connecting to a GAMS Engine SaaS server.

When a job is submitted, GAMS Studio continously shows the output received from the GAMS Engine server. As this may take some time it is possible to close Studio during the remote execution. In this case a dialog pops up asking weather the remote job should be kept or canceled. If "keep" has been chosen, the job token will be stored. On the next start of GAMS Studio a dialog will allow to resume to the job. On resume all pending messages will be polled from the Engine server and the execution will be followed until in the end the results are fetched, just like Studio hasn't been terminated in between.

The results of the remotely executed model are placed into the working directory:

- The LST and LXI files from the server are renamed to `"<model>-server.lst"` and `"<model>-server.lxi"` to protect the local files.
- All generated files will replace existing files of the same name (like GDX, REF, and LOG files).
- All file entries from the EFI file that end with `" <"` will replace the local existing file.

The log output of the remote execution has a different background color which can be configured in the settings dialog (Default Text, 3rd color). Links in the remote execution log output are replaced by their local counterpart.

6.14.8 Dialogs and Actions

6.14.8.1 GAMS Licensing

The GAMS Licensing dialog is part of the GAMS Studio Help menu. It provides information about the used GAMS as well as available solvers, which lists the solver licenses and their capabilities. Furthermore, the dialog can be used to install a GAMS license. To do so the steps below have to be applied.

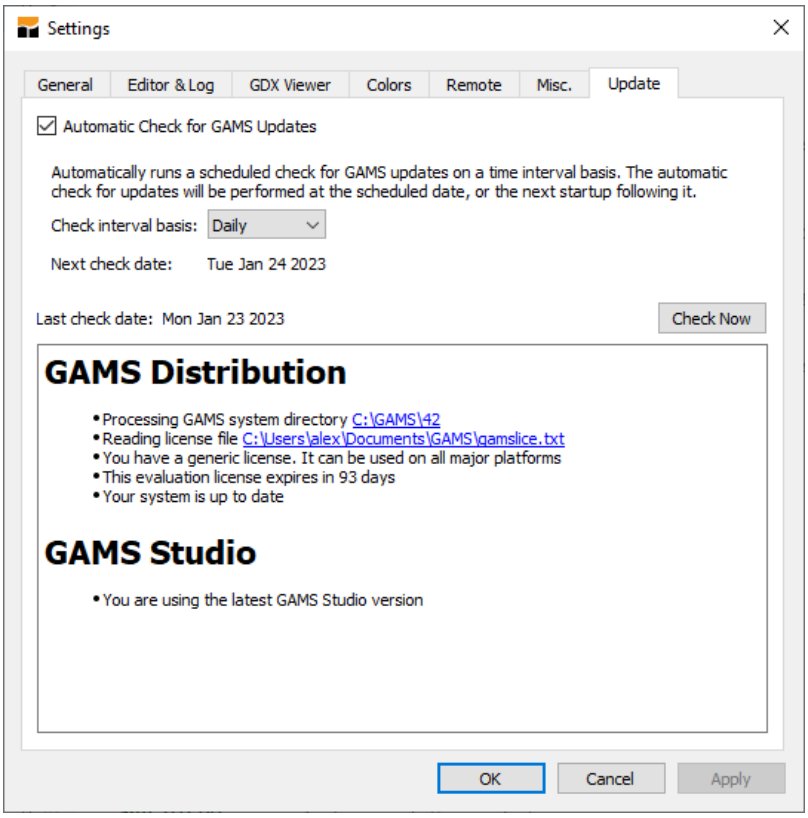
1. Copy the GAMS license to the clipboard
2. Open GAMS Studio and click on Help > GAMS Licensing
3. A messages box shows up if a GAMS license has been found on the clipboard. If 'Yes' is clicked the new license will be installed and presented via the GAMS Licensing dialog; otherwise the old license is shown.

Studio will also detect a license on the clipboard when it starts. If a license is detected the same message box will show up like if the GAMS Licensing dialog is used.

A license file can also be used as an alternative to the clipboard magic. To do so a GAMS license file needs to be selected via the 'Install License File' button. If the selected license is valid it will be installed; otherwise the previous license is kept.

6.14.8.2 Check for Update

Studio will check for a more recent GAMS version during startup. If a newer GAMS version has been found a hint will be displayed. In addition to that it is possible to check for GAMS and GAMS Studio updates at any given time. This can be done via *Help > Check for Update* or by opening *File > Settings > Update*. In both cases the following dialog is displayed. The dialog can be used to enable/disable the startup check and to configure the check interval.



6.14.8.3 Search and Replace

Action	Shortcut	macOS	Description
Open/Focus Search Dialog	**Ctrl - F**	**Command - F**	Opens the search dialog or focusses it if it is already open, if Text is selected it will automatically be entered as a search term
Find next	**F3** or **Enter**	**F3** or **Enter**	Jumps to the next match that fits all search criteria
Find previous	**Shift - F3**	**Shift - F3**	Jumps to the previous match
Find All	**Shift - Enter**	**Shift - Enter**	Starts a search, opens the page of search results and jumps to the first hit
Stop Search & Close Dialog	**Esc**	**Esc**	Closes the search dialog and stops an ongoing search. On second press any highlights of results will be removed

Pressing the `Ctrl - F` opens the Search and Replace window. If text is selected it will be pasted into the search field. Users can search specific files with it and do text replacement operations. Depending on which file type is openend in the editor, the search widget changes to visualize which actions are available. Due to technical limitations **.gdx** files cannot be searched. Also, replacing in read-only file types (e.g. lst or ref) is deactivated. For reasons of performance the search stops when reaching 50000 search results. This is visualized in all places where the number of matches is shown by showing "50000+" as the results number, indicating that there might be more results.

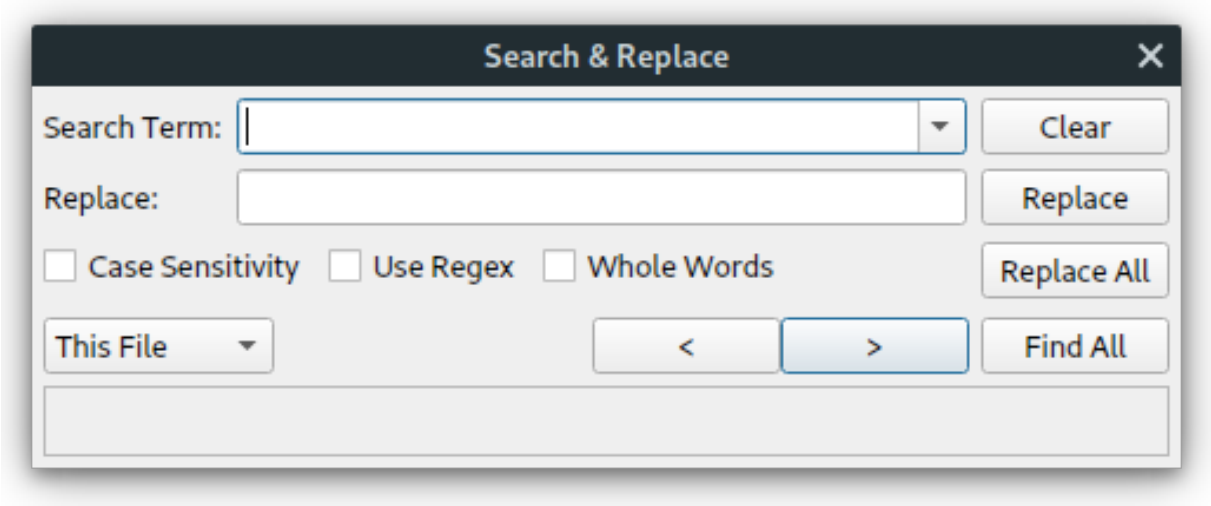


Figure 6.45 Search window in its default configuration

In the first row of the dialog there is the Search field, which takes the search term and saves a list of recent searches. Next to it is the "Clear" button that removes all highlights, results and clears both text fields. A shortcut to clear all results is to press the `Esc` key in the Code Editor. The next row houses items related to the replace functionality. Most of these are deactivated for files that are opened in read-only mode (e.g. **.lst**). First comes the replace input field where users can input the text which replaces the search term. The next button is "Replace": On the very first click it selects the next match, without replacing anything. On the next click it replaces the current selection with the replacement term and jumps to the next match in the same file, selecting it. Users can use this to click through a file, replacing words while keeping an overview over what is actually changed. Replacement actions can be undone by pressing either the undo button or hotkey `Ctrl - Z`.

"Replace All" replaces all matches *in the currently selected scope* after a pop-up is shown asking the user whether the replacement of **n** occurrences of **X** with **Y** is intended. In this pop-up the user can also decide to start a search instead, opening the results page showing all occurrences of the would-be replace action. Then come the options to narrow down a search. "Whole Words" excludes partial hits from matches. For example, when searching for "in" with this option checked only the word "in" is found but no occurrences in "information". "Case Sensitivity" sets if the case of letters in the search term matters. Reminder: The GAMS language is case insensitive, so this option can actually lead to incomplete results, depending on what it is used for. "Use Regex" actives an advanced search term interpretation mode and stands for *Regular Expressions*. When activated, instead of a single search term users can specify a pattern that matches an array of different words. `Click here` for further information about regular expressions.

In the next row is the scope selector. The scope of the search or replace action can be set here with five options available:

- "Selection" searches the text selected in the current file. The search dialog will give feedback if no selection is present. To remove the search selection press either `Esc` or the "Clear Selection" button. Keep in mind that starting a new search moves the search selection to the currently selected text.
- "This File" limits the search to the currently active file and is the default.
- "This Project" searches all files that belong to the same project as the currently opened file. These are all children of the same project in the Project Explorer.
- "Open Tabs" searches all files currently opened in Studio (except GDX files).
- "All Files" searches all files that appear in the Project Explorer and are searchable.
- "Folder" searches all files in a specified folder that can be located anywhere on disk and does not need to be opened in Studio beforehand. When opening a result the file will be added to the Project Explorer.

Next to it are the find previous occurrence and find next occurrence buttons, labeled "<" and ">" respectively. These will find and highlight the next word matching the search criteria either before or after the current text cursor position. Next to it there is the "Find All" button which starts a search in the scope selected by the user. In the output pane of studio a table containing all matches will open. Items can be double clicked to perform a jump to the result and the up and down arrow keys can be used to select quickly select and jump to results. Information about the file, the location of the match, plus some context information are also shown.

Depending on which scope is selected, additional elements of the search dialog will be shown:

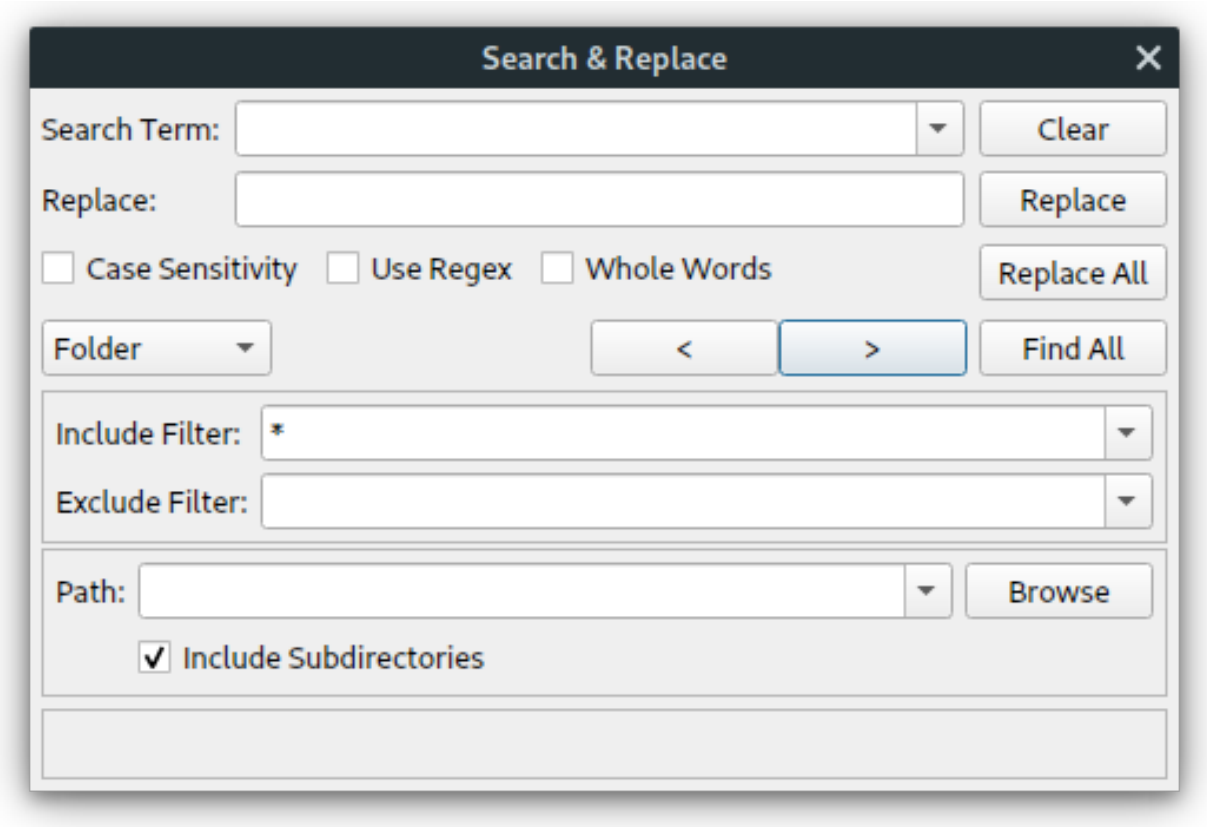


Figure 6.46 Search window fully expanded

The first section is the file filter section which is available for all modes except "Selection" and "This File". It gives the user the possibility to limit the search to certain files (Include Filter) but exclude others (Exclude Filter). Filenames can be specified by using a wildcard syntax. See the tooltips for some simple examples. The drop-down menus come with a few default options for GAMS specific file types but can also be used to enter own patterns. Comma-seperated lists of patterns are also supported. The second section is only available when "Folder" scope is selected and lets users set the path to search. It is a drop-down style textfield, so like the others it keeps a history of recent entries. Next to it is a browse button that opens the operating systems default folder picking dialog to easily find a path. Tooltips are available for most items in the search window and contain further information like keyboard shortcuts.

Workflow Tips

- Edit a file, hit `Ctrl - F` to open the search widget. The search field is focused automatically so you can start typing your search term. Pressing the `Enter` key is a shortcut for Find Next. Press `Enter` again or `F3` to step through all matches. Press `Shift - F3` to step backwards. Make changes to you document, press `Ctrl - F` again to re-focus the search widget without having to pick up the mouse.
- Enter a search term, and then `Shift - Enter` to trigger "Find All". This will open a list of results.
- The `Esc` key will close the search widget. Pressing `Esc` again will clear all highlighted search results.
- Use project or file prefixes when working with projects with many files. Use search filters to only search relevant files.
- Regular Expressions are a powerful tool and support - among other things - capture groups. This means that you can save a part of your match using parenthesis and reference the contained sequence in your replacement string. For example when some code references files with the naming

scheme of FILENAME_DD_MM_YYYY you could find all references using a regular expression like `(\w+)_(\d\d)_(\d\d)_(\d\d\d\d)`. To convert them into a more international and sorting friendly format you could use the following replacement string: `$4-$3-$2_$1` to change these to YYYY-MM-DD_FILENAME. Each pair of parenthesis created a so-called capture group. They are numbered counting from left to right starting with 1.

– It is also possible the reference a capture group in the search term itself. This is called a backreference. Creating backreferences works similarly, expcept that `\NUMBER` is used instead of `$NUMBER`. For example: `(\w+)\s*'\1'` would find symbols that have just their own name as descriptive text.

6.14.8.4 Model Library Explorer

The Model Library Explorer is used to search the different **model libraries** provided by GAMS and to retrieve their models in a convenient way. It can be opened either by choosing *GAMS > Model Library Explorer* from the menu, by using the shortcut `F6` or by clicking the corresponding icon in the Quick Access Toolbar.

Every library is presented in a separate tab. The search facility in the upper part of the Model Library Explorer allows for dynamic filtering of all model libraries simultaneously. As the search string is entered, the results are applied to the tabs representing the different libraries. If there are no more results on the selected model library tab, the first non-empty tab is automatically selected. This is undone as soon as the originally selected model library contains results again. The parenthesis enclosed numbers indicate the number of models found for the specific library and the current search input. More about the filter can be found in the Filter Section.

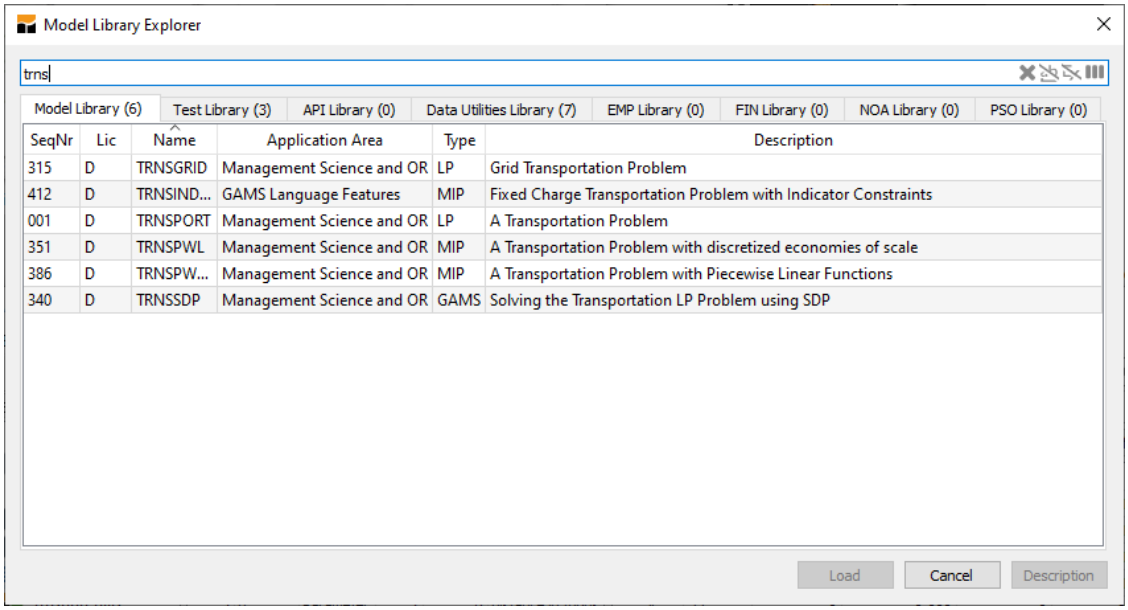


Figure 6.47 Searching for `trns` in the Model Library Explorer

Beside a short description, several models have a longer and more detailed description available. Selecting a model and clicking on the `Description` button in the lower right corner opens a pop-up dialog showing further information about the model. A model can be opened either by clicking on the `Load` button, by double clicking or by selecting the model and pressing the `Enter` key.

User Libraries

Additionally to the model libraries distributed by GAMS, it is possible to access user defined model libraries by providing a GLB file along with the corresponding files belonging to the models in the library. The Model Library Explorer looks into a specific location for user defined model libraries. This location can be accessed or changed by opening the Settings dialog (`File > Settings`) and switching to the `Misc`. tab. The `Arrow` button will show the location where the Model Library Explorer tries to find user defined model libraries, the `Folder` button allows to change the location. Adding a new library is done by copying the required files into a subdirectory in this location. See `Creating a User Library` on how to create a custom model library. Newly added libraries require a restart of the Model Library Explorer to become visible.

6.14.8.5 Settings

The GAMS Studio settings dialog can be accessed via the *File > Settings* menu entry or with the hotkey `F7`. Users can change certain aspects of Studio like behavior or appearance. Settings are categorized loosely on five tab pages.

General page

On the **"General"** tab users control general settings of **Studio**. Some additional hints:

- "Default GAMS Studio workspace". This path is used as a default location for models imported from the Model Library Explorer.
- "Automatically save modified files before running GAMS" is an option that saves all modified files when the user starts a GAMS run. Thereby, previously unsaved changes will be incorporated into that GAMS execution without the need to go through all files and make sure that changes were saved manually.
- "Open .lst file after running GAMS" automatically puts the generated lst file into the foreground after each GAMS run.
- "Jump to first compilation error" is an automatism that aims to make working with Studio a little easier. When compilation errors occur, the editor and the log both jump to the first error detected if this option is activated. This also applies when the option *Open .lst file after running GAMS* is activated. Studio then opens the lst file generated and jumps to the first error in it. If you navigate back to the gms file, the view has also moved to the first error found.
- "Open file in current project by default" switches the menu *File > Open* (`Ctrl-O`) to always open a file in the current project, even if it exists in another project. The alternative open (`Ctrl-Shift-O`) is changed to *File > Open in new project...*.
- "Open file finds or creates new project" switches the menu *File > Open* (`Ctrl-O`) to find the file in any project and open it. If not found, a new project is created. The alternative open (`Ctrl-Shift-O`) is changed to *File > Open in current project...*.
- At the bottom of this tab there is a section to backup all settings. When pressing the **Export** button a dialog pops up, asking for a filename. This name can be chosen freely and the file can than be copied to a USB stick for sharing, or just be saved for later. This is handy if you are planning a fresh install or want to copy your personal settings to a different machine. The button **Import** is used to load a saved settings file. These settings are applied immediately and the settings dialog will be closed automatically.

Editor & Log page

The **"Editor & Log"** tab contains many self explanatory appearance options. Most notably font settings which is used for all monospace fonts like text editors and logs. The font size is used also to calculate the size of proportional fonts (see Zoom Groups)

- "Enable auto-indentation" activates smart line break behavior. When pressing the `Enter` key a line break is inserted and the indentation of the line of origin is copied instead of moving the cursor to the first column.
- "Highlight word under cursor without selection" changes the highlighting behavior of word occurrences. While the default behavior highlights other occurrences of the same word when double clicking or selecting a word, activating this option changes the behavior so that every time there is a word under the text cursor, all of its occurrences will be highlighted. Only full words and no partial matches are highlighted.
- "Tab stop size" lets the user decide how many tabs will be inserted when pressing the Tab-key. GAMS Studio always replaces inserted tabs with spaces.
- "Fold DCO blocks on open" folds all foldable blocks of the types "Text", "Echo", "Put", and "Fold"
- "Syntax Highlight bound" stops highlighting lines with more than the given count of characters. Highlighting long lines is a very time consuming process which can be controlled this way. It's likely that the highlight will get out of sync for one or more lines. By placing syntax relevant characters to the next (shorter) line this side effect can be reduced. To switch off highlighting set this value to 0. To always turn on highlighting set it to -1.
- "Clear process log before GAMS execution" empties the log before running a gms file. If deactivated old log output is kept but colored in a lighter gray so users are still able to identify that they are looking at an old run instead of the most current one. One can also decide if the log should be written to disk as well and if so, how many old versions of the same log should be kept on disk as backup. Note, that in addition to this log file written by Studio, one can instruct GAMS itself to write a log file using the GAMS parameter logOption, which could cause a file naming clash.

GDX Viewer

- "Symbol View" controls if the list view or the table view is used as the default view when a symbol is displayed for the first time.
- "Attributes" allows to control which of the variable and equation attributes should be visible per default.
- "Preferences" controls different default settings for symbol data:
  - "Squeeze Defaults" affects variables and equations only and can be used in order to hide all columns that have the default value of the respective variable or equation type.
  - "Squeeze Trailing Zeroes" allows to turn on/off the truncation of trailing zeroes.
  - "Format" is used to control the default format of numerical values. For detailed information see the corresponding section in the GDX Viewer documentation.
  - "Precision" specifies the number of decimals or the number of significant digits depending on the chosen format. For detailed information see the corresponding section in the GDX Viewer documentation.
- "Decimal Separator in Copied Data" controls the decimal separator used when copying data from a GDX symbol. Note that this setting affects the copied data only but not the diplayed data itself. The following options are available:
  - "Use Studio default": Per default `.` is used as decimal separator in copied data.
  - "Follow system language": Use the decimal separator from the system language settings.
  - "Use custom character": This allows to specify a custom character.

Colors page

The **"Colors"** tab enables you to configure several colors. Studio provides two fixed themes: the "dark" theme and the "light" theme. Depending on the your operating system, this can be selected by the system and/or manually. You can use this to set Studio to use a darker color palette for when you like to work in the dark and don't like to get blinded by bright windows.

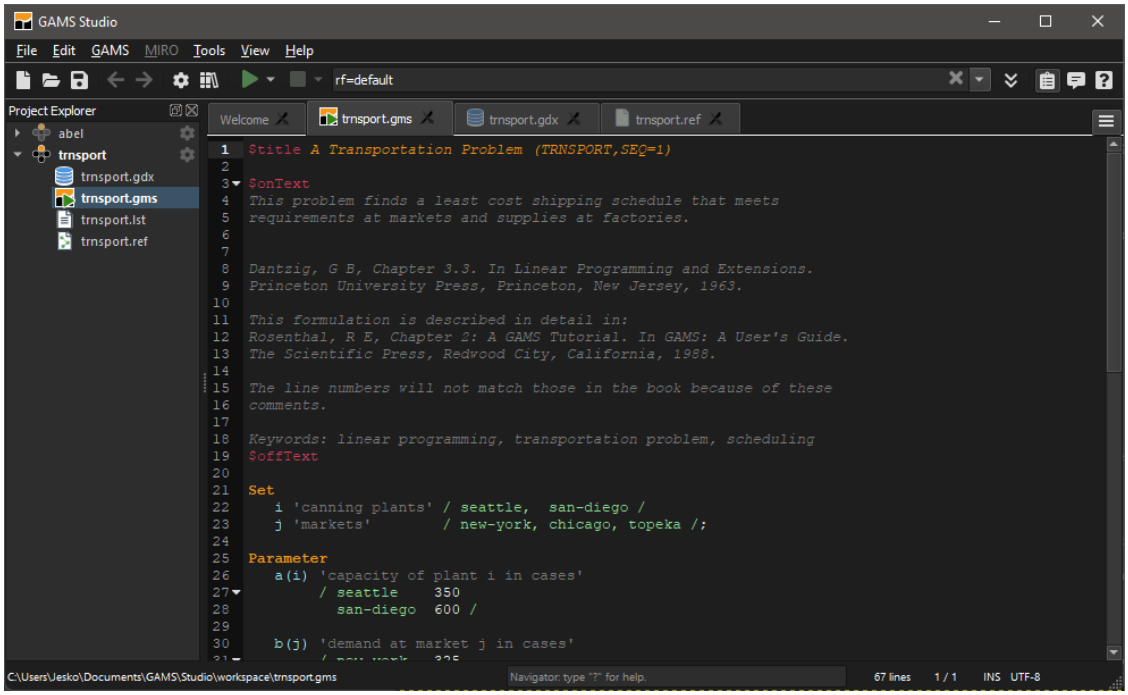


Figure 6.48 Studio using a dark theme

The fixed themes can't be modified, modifying a value implicitly creates a copy of the theme where the changed value is applied. After copying a theme, the new user theme can be adapted to your needs. Additionally these user themes can be exported and imported to a file each.

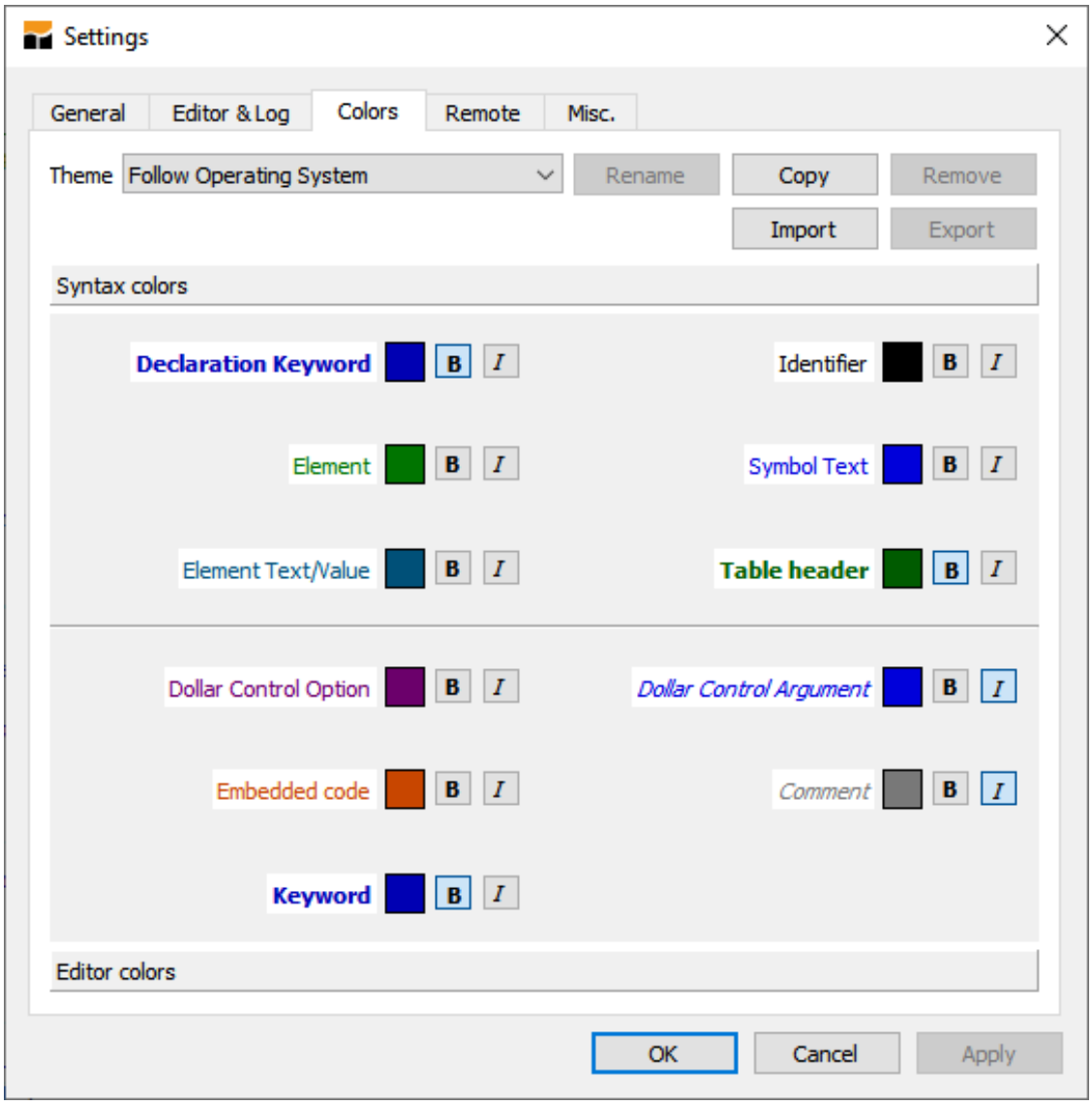


Figure 6.49 Color Settings

The colors are listed in two sections:

- On the "Syntax colors" page you can define a *text color* and the attributes *bold* and *italic*.
- The "Editor colors" page allows to change the colors of *text*, *background*, and *second background* (for blinking). The very first entry on this page, the "Default Text", is the base of the syntax colors.

Remote page

The **"Remote"** tab is used to set various remote settings.

- In the MIRO section the location of your local MIRO installation can be defined.
- The checkbox "Always agree to NEOS" allows to hide the "Terms Of Use" in the submission dialog of NEOS.
- In the GAMS Engine section it can be defined if a login token should be stored to the settings to stay logged in on restarting Studio. The next line allows to define how long the login is valid.
- If the dialog has been switched invisible by clicking "OK, don't ask", the button "Reactivate dialog" switches it on again.

Misc page

The **"Misc"** tab contains some lesser common settings.

- "GAMS source file extensions" allows to add own file extensions that **Studio** counts as *runnable GAMS*. File types added here can be executed, set as main file, and are assigned to the GAMS Syntax Highlighter. The field accepts a comma separated list of file extensions.
- "Auto-reload extensions" Specific extensions that should be *reloaded automatically* can be added here. Files of that type are reloaded into the editor if changed externally. If they also were changed internally, a dialog asks how to proceed. The field expects a comma separated list of file extensions.
- "User model library" shows the folder that is used to find user generated model libraries, see User Libraries. This folder can be changed to a custom location. Up to 10 locations are stored in the history. To remove the current entry from the history use the trash-bin button. The arrow-button opens the file browser in the current folder.
- "Reset History" clears the list of last projects and files used on the Welcome Page
- "Reset Window" resets the appearance of Studio if something went wrong.
- The next group contains a list of options for the Solver Option Editor.

6.14.8.6 Delete scratch directories

GAMS creates working folders for running processes in your working directory. Normally these are automatically removed when a run finishes. However, if a run was interrupted sometimes these working directories remain. That in itself is not a problem as GAMS will pick a new name for the next run if the default name for the working directory is already taken. To automatically clean up the user workspace, Studio offers an action to do this. In the menu under **GAMS > Delete scratch directories** the user has the possibility to remove unwanted, autogenerated directories. Before deleting anything, there will be a popup asking the user for confirmation and also showing the path of the directory that will be cleaned. If the user choses "Yes", all directories that start with 225 followed by one or two letters will be deleted in the given path, which is usually the default workspace. This action is automatically triggered when GAMS has exhausted the pool of possible scratch directory names, but not without asking the user for confirmation first. To avoid any potential errors this action should only be triggered if there are no running GAMS jobs at the moment.

6.14.8.7 Full Screen & Distraction Free Mode

Especially small devices suffer from sparse screen real estate. Studio comes with a few features to make the most of the space that is available on your machine.
The Full Screen mode is activated by pressing `Alt-Enter` on Windows and Linux systems, while on macOS the shortcut is `Command-Control-F`. This enlarges Studio over the whole screen, covering the taskbar and other operating system specific items. The same shortcuts also exit Full Screen mode.
As an alternative, there is also the Distraction Free mode. Some users want to keep their taskbar so with the shortcuts `Ctrl-Alt-Enter` on Windows and Linux or `Cmd-Option-Enter` on macOS hide all Studio widgets and enlarge the central widget to the size of the studio window. You can use this to get the log-, file- and help view out of the way and concentrate writing. Pressing the shortcut again brings everything back. Running your GAMS modell in the Distraction Free mode will also open the log, so you are not missing out on anything.

Action	Shortcut	macOS
Enter/Exit Full Screen	**Alt - Enter**	**Command - Control - F**
Enter/Exit Distraction Free Mode	**Ctrl - Alt - Enter**	**Command - Option - Enter**

6.14.8.8 GDX Diff

The GDX Diff dialog serves as a graphical interface to the command line tool GDXDIFF. It compares the data of two GDX files and writes the differences to a third GDX file. The dialog can be opened by choosing `Tools` > `GDX Diff` from the menu.

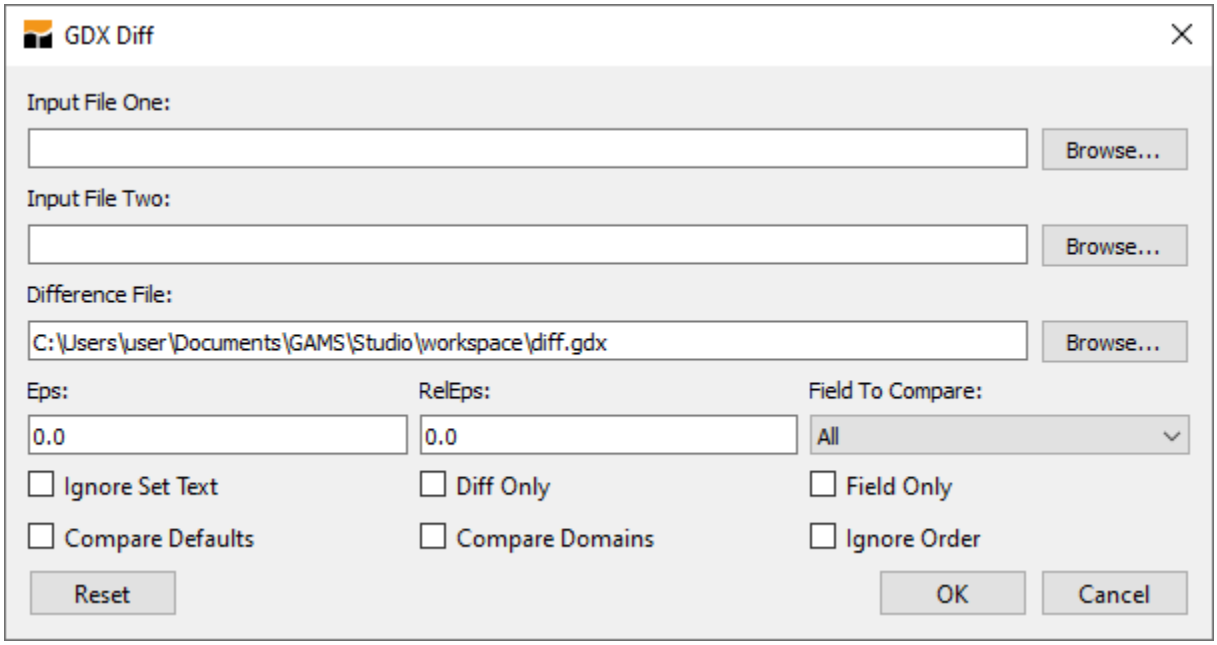


Figure 6.50 GDX Diff dialog

The dialog consists of two parts. The upper part is used to specify the two GDX files for the comparison. The resulting GDX difference file is determined automatically as long as no manual changes have been applied. The default schema for the resulting GDX difference file is the location of the first input GDX file and the file name `diff.gdx`. The difference file will be updated whenever the first input is modified to an existing file. If the GDX Diff dialog gets opened and does not contain a valid first input file, the most recently used directory is used as the location for the resulting difference file instead.

The lower part of the dialog contains different options that can be used in order to control the behavior of the comparison:

- **Eps**: Absolute difference for comparisons. If the difference between two values exceeds Eps, a difference will be reported. Default is `0.0`.
- **RelEps**: Relative difference for comparisons. If the value of RelEps is exceeded, a difference will be reported. Default is `0.0`.
- **Field To Compare**: The specified subfield is the only field used for deciding if a variable or equation is different. List of possible values:
  - **All**: All fields
  - **L**: Level
  - **M**: Marginal
  - **Lo**: Lower Bound
  - **Up**: Upper Bound
  - **Prior**

  - **Scale**

- **Ignore Set Text**: Allows to ignore explanatory text of set elements.
- **Diff Only**: Differences for variables and equations will be written as parameters. Each parameter will have an additional index which is used to store the field name. Only fields that are different will be written. This option can not be used in combination with `Field Only`.
- **Field Only**: Used in combination with `Field To Compare`. The variables and equations will be written as parameters for the selected subfield. This option can not be used in combination with `Diff Only` and it requires a value for `Field To Compare` other than `All`.
- **Compare Defaults**: Controls if default values for symbols, e.g. 0 for parameters, are reported as differences if they are found in one file and missing in the other or not.
- **Compare Domains**: Controls if different domains for the same symbol are reported as a difference.
- **Ignore Order**: Ignore UEL order of input files. This can reduce the size of the output file.

As soon as the `OK` button is clicked, a validation check on the given input is performed and the two input files are compared. If the specified difference file is already open in a GDX Viewer, the corresponding tab remains open while the underlying GDX file is detached. As soon as the file was updated, the GDX Viewer is reinitialized. In case the GDX file was not open already it gets opened in a new tab. The `Cancel` button closes the dialog and stops a process that has not been finished yet. The dialog needs to be kept open for longer running comparisons of large input files. The `Reset` button can be used to reset the whole dialog to its default state which also enables the automatic pre-population of the difference file path in case it has been disabled before due to a manual change.

The GDX Diff dialog also has an integration in the context menu of the Project Explorer. As soon as either exactly one or two GDX files are selected, the context menu contains an entry for opening the GDX Diff dialog with the file(s). If only one file is selected, the input field that is empty is populated with the corresponding file path. If both fields are filled, the first input file path gets overwritten. In case of two selected GDX files, both input fields get overwritten.

6.14.9 Terminal

GAMS Studio is able to open a native terminal via the main menu `Tools` > `Terminal` or the shortcut `CTRL - T`. Alternatively, a terminal can be opened by right clicking on any entry of the "Project Explorer" and selecting `Open terminal`. The terminal points to the workspace of the current project. On macOS the terminal location is always the user's home directory.

6.14.10 Command Line Options

GAMS Studio can be started with additional command line options that change the behavior or trigger certain functionality at start up time. The syntax for starting GAMS Studio from the command line is

```
studio [options] [files]
```

While `files` is one or more files to be opened by GAMS Studio, the following `options` can be supplied:

- **-?, -h, --help** : Displays the help.
- **-v, --version** : Displays version information.
- **--ignore-settings** : Ignore settings files. Studio will load with default settings without writing them.
- **--reset-settings** : Reset all settings to default. Studio will load with default settings and save them.
- **--reset-view** : Reset studio visually without resetting settings. Useful when a widget got lost or studio started on a disconnected monitor.
- **--gams-dir path_to_gams** : Point Studio to a specific GAMS installation.

6.14.11 General Shortcuts

This section lists any shortcuts that are global to Studio. Shortcuts for more specific areas are mentioned in their corresponding section.

6.14.11.1 Studio Workflow Shortcuts

Shortcut	macOS	Description
Ctrl - 1	**Command - 1**	Set focus to the Project Explorer. After that the arrow keys can be used for file navigation.
Ctrl - 2	**Command - 2**	Set focus to main editor.
Ctrl - 3	**Command - 3**	Set focus to the command line option input field.
Ctrl - 4	**Command - 4**	Opens the tab browser which has a list of all open tabs. Focus is on an input field that can be used to filter tabs. Pressing Return opens the highlighted file.
Ctrl - 5	**Command - 5**	Set focus to log widget.
Shift - Ctrl - 3	**Control - Option - 3**	Opens and closes extended command line option editor.
Ctrl - Tab		Toggles through the open tabs of the central widget from left to right.
Ctrl - Shift - TAB		Toggles through the open tabs of the central widget from right to left.
Alt - Return	**Command - Control - F**	Full Screen
Ctrl - Shift - T	**Command - Shift - T**	Restore recently closed tab
Ctrl - Shift - W	**Command - Shift - W**	Close all tabs
Ctrl - W	**Command - W**	Close current tab. Alternatively, a tab can be closed by clicking the middle mouse button.
Ctrl - Q	**Command - Q**	Close GAMS Studio

6.14.11.2 Studio Dialogs

Shortcut	macOS	Description
F1	**F1**	Open the GAMS Help. If the word under the cursor is a GAMS keyword a search in the documentation document will be started
F6	**F6**	Open GAMS Model Library Explorer
F7	**Command - ,**	Open Studio Settings
Ctrl - F	**Command - F**	Open search and replace widget
Ctrl - G	**Command - G**	Go to specific line number
Ctrl - N	**Command - N**	Open new file dialog
Ctrl - O	**Command - O**	Open file open dialog
Ctrl - Shift - O	**Command - Shift - O**	Open file dialog to open the file(s) in the current (a new) project
Ctrl - P	**Command - P**	Open print dialog
Ctrl - S	**Command - S**	Save the current file. If the file does not exist on disk open the save file dialog
Ctrl - Alt - S	**Command - Option - S**	Open save as dialog
Ctrl - Shift - S	**Command - Shift - S**	Save all open files. Open save file dialog if needed

Shortcut	macOS	Description
Ctrl - K	**Command - K**	Open Navigator widget
Ctrl - T	**Command - T**	Open a terminal for the current project location. On macOS the terminal points to the users home directory

6.14.11.3 Studio Recovery

Shortcut	macOS	Description
Ctrl - F2	**Command - F2**	Reset Studio to default visual settings (window position and -size, widget position and -visibility) Useful when a widget got lost or studio started on a disconnected monitor.

6.14.11.4 Zoom Editor and View

All editors and views can be zoomed in and out. To avoid repeating action a zoom is applied to all editors and views of the same group. Beneath the Integrated Help and the Welcome Page these groups are defined:

- **Text**: contains all editors and views that show flowing text in monospace fonts and appear as Central Widget, like the Code Editor and the Listing Viewer.
- **Log**: contains the Process Logs and the System Log.
- **Table**: contains all editors and viewers that show table and tree elements in proportional fonts as a Central Widget, like the GDX Viewer and the Solver Option Editor. It additionally contains the Command Line Option Editor.

Action	Shortcut	macOS	Description
Zoom In	**Ctrl** - + or **Ctrl** - = or **Ctrl - Wheel Up**	**Command** - + or **Command** - = or **Command - Wheel Up**	Zoom **in** all editors and views of the same group
Zoom Out	**Ctrl** - **-** or **Ctrl - Wheel Down**	**Command** - **-** or **Command - Wheel Down**	Zoom **out** all editors and views of the same group
Reset Zoom	**Ctrl - 0**	**Command - 0**	Reset zoom of all editors and views

6.14.12 Usage Hints

6.14.12.1 Filter in Tables and Trees

To find relevant entries in tables and tree views Studio provides a special line edit containing up to four buttons aligned to the right:

- **Clear** - clears the term in the line edit.
- **Exact Match** - when active, the filter accepts only rows with one cell matching exactly the term regarding wildcard or regular expression setting

- **Regular Expression** - when inactive, the wildcards '∗' (any amount of characters) and '?' (one character) can be used. Otherwise the filter provides full regular expression syntax. Click here for further information about regular expressions.
- **All Columns** - when inactive, the filter checks only the key column. When active, all cells of the table/tree-view are taken into account

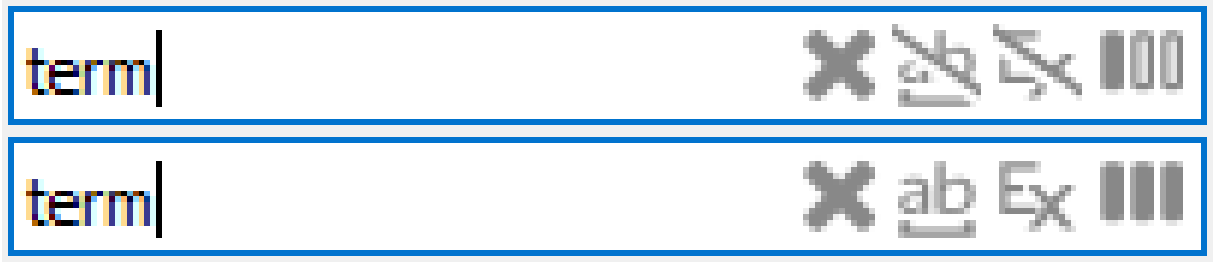


Figure 6.51 Filter LineEdit inactive and active

6.14.13 System Requirements

Compared to most of the GAMS system, GAMS Studio has some additional system requirements, which depend on the platform used. Also, additional information about technical requirements are listed at the GAMS Studio GitHub Wiki.

6.14.13.1 Linux

The Linux version of GAMS Studio is distributed as AppImage and requires glibc 2.30, glibcxx 3.4.28 as well as FUSE to be installed.

The GAMS Studio AppImage was tested and it was verified that it worked for the following Linux distributions (which are all under maintenance). All Linux distributions are under constant development and updates may change the behavior of the AppImage. Please check the Studio issue page if you encounter any issue.

- Debian 12
- Fedora 38
- Manjaro
- openSUSE Tumbleweed
- Ubuntu 22.04 LTS

Other flavors might work as well, but were not tested. If you experience problems with a particular Linux distribution, please let us know.

In some cases the Studio AppImage might not run. If this is the case extract the AppImage by running

```
$ ./studio.AppImage --appimage-extract
```

via the command line and start Studio from the extracted directory.

On some Linux distributions Wayland is used, which can cause issues. If this is the case try to run Studio as shown below.

```
$ QT_QPA_PLATFORM=xcb ./GAMS_Studio-1.12.1-x86_64.AppImage
```

6.14.13.2 macOS

macOS has some special platform requirements related to finding GAMS. Below the steps are listed which are used to link GAMS Studio to GAMS.

- GAMS Studio is going to find GAMS in `/Applications/**` and `/Users/username/Applications/**` if it is part of a GAMS folder. It is required that GAMS Studio gets copied via `Finder` to its target location. This is the case if the GAMS installer is used or if the GAMS Studio DMG is open and GAMS Studio gets copied from there.
- If GAMS Studio could not find GAMS like described previously then it will search `/Applications/**` for a valid GAMS installation, i.e. a GAMS installation which has the minimum required GAMS version or higher.
- GAMS Studio will check the `$PATH` to find GAMS. In almost all cases this will not work on macOS.
- In case of special needs the path to GAMS can be set explicitly by starting GAMS Studio with the command line argument `--gams-dir`.

6.14.13.3 Windows

There are dependencies on certain Visual C++ libraries. These are present on most Windows systems but are missing on some. If you get a complaint about missing libraries on startup of GAMS Studio, please run the appropriate installer for these libraries called `vcredist_x64.exe`, which can be found in the GAMS installation folder at `[GAMS system]\studio`.

6.14.14 Comparing GAMS Studio and GAMSIDE

As mentioned above, the classic GAMS IDE has been shipped with the GAMS system for many years and is still the workhorse for many GAMS programmers. So it is still around and an alternative to GAMS Studio, especially since it has features, which are not available in GAMS Studio (yet), but it also lacks some features which are available in GAMS Studio. The following table gives a compact overview about most of the differences.

GAMSIDE only	GAMS Studio only
Supported Platforms	
- Windows	- Windows
	- Linux
	- macOS
GDX Viewer	
- Creating charts	- Filtering (both, symbols and data, labels and values)
- Writing to Html	- Sorting by numerical value
- Heat-map functionality (aka Data Colors)	- Option to change display format of numerical values
Editor	
- Fixed encoding (ISO 8859-1)	- Allow change of encodings and use UTF-8 by default
- Spell checker	- Block edit
- Recognize URLs and open them in the browser	- Help integration (pressing `F1` on a keyword opens the relevant index page of the documentation)
- Different set of keyboard shortcuts	- Tooltips for compilation errors

GAMSIDE only	GAMS Studio only
	- Navigation history
	- Code Completion
	- "Smart typing": Automatic insertion of closing character for brackets and quotes
	- Distraction Free Mode
	- Code Folding
	- Open `$[bat/lib]include` files directly from editor
Execution	
- IDE Projects	- Studio Projects (see below for details)
- Set default options for the IDE only	- Interactive editor for the GAMS User Configuration
- Use different GAMS versions	- Interactive editor of GAMS command line parameters
- Use alternative license	- Interactive editor for GAMS Connect
- Execute program based on extension	- Option to run with GDX creation automatically (`F10` vs. `F9`)
- Solver selection menus	- Integration of GAMS MIRO
	- Remote execution on NEOS Server for Optimization
	- Remote execution with GAMS Engine
	- Support of standard locations, e.g., to detect the license file at a standard path
	- Option to jump to first error automatically
	- Option to not write log to disk
Other	
- Text Diff	- Welcome page
- Script recording	- Project Explorer
- Email file	- Navigator
- Save test files in UNIX format	- Filtering in Model Library Explorer (all model libraries at the same time)
- Link to McCarl chm	- Allow to switch between light, dark or custom themes
	- Reference File Viewer allows to jump to all references etc. and not just the first one
- 32-bit application	- 64-bit application

6.14.14.1 GAMSIDE Projects vs. GAMS Studio Projects

With the GAMSIDE one always started with creating a "Project". This mainly defined a directory used to execute the models added to it (so, it is used to implicitly define the workDir when starting GAMS). When starting the IDE for the first time, one is asked to create a new project. For example, one creates this in directory `a`. After adding models `a/a.gms` and `b/b.gms` both get executed in the project directory `a`, meaning that data sources are expected in `a` (or relative to it) and also output files are generated in `a`, and not necessarily at the location of the `gms` file.

When starting GAMS Studio for the first time, it starts with an empty Project Explorer. Opening a new file creates a new Project per default. Studio Projects also have a "working directory". Initially, this is set automatically to the location of the file that was opened to create the new project. If a model gets executed, it gets executed in this working directory. So, if ones opens `a/a.gms` and `b/b.gms`, by default,

Studio will create two projects: `a` and `b`. When `a` is started input files are expected in directory `a` and output files get generated in `a`. When `b` is started input files are expected in directory `b` and output files get generated in `b`. If one has structured the models so that they rely on the "GAMSIDE way" to execute a model, one can just change the working directory in the project options. Also, it is possible to start with an empty project and just define the working directory explicitly in the way the GAMSIDE did it using the entry `New Project` from the `File` menu.

It is also possible to have multiple files in one project. For example, one could add `b.gms` to project `a` by drag'n'drop in the Project Explorer. Now, the concept of the "main file" of projects becomes important: While the GAMSIDE always executed the currently active `.gms` file, GAMS Studio always executes the main file of the currently active project. So, in our example, you can look at `a.lst` and press `F9` to rerun `a.gms` (the GAMSIDE would not run anything in this case). After adding `b.gms` to project `a`, one would also execute `a.gms` by pressing `F9` while looking at `b.gms`. If you want to execute `b.gms` instead, you can make it the main file of the project. For this, right-click in the Project Explorer at `b.gms` and select "Set as main file". Note that the green arrow indicating the main file of a project in the Project Explorer switches from `a.gms` to `b.gms`. As a reminder: Projects do not know a particular working directory. Pressing `F9` after switching the main file will execute `b.gms` in folder `b`.

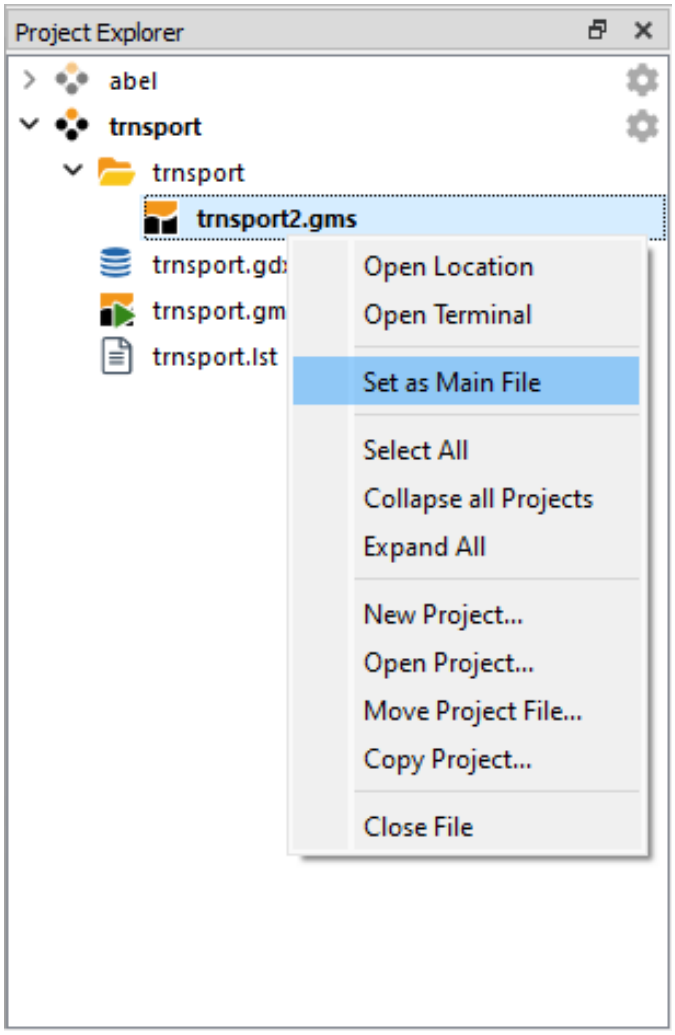


Figure 6.52 Switching the main file

6.15 GDX2ACCESS

6.15.1 Overview

GDX2ACCESS is a tool to dump the contents of a GDX file to an MS Access file (MDB or ACCDB format). Every identifier gets its own table in the database.

Attention

GDX2ACCESS is deprecated (see GAMS 40 GDX2ACCESS release notes). Please use Connect agent SQLWriter instead.

6.15.2 Usage

```
gdx2access inputFile {options}
```

The .gdx file extension of the inputFile can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

6.15.3 Options

Options are specified in an [INI] (https://en.wikipedia.org/wiki/INI_file) file and not directly on the command line. By default, the file gdx2access.ini located in the same directory as gdx2access.exe is consulted. If this file is not available, the program will continue using default settings (listed in the tables below).

It is also possible to tell the program to use a different INI file. This is done by using an extra argument of the form `@iniFile`. If you want to dump the contents of myFile.gdx to an MS Access file according to the options specified in a file called myIniFile.ini, run the following code:

```
gdx2access myFile.gdx @myIniFile.ini
```

In this case, the program will not read gdx2access.ini located in the same directory as gdx2access.exe, but rather myIniFile.ini in the current directory.

The INI file can contain two sections: [settings] and [debug]. A complete INI file with all possible settings looks like:

```
[settings]
scrdir=c:\tmp
inf=1.0e100
mininf=-1.0e100
eps=0.0
na=0.0
undf=0.0
scalarTable=1
etFlag=1
dbVersion=9
[debug]
method=5
thresholdCount=5
keepFiles=1
```

Note that the values above are not the default values for some options!

Section [settings]

Below some short descriptions for the options in the [settings] section:

Option	Default	Description
scrdir	<windowstemp>	Directory for temporary scratch files.
inf	1.0e100	The value used for GAMS special value `+INF`. See also the example Special value mapping.
mininf	-1.0e100	The value used for GAMS special value `-INF`. See also the example Special value mapping.
eps	0.0	The value used for GAMS special value `EPS`. See also the example Special value mapping.
na	0.0	The value used for GAMS special value `NA`. See also the example Special value mapping.
undf	0.0	The value used for GAMS special value `UNDEF`. See also the example Special value mapping.
scalarTable	0	Possible values: 0 or 1. When set to 1, scalars of the same type are combined in a single table. The names for these tables are fixed: `ScalarParameter`, `ScalarEquation` and `ScalarVariable`. See also the example Special value mapping.
etFlag	0	Possible values: 0 or 1. When set to 1, include the text strings for sets containing element texts. See also the example Writing Explanatory Text to Database.
dbVersion	0	Specify the format of the output database. See also the example Writing Explanatory Text to Database. 0: Create database using current default format. 9: Create a database in the Microsoft Access 2000 (.mdb) file format. 10: Create a database in the Microsoft Access 2002-2003 (.mdb) file format. 12: Create a database in the Microsoft Office Access 2007 (.accdb) file format.

Section [debug]

Below some short descriptions for the options in the [debug] section:

Option	Default	Description
method	5	Select an algorithm to insert data into the Access database. 1: Write a CSV file and use the TransferText action to read this into Access. This is fast, but does not always work when non-US language settings are used. 2: Use recordset.add to add records. This is slow, but does not use intermediate files. 3: Write a tab delimited file with a complete file specification (schema.ini) and use the ISAM Text driver to import the data. This is fast and should work in international settings. 4: For small data use method=2 and for larger data items use method=1. 5: For small data use method=2 and for larger data items use method=3.
thresholdCount	5	When to change between algorithms while using method=4 or method=5. The default is 5 records.
keepFiles	0	Possible values: 0 or 1. When set to 1, the program will not delete intermediate scratch files.

6.15.4 Examples

6.15.4.1 Intro

Suppose we want to write the data of the [**trnsport**] model from the GAMS model library after solving to an MS Access data file. First of all, we solve the model by running the following command. Note that we will save the complete symbol table to a GDX file called trnsport.gdx by adding the command line option `gdx=trnsport`.

```
gams trnsport gdx=trnsport lo=2
```

The option `lo=2` causes GAMS to save the log to an external file (in this case trnsport.log), instead of writing it to the screen. In order to get an idea about the data at this point, we use the GDXDUMP tool to display the contents of the GDX file by running the following command:

```
gdxdump trnsport.gdx symbols
```

By adding the `GDXDUMP` option `symbols`, we will only display the table of contents (shown below) rather than all data stored in trnsport.gdx.

```
*  GDX dump of trnsport.gdx
*  Library in use : C:\GAMS23.3
*  Library version: GDX Library      Nov  1, 2009 23.3.3 WIN 14596.15043 VIS x86/MS Windows
*  File version   : GDX Library      Nov  1, 2009 23.3.3 WIN 14596.15043 VIS x86/MS Windows
*  Producer       : GAMS Base Module Nov  1, 2009 23.3.3 WIN 14929.15043 VIS x86/MS Windows
*  File format    :    7
*  Compression    :    0
*  Symbols        :   12
*  Unique Elements:    5
   Symbol Dim Type  Explanatory text
```

```
 1 a        1  Par  capacity of plant i in cases
 2 b        1  Par  demand at market j in cases
 3 c        2  Par  transport cost in thousands of dollars per case
 4 cost     0  Equ  define objective function
 5 d        2  Par  distance in thousands of miles
 6 demand   1  Equ  satisfy demand at market j
 7 f        0  Par  freight in dollars per case per thousand miles
 8 i        1  Set  canning plants
 9 j        1  Set  markets
10 supply   1  Equ  observe supply limit at plant i
11 x        2  Var  shipment quantities in cases
12 z        0  Var  total transportation costs in thousands of dollars
```

Once we have a GDX file, we can use `GDX2ACCESS` to create a MDB or an ACCDB file. Versions of MicroSoft Office prior to version 2007 use the file extension .mdb, while version 2007 and later versions use the file extension .accdb. We write all the data stored in the GDX file trnsport.gdx to trnsport.mdb resp. trnsport.accdb by running the following code:

```
gdx2access trnsport.gdx
```

Note that we do not referenced an INI file in the previous command in order to specify the options, i.e. `GDX2ACCESS` will consult the file gdx2access.ini located in the same directory as gdx2access.exe or, if the file does not exist, the tool will continue using the default settings (see Options). The information written to the log is shown below, including the identifiers of the symbols which are written to trnsport.accdb and the elapsed time per symbol and in total.

```
GDX Access       ALFA 23Mar10 23.4.0 WIN 16693.16738 VS8 x86/MS Windows
Creating C:\GAMS support\settext\temp1.accdb with Access: 0.48 seconds
Using temp directory C:\Users\Paul\AppData\Local\Temp\
           i.   Insert:  0.00 seconds
           j.   Insert:  0.00 seconds
           a.   Insert:  0.00 seconds
           b.   Insert:  0.00 seconds
           d.   Dump:  0.00 seconds  Load:  0.05 seconds
           f.   Insert:  0.00 seconds
           c.   Dump:  0.00 seconds  Load:  0.05 seconds
           x.   Dump:  0.00 seconds  Load:  0.03 seconds
           z.   Insert:  0.00 seconds
        cost.   Insert:  0.00 seconds
      supply.   Insert:  0.02 seconds
      demand.   Insert:  0.02 seconds
Renaming  C:\GAMS support\settext\temp1.accdb -> trnsport.accdb
Total elapsed time: 0.92 seconds
```

The resulting ACCBD, opened with MS Access, is shown in Figure 1 (limited to the parameter `c` on the left and the variable `x` on the right).

As can be seen, every identifier is stored in its own table. For parameters like `c`, the value is stored in a column named value, while variables like `x` and equations have the columns level, marginal, lowerbound and upperbound. A possible additional field (scale for NLP's, priority for MIP's, stage for stochastic problems) is not exported. If needed, you can assign such a quantity to a parameter in GAMS before writing the GDX file. The complete process shown here can be automated as demonstrated in Example 1.

If no domain information is available for a symbol (like `B` in the code below), each index position gets a column and will be labeled automatically with dim1, dim2, etc. If domain information is available, the columns will use that information but keeping the names unique (like `A` in the code below). The small example below shows how the index positions are made unique using the (relaxed) domain information by adding an ascending number to the identifier.

```
Set i / i1*i5 /;
Alias (i,j,k);

Parameter A(i,i,i) 'domain informations, but the column name would not be unique';
A(i,j,k) = uniform(0,1);

Parameter B 'no domain information' / i1.i1 1, i1.i2 2, i2.i1 3, i2.i2 4 /;

execute_unload 'AB.gdx', A, B;
execute '=gdx2access AB.gdx';
```

The resulting database file, opened with MicroSoft Access, is shown in Figure 2 (parameter `A` on the left, parameter `B` on the right).

6.15.4.2 Example 1 - Dumping the Contents of trnsport.gdx

This example will solve the [**trnsport**] model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Access and MS Access is launched to inspect the results. This is a small example that should run very quickly.

```
execute '=gamslib trnsport';
execute '=gams trnsport lo=3 gdx=trnsport';
execute '=gdx2access trnsport.gdx';
executeTool 'win32.shellExecute trnsport.accdb';
```

Note: the equal signs in front of the external programs indicate we don't go through a shell (e.g. command.com or cmd.exe). This will improve reliability in case the external program is not found. In such a case a proper error will be triggered. Without the '=', such errors go undetected and the GAMS model will continue.

The command 'executeTool 'win32.shellExecute trnsport.accdb';` will launch Access to view the ACCDB file. This assumes that the version of Access installed is a version as recent as version 2007. Earlier versions will generate a database with the extension .mdb and the shellExecute command needs to be changed as follows:

```
* view generated file create
executeTool 'win32.shellExecute trnsport.mdb';
```

The complete example is also part of the GAMS Data Utilities Library, see model [**GDX2ACCESSExample1**] for reference.

6.15.4.3 Example 2 - Writing Explanatory Text to Database

In this example, we write a few sets to a GDX file; two of the sets written have explanatory text for set elements. We use the option etFlag to get this text saved in the Access database file along with the corresponding set elements. Without using the option, only the set tuples are saved in the database.

Running this example on a machine with Access 2007 or a later version installed will create a database with the .accdb file extension that cannot be read by an older version of Access. We use the dbVersion option to save the database in the MDB format. Both options must be specified in an user defined INI file within the settings section, see howToWrite.ini in the code below.

```
Set
   i       / i1 'one', i2 'two', i3 'three', i4 'four' /
   j       / j1*j4 /
   ij(i,j) / i1.j1 'red', i2.(j2,j3) 'green', i3.(j1,j2) 'blue' /;

$gdxOut example2.gdx
$unload i j ij
$gdxOut

$onEcho > howToWrite.ini
[settings]
etFlag=1
dbVersion=10
$offEcho

$call =gdx2access example2.gdx @howToWrite.ini%
```

When we open the database and inspect the table created from the set `ij`, we see the explanatory texts stored along with the corresponding set elements. The column containing the explanatory text will be labeled with `SetText` automatically. The column headers can be renamed as demonstrated in Example 5 - Renaming Fields.

The complete example is also part of the GAMS Data Utilities Library, see model **[GDX2ACCESSExample2]** for reference.

6.15.4.4 Example 3 - Dumping a large Table to Database

This is an artificial example where we generate a large identifier in GAMS: a parameter with a million elements. This parameter will be dumped to an MS Access Database afterwards. Note that the `GDX2ACCESS` execution will last several seconds. The resulting database file has approximately 36MB of size.

```
Set i / i1*i1000 /;
Alias (i,j);
Parameter p(i,j);
p(i,j) = uniform(-100,100);
execute_unload 'example3.gdx', p;
execute '=gdx2access example3.gdx';
```

The complete example is also part of the GAMS Data Utilities Library, see model **[GDX2ACCESSExample3]** for reference.

6.15.4.5 Example 4 - Special Value Mapping

To store special values like +INF, -INF, EPS, NA and UNDF in a numeric field in the database, GDX2ACCESS uses a mapping. This mapping can be changed by using an INI file (for the default values, see Options resp. the comments in the second code below). We will define a scalar for each special value in GAMS in order to demonstrate the scalarTable option in addition. By default, every scalar will be written to a new table. By activating the scalarTable option in the INI file, all scalars will be stored together in a single table.

```
$onEcho > howToWrite.ini
[settings]
inf=1
mininf=2
eps=3
na=4
undf=5
scalarTable=1
$offEcho
```

The usage of our previously customized INI file howToWrite.ini is indicated by the argument @howToWrite.ini within the execute statement.

```
$onUndf
Scalar
   pInf      /  inf /
   mInf      / -inf /
   epsilon   /  eps /
   notAvail  /   na /
   undefined / undf /;

* save scalars in scalars.mdb in a single table named ScalarParameter using
* the scalarTable option
* special values are translated to default values:
*  INF ->  1.0e100
* -INF -> -1.0e100
* EPS,NA,UNDF -> 0

execute_unload 'scalars.gdx', pInf, mInf, epsilon, notAvail, undefined;
execute 'gdx2access scalars.gdx @howToWrite.ini';
```

Note the $onUndf command in order to enter Undf values directly within the definition of the scalars. When we view the generated GDX file in the GAMS IDE or GAMS Studio, the special values are shown (note that the figure shows a parameter actually instead of five single scalars for a more compact presentation):

Viewing the resulting table in Access shows how the mapping for special values was applied (i.e. the GAMS special values have been substituted by our customized values defined in the INI file):

Note that the table will be named ScalarParameter automatically.

The complete example is also part of the GAMS Data Utilities Library, see model **[GDX2ACCESSExample4]** for reference.

6.15.4.6 Example 5 - Renaming Fields

GDX2ACCES will use names like i, j, dim1, dim2, value etc. for the column headers in the resulting database file. In some cases, this may not be convenient, e.g. when more descriptive field names are required. In the following model, we will show how a small script in VBscript[1] can handle this task. The script will rename the columns i, j, and Value in table c to ifrom, jto, and transportcost. At first, the data is defined and dumped to an MicroSoft Access database. The VBscript named access.vbs is written at compile time and later executed at execution time in order to rename the column headers.

```
Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

Parameter
   a(i) 'capacity of plant i in cases'
        / seattle    350
          san-diego  600 /

   b(j) 'demand at market j in cases'
        / new-york   325
          chicago    300
          topeka     275 /;

Table d(i,j) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

* export to gdx file.
execute_unload 'c.gdx', c;

* move to access database
* column names are i and j
execute '=gdx2access c.gdx';

* rename columns
execute '=cscript access.vbs';

* view results
executeTool 'win32.shellExecute c.accdb';

$onEcho > access.vbs
'this is a VBscript script
WScript.Echo "Running script: access.vbs"
set oa = CreateObject("Access.Application")
set oDAO = oa.DBEngine
Wscript.Echo "DAO Version: " & oDAO.version
Set oDB = oDAO.openDatabase("%system.fp%c.accdb")
Wscript.Echo "Opened : " & oDB.name
Set oTable = oDB.TableDefs.Item("c")
Wscript.Echo "Table : " & oTable.name
' rename fields
```

```
oTable.Fields.Item("i").name = "ifrom"
oTable.Fields.Item("j").name = "jto"
oTable.Fields.Item("Value").name = "transportcost"
Wscript.Echo "Renamed fields"
oDB.Close
Wscript.Echo "Done"
$offEcho
```

The resulting ACCBD, opened with MS Access, is shown in Figure 6 (parameter `c` before executing the VBScript on the left, parameter `c` after renaming the column headers on the right).

The complete example is also part of the GAMS Data Utilities Library, see model **[GDX2ACCESSExample5]** for reference.

6.15.5 References

1. VBScript Language Reference, https://www.vbsedit.com/html/ddfa5183-d458-41bc-a489-070296ced968.asp 2023

6.16 GDX2SQLITE

A Tool to dump GDX contents into SQLite database file.

Author

Erwin Kalvelagen

Date

November 30, 2015

Attention

`GDX2SQLITE` is deprecated (see GAMS 40 GDX2SQLITE release notes). Please use Connect agent SQLWriter instead.

6.16.1 Introduction

`GDX2SQLITE.EXE` is a tool to dump the complete contents of a GAMS GDX file (see GAMS Data eXchange (GDX)) into a SQLite database file (the website http://www.sqlite.org/ contains a wealth of information on SQLite).

A SQLite database is stored in a single file so it can be easily e-mailed or otherwise transmitted. The main advantages of using SQLite over other single file database systems such as MS Access is that SQLite is free and in the public domain and that it does not impose a 2 GB file size limit. For some large data sets this size limit present in MS Access and DBF database files causes problems. Another useful format is CSV files, but typically several CSV files are needed to store a data set stored in a GDX file. Many programs support reading SQLite database files, either through a native database access driver or via a standard ODBC interface. In summary: SQLite is a useful export format for GAMS solution data sets.

6.16.2 Usage

'GDX2SQLITE' is a command line tool that is best called from within a GAMS program using the `$call` or `Execute` statement, e.g.:

```
execute_unload "results.gdx", yield, price;
execute "gdx2sqlite -i results.gdx -o results.db";
```

The following options are available:

Options	Description
`-i gdxinputfile`	Specifies the input GDX file. Typically this is a file with a .gdx extension
`-o sqloutputfile`	Specifies the output SQLite database. Typically this file has a .db extension.
`-debug`	This is an optional flag that will cause gdx2sqlite to print additional debugging information.
`-expltext`	This optional flag will export explanatory text for set elements.
`-append`	Don't delete the database file before processing. This will allow adding new symbols in new tables. We will not allow adding data to existing tables.
`-small`	Write data strings in a separate table. A user-friendly SQL VIEW is created to hide the complexities of the joins.
`-fast`	Try to speed up writing the data using some non-standard pragmas. Using both `-small -fast` will write the data most efficiently.
`-varchar`	String columns will have the type `VARCHAR(255)` instead of `TEXT`.

An example of explanatory text is:

```
Set cty / AFG  'Afghanistan'
          AGO  'Angola'
          ALB  'Albania'     /;
```

In GAMS set elements have a maximum length of 63 characters. Explanatory text has a maximum length of 255 characters.

6.16.3 How data is stored

6.16.3.1 Gams issues

GAMS does not store zero values (or default records for variables and equations). Such non-existing records will not be exported to the GDX file and to the database either. To force a zero to be exported, set it to `EPS` in GAMS. E.g.:

```
p(i)$(p(i)=0) = EPS;
```

In case of doubt you are encouraged to inspect the GDX file.

6.16.3.2 Sets

n-dimensional sets are stored as tables with n text columns. In case the option `-expltext` is used, another column may be added with explanatory text.

GAMS	SQLite
Set month / jan, feb, mar / year / 2013, 2014 / date(year,month) /(2013,2014).(jan,feb,mar) /; execute_unload "sets.gdx"; execute "gdx2sqlite -i sets.gdx -o sets.db";	sqlite>select * from month; month ---------- jan feb mar sqlite>select * from year; year ---------- 2013 2014 sqlite>select * from date; year month ---------- ---------- 2013 jan 2013 feb 2013 mar 2014 jan 2014 feb 2014 mar
Set month / jan 'january' feb 'fabruary' mar 'march' /; execute_unload "sets.gdx"; execute "gdx2sqlite -i sets.gdx -o sets.db -expltext";	sqlite>select * from month; month expltext ---------- ---------- jan january feb fabruary mar march

6.16.3.3 Parameters

n-dimensional parameters will have n index columns plus a value column. Scalars are collected in a separate table.

GAMS	SQLite
Set i / i1*i4 /; Parameter p(i); p(i) = uniform(0,1); Scalar s1 / 10 / s2 / 20 /; execute_unload "data.gdx"; execute "gdx2sqlite -i data.gdx -o data.db";	sqlite>select * from p; i value ---------- ---------- i1 0.171747132 i2 0.843266708 i3 0.550375356 i4 0.301137904 sqlite>select * from scalars; name value ---------- ---------- s1 10.0 s2 20.0

6.16.3.4 Variables and Equations

n-dimensional variables and equations have besides n index columns also columns for the level, the lower and upper-bound and the marginal. Scalars are collected in the tables scalarvariables and scalarequations. Note that INF and -INF are mapped to 1.0e100 and -1.0e100. The special value EPS is exported as zero. To be complete: UNDEF, NA and acronyms are exported as NULLs.

GAMS	SQLite
Set i / i1*i4 /; Positive Variable x(i); x.l(i) = uniform(0,1); Variable z; z.m = 1; execute_unload "data.gdx"; execute "gdx2sqlite -i data.gdx -o data.db";	sqlite>select * from x; i level lo up marginal ---------- ---------- ---------- ---------- ---------- i1 0.171747132 0.0 1.0e+100 0.0 i2 0.843266708 0.0 1.0e+100 0.0 i3 0.550375356 0.0 1.0e+100 0.0 i4 0.301137904 0.0 1.0e+100 0.0 sqlite>select * from scalarvariables; name level lo up marginal ---------- ---------- ---------- ---------- ---------- z 0.0 -1.0e+100 1.0e+100 1.0

6.16.3.5 Fixing up names

A database table is not allowed to have columns with the same name. If a name clash is detected new names may be invented.

GAMS	SQLite
`Set i / i1*i4 /; Parameter p(i,i); p(i,i) = 1;` `execute_unload "data.gdx"; execute "gdx2sqlite -i data.gdx -o data.db";`	`sqlite>select * from p; i i2 value ---------- ---------- ---------- i1 i1 1.0 i2 i2 1.0 i3 i3 1.0 i4 i4 1.0`

6.16.3.6 Speeding up writing data

With the `-small` option we write data in a slightly different format. Instead of using strings for the GAMS indices we write integers. The integers can be looked up in a separate table `UEL$` were the GAMS UELS (Unique Elements) are stored. We export an SQL view for each symbol to hide the complexities of the joins needed to replace the integers by strings.

For more information see: http://yetanothermathprogrammingconsultant.blogspot.com/2014/06/big-data-cube

The `-fast` option will set some SQLite pragmas that can speed up the inserts. Basically they will give up some consistency in case the program crashes, in which case the created database may be invalid.

For more information see: http://yetanothermathprogrammingconsultant.blogspot.com/2014/07/a-little-bit-

6.16.4 SQLite Browsers and compatible software

6.16.4.1 SQLite3.exe

From the distribution on http://www.sqlite.org/download.html a command line tool is available that functions as shell for SQLite. An example session can look like:

```
C:\projects\impact3\sqlite>sqlite3.exe data.db
SQLite version 3.8.0.2 2013-09-03 17:11:13
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE [i]([i] TEXT);
CREATE TABLE [p]([i] TEXT,[i2] TEXT,[value] REAL);
sqlite> select * from p;
i1|i1|1.0
i2|i2|1.0
i3|i3|1.0
i4|i4|1.0
sqlite> .quit

C:\projects\impact3\sqlite>
```

6.16.4.2 SQLite Studio

A visual front-end can be downloaded from http://sqlitestudio.pl/.

6.16.4.3 SQLite Database Browser

Another visual browser is available from http://sqlitebrowser.sourceforge.net/.

6.16.4.4 SQLite and R

The statistical software R can conveniently use SQLite data, and can be called from a GAMS environment as follows:

```
$onText
Get data from GAMS into R via SQLite
$offText

$set IMPACTPATH     c:\projects\impact3\impact_3\IMPACTv3.0
$set SCRIPT         script.R
$set RPATH          "C:\Program Files\R\R-3.0.2\bin\R.exe"
$set DB             mapdata.db

Set cty;
$gdxIn %IMPACTPATH%\GDXs\Sets.gdx
$load cty

display cty;

Set maps / data1 'uniform random data between 0 and 1'
           data2 'uniform random data between 0 and 2' /;

Parameter mapdata(cty,maps);
mapdata(cty,"data1") = uniform(0,1);
mapdata(cty,"data2") = uniform(0,2);

execute_unload "mapdata.gdx";
execute "gdx2sqlite -i mapdata.gdx -o %DB% -expltext";
execute '"%RPATH%" --vanilla < %SCRIPT%';

$onEcho > %SCRIPT%
if (!require(RSQLite)) {
   install.packages("RSQLite", repos="http://cran.r-project.org")
   library(RSQLite)
}

sqlite<-dbDriver("SQLite")
db <- dbConnect(sqlite,"%DB%")
dbListTables(db)

maps<-dbGetQuery(db,"select * from maps")
maps

mapdata<-dbGetQuery(db,"select * from mapdata")
mapdata
$offEcho
```

6.16.4.5 SQLite and Python

Python has built-in support for SQLite:

```
import sqlite3

db = sqlite3.connect("turkey.db")

c = db.execute("select * from yieldl")
for row in c:
     print row

db.close()
```

6.16.4.6 SQLite ODBC Driver

ODBC is a database access layer for Windows. It allows many Windows programs that need to talk to databases to do this in a database independent manner. The SQLite ODBC driver can be downloaded from: http://www.ch-werner.de/sqliteodbc/.

6.16.4.7 SQLite and Excel

Excel can read SQLite database files through ODBC.

Import as Table

To import a table from a SQLite database perform the following steps:

1. Select the Data tab and choose Get External Data From Other Sources
2. Choose the Data Connection Wizard
3. Choose ODBC DSN
4. Select SQLite3 Datasource
5. Enter the name and path of the database file and press OK.
6. Choose a table from the database.
7. Give this selection a name.
8. Choose Import as Table

The result is a table:

Import as Pivot Table

The same steps can be used to import as Pivot Table. This way we can easily create summary reports, such as:

6.17 GDX2VEDA

Translates a GDX file into the VEDA format.

6.17.1 Usage

```
gdx2veda gdx vdd [run]
```

gdx

GAMS GDX file

vdd

VEDA Data Definition file

run

VEDA Run identifier (optional)

The VEDA data file name and run identifier are either taken from the gdx file name or specified with the run name. Use "token with blanks" if needed.

6.17.2 Examples

This example dumps the gdx symbols:

```
gdx2veda mygdx
```

This exmaple prints usage and example:

```
gdx2veda
```

To print more detailed help message:

```
gdx2veda --help
```

Add .csv to the run name to write in csv format

6.17.3 Detailed Help Message

6.17.3.1 VDD file Summary

```
[DataBaseName]
 myveda

[Dimensions] cube dimensions
 long_name tuple_element1 tuple_element2 ...

[DataEntries] data for the cube
 long_name gams_name tuple_element1 tuple_element2 ...

[DimensionText] for generating .vde file (only for data in [DataEntries])
 gams_set tuple_element1 tuple_element2 ...

[DimensionTextAll] for generating .vde file (also for data not in [DataEntries])
 gams_set tuple_element1 tuple_element2 ...

[SubSets] for generating .vds file
 sub_name gams_name tuple_element1 tuple_element2 ...

[ParentDimension] defines parent-child structure
 parent_tab  child_tab1 child_tab2 ...

[ParentDimensionTextAll] .vde file definitions with parent-child structure
2d_gams_set  parent_tab child_tab
2d_gams_set  child_tab parent_tab

[ParentSubSets] .vds file definitions with parent-child structure
sub_name 2d_gams_set parent_tab child_tab
sub_name 2d_gams_set child_tab parent_tab

[Options]
TupleSeparator "string"  use a different separator symbol between tuple elements
ShowAllSeparators        don't squeeze unnecessary separators
RelaxDimensionAll        relax strict dimensionality checks in DimensionText(All) sections.
ValueDim n               if n=2 write PV/DV value pairs for VEDA
SetsAllowed dim1 dim2 .. write SetsAllowed specification line to VEDA .vd file
Scenario scenarioSet     specify the scenario set; a record with expl text goes to .vde
Format veda/csv          specify the format of the data files
Not-0 attribute ...      don't write records with zero values for these attributes

[SpecialValues]
EPS "string"             value to be used for EPS
INF "string"             value to be used for +INF
MINF "string"            value to be used for -INF
NA "string"              value to be used for NA
UNDEF "string"           value to be used for UNDEF
```

<myveda>

is usually the application name which will be displayed on the top of the VEDA splash screen. When a new VEDA database is created, a new folder with this name will appear:

`...veda\database\mayveda_date_time.`

Where data and time are the creation time stamp.

<tab_name>

corresponds to the tabs of your VEDA screen

Lines starting with ∗ and empty lines are ignored. Blanks, commas and tabs are delimiters, blanks before and after delimiters are ignored. Quotes around data items are optional. The input data is NOT case sensitive.

6.17.3.2 Veda Data Definition file

Example of a Veda Data Definition file:

```
* Transport model

[DataBaseName]
myveda

[Dimensions]
* tab-name indices
Plants     i
Warehouses j
Links      ii jj

[DataEntries]
* veda_attribute gams_name tab1 tab2 ... for gams index 1, 2, ...
"x(i,j) duals" x.m      i  Warehouses
Shipments      x.l      i  j
SupplyPrice    supply.m i
DemandPrice    demand.m j
TransportCost  c        i  j
Distance       d        ii jj
Supply         a        i
Demand         b        j
TotalCost      z.l
SupplyNodes    i        i
DemandNodes    j        j
Rate           f

[DimensionText]
* gams_set tab
i  i

[DimensionTextAll]
* gams_set tab
j  j

[SubSets]
* sub_name gams_name tab
i1 ic  Plants
i1 id  i
```

Notes:

The long name from the `[Dimensions]` section can be used as a macro that expands to the tuples it defines. E.g. "Links" is identical to "ii jj".

In the `[DataEntries]` section a literal tuple element can be defined as `/element/`.

When `ValueDim=2`, the `[DataEntries]` section can contain `X.LM` entries, indicating both `.L` and `.M` needs to written as a pair.

6.18 GDX2XLS

A Tool to convert GDX data to MS Excel spreadsheets.

Author

Erwin Kalvelagen

Date

June 10, 2005; December 6, 2006

This document describes the GDX2XLS utility which allows to convert data stored in a GDX file into Excel spreadsheets.

6.18.1 Overview

Attention

GDX2XLS is deprecated (see GAMS 42 GDX2XLS release notes). Please use Connect agent ExcelWriter instead.

GDX2XLS is a tool to dump the complete contents of a GDX file to an MS Excel spreadsheet file (.xlsx or .xls file). Every identifier gets its own sheet in the .XLSX file. Excel 2007 or more recent versions will default to an xlsx file; versions before that default to the .xls file format. For instance when we save the results of the trnsport model from the model library:

```
C:\tmp>gamslib trnsport
Model trnsport.gms retrieved

C:\tmp>gams trnsport gdx=trnsport lo=2

C:\tmp>gdxdump trnsport.gdx symbols
   Symbol Dim Type Records  Explanatory text
 1 a        1  Par       2  capacity of plant i in cases
 2 b        1  Par       3  demand at market j in cases
 3 c        2  Par       6  transport cost in thousands of dollars per case
 4 cost     0  Equ       1  define objective function
 5 d        2  Par       6  distance in thousands of miles
 6 demand   1  Equ       3  satisfy demand at market j
 7 f        0  Par       1  freight in dollars per case per thousand miles
 8 i        1  Set       2  canning plants
 9 j        1  Set       3  markets
10 supply   1  Equ       2  observe supply limit at plant i
11 x        2  Var       6  shipment quantities in cases
12 z        0  Var       1  total transportation costs in thousands of dollars

C:\tmp>
```

The example shows how we copy the trnsport.gms model from the model library, and then solve it. The option gdx=filename will save the complete symbol table to a GDX file. The option lo=2 tells GAMS to save the log to a file (in this case trnsport.log) instead of writing it to the screen. The gdxdump will display the contents of the GDX file (the option symbols will only display the table of contents, rather than all data)

Once we have a GDX file we can use GDX2XLS to create an .XLSX file:

```
C:\tmp>gdx2xls trnsport.gdx
GDX2XLS          34.1.0 r644dbd9 Released Jan 29, 2021 WEI x86 64bit/MS Window
Output file: C:\tmp\trnsport.xlsx
C:\tmp>
```

The resulting XLSX file, opened with MS Excel is shown in Figure 1. The first page is the Table of Contents page with all identifiers sorted alphabetically. When clicking on variable `x`, the sheet shown in Figure 2 is displayed.

The table of contents can be reached again by clicking on the `TOC` link in the top left corner. The complete process shown here can be automated as is shown in section Model gdx2xls1: import trnsport.gdx. As can be seen, every identifier is stored in its own sheet. Index positions get a column with labels showing their domains, or `dim1`, `dim2`, etc. if domain information is not available. By default scalar quantities are collected in a single sheet called `scalar`.

6.18.2 AutoFilter

By default the exported tables are organized in `AutoFilter` tables. This will allow you to easily make selections and sort the results.

It is possible to set filters for different columns. Only the rows that meet the criteria will be shown. The columns used in the filter can be recognized by having a blue arrow instead of a black one in the drop down menu header.

Sorting can also be performed on multiple columns: e.g. first sort on one column, then sort on a second column.

The autofilter generation can be turned off using an option in the .ini file.

6.18.3 Options

6.18.3.1 Default ini file

Options are specified in an .INI file. By default, the file `gdx2xls.ini` located in the same directory as `gdx2xls.exe` is consulted. If this file is not available, the program will continue using default settings.

6.18.3.2 Custom ini file

It is also possible to tell the program to use a different .ini file. This is done by using an extra argument of the form `@inifile`. An example would be:

```
C:\TMP> gdx2xls myfile.gdx @myinifile.ini
```

In this case the program will not read `gdx2xls.ini` located in the same directory as `gdx2xls.exe` but rather `myinifile.ini` in the current directory.

The ini file can contain two sections: `[settings]` and `[colors]`. A complete ini file with all possible settings looks like:

```
[settings]
inf=INF
mininf=-INF
eps=EPS
na=NA
undf=INDF
scalarsheet=1
tableformatting=1
toc=1
sorttoc=1
autofilter=1
freezeheader=1
indexformat=
valueformat=

[colors]
header=17
body=19
italics=48

[xmlcolors]
link=#0000FF
header=#9999FF
body=#FFFFCC
italics=#969696
```

6.18.3.3 Settings section

A complete description for the `[settings]` section is:

[settings]	Description
inf	Special values may need to be mapped to numeric values so the values can be used in formula's etc. This setting will specify the value for the GAMS `INF` quantity. The default is the string `INF`.
mininf	This is the mapped value for `-INF`. The default is `-INF`.
eps	This is the mapped value to be used for `EPS`. The default is `EPS`.
na	This is the mapped value to be used for `NA`. The default is `NA`.
undf	This is the mapped value to be used for `UNDF`. The default is `UNDF`.
scalarsheet	When this parameter is set to 1, GDX2XLS will generate a separate sheet to collect scalar parameters, scalar equations and scalar variables. This can reduce the number of sheets created with just a single data item. The name of the sheet is fixed: `scalar`. By default this option is turned on.
tableformatting	If this option is turned on, extra table formatting is used (adding colors, etc.) to make the tables look better. If this is not needed, this option can be turned off. Default: `tableformatting=0`.
toc	Whether or not to add a {*Table* of Contents} sheet. Default is to generate such a table.
sorttoc	Whether or not to sort the table of contents alphabetically. If turned off, the table will be displayed in the order in which the identifiers appear in the GDX file. Default is to sort.
autofilter	Automatically generate `AutoFilter` enabled tables in Excel.
freezeheader	Keep headers fixed so they don't scroll off the screen.
indexformat	Custom format for index columns. By default this is an empty string.
valueformat	Custom format for value columns. By default this is an empty string.

An example of setting special values can be found in section Model gdx2xls4: special value mapping.

6.18.3.4 Colors section

A complete description for the `[colors]` section is:

[colors]	Description
header	The colorindex to be used as background for table headers. Default is 17.
body	The colorindex to be used as background for table bodies. Default is 19.
italics	The colorindex to be used for the font when writing explanatory text. The default is light grey (color index 48).

The `[xmlcolors]` section is used to specify colors in the XML file to be generated.

6.18.3.5 Custom formats

The format strings consists of four pieces:

```
[format for $x>0$];[format for $x<0$];[format for $x=0$];[format for strings]
```

An example given in the Excel help is:

```
#,###.00_);[Red](#,###.00);0.00;"sales "@
```

The codes used here have the following meaning:

Formatting Characters	Description
# (number sign)	displays only significant digits and does not display insignificant zeros.
, (comma)	To display a comma as a thousands separator or to scale a number by a multiple of one thousand, include a comma in the number format.
0 (zero)	displays insignificant zeros if a number has fewer digits than there are zeros in the format.
_ (underscore)	To create a space the width of a character in a number format, include an underscore, followed by the character. For example, when you follow an underscore with a right parenthesis, such as `_)`, positive numbers line up correctly with negative numbers that are enclosed in parentheses.
[color]	One of `[Black]`, `[Blue]`, `[Cyan]`, `[Green]`, `[Magenta]`, `[Red]`, `[White]`, `[Yellow]`.
@ (at sign)	Include an at sign (`@`) in the section where you want to display any text entered in the cell.

Additional formatting characters include:

Formatting Characters	Description
? (question mark)	adds spaces for insignificant zeros on either side of the decimal point so that decimal points align when formatted with a fixed-width font, such as Courier New. You can also use ? for fractions that have varying numbers of digits.
condition	Conditions can be specified as follows: `[Red][<=100];[Blue][>100]`.
exponent	To display numbers in scientific format, use exponent codes in a section, for example, `E-`, `E+`, `e-`, or `e+`.

A useful format is:

```
[settings]
valueformat=#.????
```

which aligns numbers on the decimal point and depicts zero's as dots just as the listing file is doing.

6.18.4 Examples

6.18.4.1 Model gdx2xls1: import trnsport.gdx

This example will solve the `trnsport.gms` model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Excel and MS Excel is launched to inspect the results. This is a small example that should run very quickly.

```
$onText
Test of GDX2XLS. Dumps all symbols of
trnsport.gms to trnsport.xlsx.
$offText

execute '=gamslib trnsport';
execute '=gams trnsport lo=3 gdx=trnsport';
execute '=gdx2xls trnsport.gdx';
executeTool 'win32.shellExecute trnsport.xlsx';
```

Notes: the equal signs in from of the external programs indicate we don't go through a shell (e.g. `command.com` or`cmd.exe`). This will improve reliability in case the external program is not found. In such a case a proper error will be triggered. Without the '=' such errors go undetected and the GAMS model will continue.

The command 'executeTool 'win32.shellExecute trnsport.xlsx';` will launch Excel to view the .XLSX file.

6.18.4.2 Model gdx2xls2: import indus89.gdx

This example will solve the `indus89.gms` model from the model library and generate a GDX file containing the complete symbol table. This GDX file is exported to Excel and MS Excel is launched to inspect the results. This is a fairly large GDX file, with many identifiers, resulting in many sheets in the workbook.

```
$onText
Test of GDX2XLS. Dumps all symbols of
indus89.gms to indus89.xlsx. This takes
longer as there is a large number of symbols.
$offText

execute '=gamslib indus89';
execute '=gams indus89 lo=3 gdx=indus89';
execute '=gdx2xls indus89.gdx';
executeTool 'win32.shellExecute indus89.xlsx';
```

6.18.4.3 Model gdx2xls3: a large table

This is an artificial example where we generate a large identifier in GAMS: a parameter with as many elements as the number of rows that Excel can handle.

```
$onText
Test of GDX2XLS. Single symbol with 65536-3=65533 records.
Maximum rows that XLS can handle is 65536; an XLSX file allows for slightly more than a million rows
$offText

Set i / i1*i65533 /;

Parameter p(i);
p(i) = uniform(-100,100);
execute_unload 'test.gdx', p;

execute '=gdx2xls test.gdx';
executeTool 'win32.shellExecute test.xlsx';
```

If you create a spreadsheet with too many rows, the XLSX file writer will return `OLE error 800A03EC`. When generating an XML file, an error will occur when Excel loads the file.

6.18.4.4 Model gdx2xls4: special value mapping

To store special values like `INF`, `EPS`, `NA` in a numeric field in the database, GDX2XLS uses a mapping. This mapping can be changed using an INI file.

```
$onText
Test of GDX2XLS.
Check special value mapping.
$offText

$onEcho > m.ini
[settings]
inf=1.0e100
mininf=-1.0e100
eps=0.0
na=#NA!
undf=#UNDF!
$offEcho

Parameter p(*) / i1 inf, i2 -inf, i3 eps, i4 na /;
p('i5') = 1/0;
display p;


* save parameter p in p.xlsx
* special values are translated to default values:
execute_unload "p.gdx", p;
execute '=gdx2xls p.gdx';
executeTool 'win32.shellExecute p.xlsx';

* save parameter p in q.xls using new mapping
* INF -> 1.0e100  (numeric)
```

```
* -INF -> -1.0e100  (numeric)
* EPS -> 0.0  (numeric)
* NA -> #NA! (string)
* UNDF -> #UNDF! (string)
*
execute_unload "q.gdx", p;
execute '=gdx2xls q.gdx @m.ini';
executeTool 'win32.shellExecute q.xlsx';
```

Numeric values are important if you want Excel being able to operate on these numbers.

6.18.4.5 Model gdx2xls8: custom format

We use a custom value format to color the different values `x<0`, `x=0`, `x>0` differently. Also align on the decimal point.

```
$onText
GDX2XLS example: use of custom format
$offText

$onEcho > mexls.ini
[settings]
valueformat=[Blue]#.????;[Red]-#.????;[Green]0.????;[Magenta]
$offEcho

execute '=gamslib mexls';
execute '=gams mexls lo=3 gdx=mexls';
execute '=gdx2xls mexls.gdx @mexls.ini';
executeTool 'win32.shellExecute mexls.xlsx';
```

6.18.4.6 Model gdx2xls9: custom format 2

This uses the more useful custom format `valueformat=#.????` (see Figure 7).

```
$onText
GDX2XLS example: use of custom format
$offText

$onEcho > align.ini
[settings]
valueformat=#.????
$offEcho

execute '=gamslib mexls';
execute '=gams mexls lo=3 gdx=mexls';
execute '=gdx2xls mexls.gdx @align.ini';
executeTool 'win32.shellExecute align.xlsx';
```

6.19 GDXCOPY

GDX files from different GAMS version can possibly be incompatible. A current GAMS system can read all older GDX file formats. The GDXCOPY utility provides a mechanism to convert GDX files to a format that different GAMS systems can read.

6.19.1 Usage

```
gdxcopy option inFile outDir
```

or

```
gdxcopy option -Replace inFile
```

The first form copies the converted files to a directory; the second form replaces the original file(s).

inFile

Single file or a file pattern with .gdx file extension.

outDir

The output directory.

Instead of converting the files explicitly using the GDXCOPY utility, files can also be converted by using the environment variable GDXCONVERT with values V5, V6 or V7. The values specified will be used together with the value of the environment variable GDXCOMPRESS to call GDXCOPY as soon as a GDX file is created.

The values of the environment variables can also be set using the GAMS parameters GDXCONVERT and GDXCOMPRESS.

Option:

Option	Target format
-V5	Version 5
-V6U	Version 6 uncompressed
-V6C	Version 6 compressed
-V7U	Version 7 uncompressed
-V7C	Version 7 compressed

A current GAMS system can always handle older GDX file formats. The GDXCOPY utility provides a mechanism to convert GDX files to a prior format, so an older GAMS system can read these files.

Notes:

- Version 7 formatted files were introduced with version 22.6 of GAMS; version 6 formatted files were introduced with version 22.3 of GAMS. Prior versions used version 5.

- Some features introduced in version 7 of the GDX file format cannot be represented in older formats.

Feature	Action taken
Dimension > 10	Symbol is ignored
Identifier longer than 31 characters	Truncated to 31 characters
Unique element longer than 31 characters	Truncated to 31 characters
Domain of a symbol	Domain is ignored
Aliased symbol	Symbol is entered as a set
Additional text for symbol	Additional text is ignored

- The macOS systems do not support GDX conversion into format version 6 and version 5.

6.19.2 Example

In the example below we convert all GDX files to a compressed format for version 6.

```
dir
  1,219 t1.gdx
  1,740 t0.gdx
889,973  i.gdx
  1,740 pv.gdx
894,672 bytes

gdxcopy -v6c *.gdx newdir
dir newdir
  1,219 t1.gdx
  1,219 t0.gdx
203,316  i.gdx
  1,219 pv.gdx
206,973 bytes
```

6.20 GDXDIFF

The GDXDIFF tool compares the data of symbols with identical name, type and dimension in two GDX files and writes the differences to a third GDX file. A summary report will be written to standard output.

6.20.1 Usage

```
gdxdiff file1 file2 {diffile} {options}
```

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory. GDXDIFF requires two parameters, the file names of two GDX files. An optional third parameter is the name of the GDX difference file. Without the third parameter, the difference file will be diffile.gdx in the current directory.

diffile = *fileName* (default = diffile.gdx)

An optional name of the GDX difference file.

6.20.2 Options

The following options can be used when calling GDXDIFF:

Option	Default	Description
Eps	0.0	Epsilon for comparison (absolute).
RelEps	0.0	Epsilon for comparison (relative).
field	all	Specify a single subfield (`l`, `m`, `up`, `lo`, `prior`, `scale`) of a variable or equation to be compared.
fldOnly	disabled	Write variables and equations as parameters for the selected subfield.
id	all	Define specific identifiers of the GDX files to be compared.
diffOnly	disabled	Controls if differences of variables and equations will be written as parameters or not.
cmpDefaults	disabled	Enables the comparison of default values.
cmpDomains	disabled	Enables the comparison of symbol domains.
matrixFile	disabled	Enables the comparison of GAMS matrix files in GDX format.
ignoreOrder	disabled	Ignores UEL order of input files to reduce size of output file.
setDesc	Y	Control if associated text of matching set elements is compared.

Some more detailed remarks on the options:

Eps = *value* (default = 0.0)

Absolute difference for comparisons; see also Comparing numeric values. If the difference between two values exceeds `Eps`, a difference will be reported. The valid range is

RelEps = *value* (default = 0.0)

Relative difference for comparisons; see also Comparing numeric values. If the value of `RelEps` is exceeded, a difference will be reported.

field = *fieldName* (default = all)

The specified subfield is the only field used for deciding if a variable or equation is different. FieldName is one of the following: `l`, `m`, `up`, `lo`, `prior`, `scale` or `all`.

fldOnly (disabled by default)

Used in combination with the field option; The variables and equations will be written as parameters for the selected subfield. This option cannot be used in combination with diffOnly and requires field being set to an actual field name but not to `all`.

id = *identifier* (default = all)

Limits the comparison to one or more symbols; symbols not specified will be ignored. Multiple identifiers can be specified as: `id=id1 id=id2` or as `id="id1 id2"`. When using `GDXDIFF` from the menu bar in the GAMS IDE (Utilities), this option is not available.

diffOnly (disabled by default)

Differences for variables and equations will be written as parameters; each parameter will have an additional index which is used to store the field name. Only fields that are different will be written. This option cannot be used in combination with `fldOnly`.

cmpDefaults

Enables the comparison of default values. When using `GDXDIFF` from the menu bar in the GAMS IDE (Utilities), this option is not available.

cmpDomains (disabled by default)

Enable the comparison of symbol domains. Note that the difference are not listed in particular in the diffile. When using `GDXDIFF` from the menu bar in the GAMS IDE (Utilities), this option is not available.

matrixFile

This activates a special mode to compare GAMS matrix files in GDX format. This is mostly done for internal use. When using `GDXDIFF` from the menu bar in the GAMS IDE (Utilities), this option is not available.

ignoreOrder

By default, GDXDiff preserves the UEL order of the input files. If this is set, this is disabled so that records in the output file could show up in a different order than in the input files. Doing this, the output file can be reduced in size.

setDesc = *boolean* (default = `Y`)

Enable or disable the comparison of associated texts for set elements.

Criterion for comparing numeric Values
The use of Eps and RelEps is best described by the code fragment below.

```
AbsDiff := Abs(V1 - V2);
if  AbsDiff <= EpsAbsolute
then
  Result := true
else
  if EpsRelative > 0.0
  then
     Result := AbsDiff / (1.0 + DMin(Abs(V1), Abs(V2))) <= EpsRelative
  else
     Result := false;
```

Interpreting the Labels in the diffile
Only symbols with the same name, type and dimension will be compared. Tuples with different values are written to the GDX difference file, and a dimension is added to describe the difference using the following labels:

- `ins1` indicates that the tuple only occurs in the first file.
- `ins2` indicates that the tuple only occurs in the second file.
- `dif1` indicates that the tuple occurs in both files; contains the value from the first file.
- `dif2` indicates that the tuple occurs in both files; contains the value from the second file.

6.20.3 Examples

6.20.3.1 Compares two GDX Files and writes the Differences to a third GDX File

In the following example, the [**trnsport**] model is solved twice with different capacity data. GDX files are saved for each run, and compared afterwards using `GDXDIFF`. The shipments variable is loaded into a new variable used for a display statement. We introduce four new unique elements that are used in the difference file.

```
* solve and write to unmodified.gdx before manipulating the data
solve transport using lp minimizing z;
execute_unload 'unmodified.gdx', a, x;

* manipulate the data and solve again, write to modified.gdx
a('seattle') = 1.2*a('seattle');
solve transport using lp minimizing z;
execute_unload 'modified.gdx', a, x;

execute 'gdxdiff unmodified modified diffile > %system.nullfile%';

* Declare symbols to hold the data for differences
Set difftags / dif1, dif2, ins1, ins2 /;
Variable xdif(i,j,difftags);
Parameter adif(i,difftags);

execute_load 'diffile' adif=a, xdif=x;
display a, xdif.l;
```

The display statement generates the following output in the listing file:

```
----    101 PARAMETER a  capacity of plant i in cases

seattle   420.000,    san-diego 600.000


----    101 VARIABLE xdif.L

                           dif1        dif2

seattle  .new-york       50.000     120.000
san-diego.new-york      275.000     205.000
```

Alternatively, one can open the diffile in GAMS Studio to display the differences.

This example is also part of the GAMS Data Utilities Library, see model [**GDXDIFFExample16**] for reference.

6.21 GDXDUMP

`GDXDUMP` is a tool to write scalars, sets, parameters (tables), variables and equations from a GDX file formatted as a GAMS program with data statements to standard output, GMS or CSV files. To write to a file, use the output redirection '>' provided by the operating system or the output option of `GDXDUMP`.

6.21.1 Usage

```
gdxdump filename {options}
```

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

6.21.2 Options

The table summarizes the options of GDXDUMP. Running GDXDUMP without any arguments will write the sets, parameters, scalars etc. to standard output formatted as a GAMS program with data statements.

Option	Default	Description
output	none	Specify the output filename.
version	disabled	Writing version information only.
symb	none	Specify a single identifier to be written from the GDX.
UelTable	none	Write all unique elements in the GDX to a single set.
delim	if format=normal: period if format=csv: comma	Specify a delimiter to separate elements in the output.
decimalSep	period	Specify a decimal separator.
noHeader	disabled	Suppress writing the header information.
header	none	Specify a new header when writing to CSV.
noData	disabled	Only write the headers of the symbols.
CSVAllFields	disabled	Controls the writing of subfields of a variable or equation to CSV.
CSVSetText	disabled	Controls the writing of set element text to CSV.
symbols	disabled	Generate an alphabetical list of all symbols in the GDX file.
symbolsAsSet	disabled	Write some basic information for all symbols in the GDX as set.
symbolsAsSetDI	disabled	Write some basic information including domain information for all symbols in the GDX as set.
domainInfo	disabled	Generate an alphabetical list of all symbols in the GDX file that includes domain information.
setText	disabled	Show the set text (aka the associated text) in the GDX file.
format	normal	Specify the output file format.
dFormat	normal	Specify the numerical format in the output file.
cDim	N	Controls the writing of the column header when using the CSV format.
filterDef	Y	Controls the writing of default values.
EpsOut	EPS	String to be used when writing the value for 'Epsilon'.
NaOut	NA	String to be used when writing the value for 'Not Available'.
pInfOut	+Inf	String to be used when writing the value for 'Positive Infinity'.
mInfOut	-Inf	String to be used when writing the value for 'Negative Infinity'.
UndfOut	Undf	String to be used when writing the value for 'Undefined'.
ZeroOut	0	String to be used when writing the value for 'Zero'.

Some more detailed remarks on the options:

output = *fileName* (default=none)

Write the output to a file. The .gms resp. the .csv file extension must be added when writing to GMS resp. CSV.

This option is demonstrated in Writing GDX to CSV.

-version (disabled by default)

Synonym: -v

Write version information only and terminate; all other options will be ignored.

symb = *identifier* (default=none)

Specify a single identifier in the GDX file to be written to standard output, GMS or CSV. When writing to CSV, one must specify the `symb` option.

This option is demonstrated in Writing GDX to standard output and to GMS.

UelTable = *identifier* (default=none)

Write all unique elements found in the GDX file to a set using the identifier specified as the name for the set.

This option is demonstrated in Writing GDX to standard output and to GMS.

delim = [period, comma, tab, blank, semiColon] (if format=normal: default=period, if format=csv: default=comma)

Selects a different delimiter to separate unique elements; period is the default when writing to standard output or GMS, while comma is the default when writing to CSV.

This option is demonstrated in Writing GDX to CSV.

decimalSep = [period, comma] (default=period)

Specify a decimal separator.

This option is demonstrated in Writing GDX to CSV.

noHeader (disabled by default)

Suppress the header information when writing a single symbol; only the data for the symbol will be written, not its declaration. The option is ignored when writing all symbols. When writing to CSV, the header row can be suppressed by enabling `noHeader`.

This option is demonstrated in Adding double Quotes to an user defined Header when writing to CSV.

header = *string* (default=none)

The string supplied replaces the default header written by the program to a CSV file. If an empty header is desired, the string can be empty; such a string can be written using two single quotes (header=''), while in general it is best to enclose the string with double quotes.

This option is demonstrated in Writing GDX to CSV and Adding double Quotes to an user defined Header whe

noData (disabled by default)

Only write the headers for the symbols; no data is written. The option is ignored when writing to CSV.

CSVAllFields (disabled by default)

When writing a variable or equation to CSV, all fields (level, marginal, lower, upper, and scale) will be written. Without this option, only the level will be written. When writing a set the option control the writing of the element text.

This option is demonstrated in Writing GDX to CSV.

CSVSetText (disabled by default)

When writing a set to CSV, the set element text will be written as the last column in the CSV file in addition to the set elements. Without this option, only the set elements will be written.

This option is demonstrated in Writing GDX to CSV.

symbols (disabled by default)

Generate an alphabetical list of all symbols in the GDX file (not valid when writing to CSV).

This option is demonstrated in Writing GDX to standard output and to GMS.

symbolsAsSet (disabled by default)

Generate a set declaration where the data represents basic information (symbol identifier, dimension, type, explanatory text) of all symbols in the GDX file (not valid when writing to CSV).

This option is demonstrated in Writing GDX to standard output and to GMS.

symbolsAsSetDI (disabled by default)

Generate a set declaration where the data represents basic information (symbol identifier, dimension, type, domain information) of all symbols in the GDX file (not valid when writing to CSV).

domainInfo (disabled by default)

Generate an alphabetical list of all symbols in the GDX file that includes domain information (not valid when writing to CSV). The column `DomInf` can have the following values:

`N/A` - The function to get the type of domain information is not available
`None` - No domain was specified (domain is the universe)
`Relaxed` - The domain is relaxed, i.e. the identifiers shown do not necessarily represent one dimensional sets
`Regular` - Regular domain; the identifiers shown are one dimensional sets

This option is demonstrated in Writing GDX to standard output and to GMS.

setText (disabled by default)

Show the set text (aka the associated set text or the set element text) in the GDX file (not valid when writing to CSV).

GDX allows a string of text to be associated with each element of a set. The universe of such strings stored for use in any particular GDX file (i.e. the set text list) can be shown with this option.

format = [normal, gamsbas, CSV] (default=normal)

Change the output format and the symbols written.

When using the gamsbas format, the program will not write the declarations for the symbols and only write the Level and Marginal assignment statements for the variables, and the Marginal assignment statements for equations.

The CSV format adds column headers to the output. By enabling the cDim option the unique elements of the last dimension of the symbol will be used as column headers for the values. If domain information is available, the column headers will be made unique if overlapping names have been used for the names of the index positions. If no domain information is available, the index names used will be of the form `dim1`, `dim2`, ...

In order to run `GDXDUMP`, one must specify a single symbol using the symb option when writing to CSV.

The gamsbas and CSV format is demonstrated in Writing GDX to standard output and to GMS resp. Writing GDX to CSV.

dFormat = [normal, hexponential, hexBytes] (default=normal)

Specify the numerical format in the output file.

cDim = [`Y`, `N`] (default = `N`)

Can be used when writing a CSV file; when enabled, the unique elements of the last dimension will be used as column headers for the values.

This option is demonstrated in Writing GDX to CSV.

filterDef = [`Y`, `N`] (default = `Y`)

When enabled, default values will be filtered and not written. This option is enabled by default. For example, if the Level field (`.L`) of a variable is zero, the value will not be written.

EpsOut = *string* (default = `EPS`)

String to be used when writing the value for 'Epsilon'.

NaOut = *string* (default = `NA`)

String to be used when writing the value for 'Not Available'.

pInfOut = *string* (default = `+Inf`)

String to be used when writing the value for 'Positive Infinity'.

This option is demonstrated in Writing GDX to CSV.

mInfOut = *string* (default = `-Inf`)

String to be used when writing the value for 'Negative Infinity'.

UndfOut = *string* (default = `Undf`)

String to be used when writing the value for 'Undefined'.

zeroOut = *string* (default = 0)

String to be used when writing the value for 'Zero'.

Note that GDX files can also be viewed using the GAMS IDE or GAMS Studio.

6.21.3 Examples

To demonstrate the features of GDXDUMP in the Writing GDX to standard output and to GMS and Writing GDX to CSV examples, we execute the model trnsport.gms ([TRNSPORT] from GAMS Model Library) initially to create the GDX file trnsport.gdx by using the GDX command line option. Afterwards, some important features of GDXDUMP are demonstrated.

```
gams trnsport gdx=trnsport
```

The figure shows the symbol listing of the GDX file in total and the variable x in particular, since those data is used quite often in the following examples:

All subfields of the variable x are shown in the next figure:

6.21.3.1 Writing GDX to standard output and to GMS

In this example, we will demonstrate the effect of some basic features of GDXDUMP.

While we write to standard output in the most sections of this example, one can also write the GDX file contents into a GAMS file using the following command to redirect the output:

```
gdxdump trnsport.gdx > GDXContents.gms
```

Alternatively, the output option can be used to specify an output filename.

Option symbols
This listing shown above can also be written to standard output by using the symbols option of GDXDUMP:

```
gdxdump trnsport symbols
```

The GDXDUMP program writes the following output (except for the numbering and the records column, the content is identical compared to the figure):

```
   Symbol Dim Type Records  Explanatory text
 1 a        1  Par       2  capacity of plant i in cases
 2 b        1  Par       3  demand at market j in cases
 3 c        2  Par       6  transport cost in thousands of dollars per case
 4 cost     0  Equ       1  define objective function
 5 d        2  Par       6  distance in thousands of miles
 6 demand   1  Equ       3  satisfy demand at market j
 7 f        0  Par       1  freight in dollars per case per thousand miles
 8 i        1  Set       2  canning plants
 9 j        1  Set       3  markets
10 supply   1  Equ       2  observe supply limit at plant i
11 x        2  Var       6  shipment quantities in cases
12 z        0  Var       1  total transportation costs in thousands of dollars
```

Option domainInfo
We use the domainInfo option to write some basic information including domain information (column DomInf) to standard output:

```
gdxdump trnsport domainInfo
```

The GDXDUMP program writes the following output:

```
SyNr  Type  DomInf Symbol
  3   Par Regular a(i)
  4   Par Regular b(j)
  7   Par Regular c(i, j)
 10   Equ    None cost
  5   Par Regular d(i, j)
 12   Equ Regular demand(j)
  6   Par    None f
  1   Set    None i(*)
  2   Set    None j(*)
 11   Equ Regular supply(i)
  8   Var Regular x(i, j)
  9   Var    None z
```

Check the domainInfo option explanation linked above to get a full list of possible values for DomInf and their interpretation. In this example, the symbol dimensions were not specified (none) or they are defined on the one dimensional sets i and j (regular).

Option symbolsAsSet
Using the symbolsAsSet option one can write the information displayed by enabling the symbols option to a set declaration:

```
gdxdump trnsport symbolsAsSet
```

The GDXDUMP program writes the following:

```
alias (Symbol, Dim, Type, *)
set    gdxitems(Symbol,Dim,Type)  Items in the GDX file /
"i".1."Set" "canning plants",
"j".1."Set" "markets",
"a".1."Par" "capacity of plant i in cases",
"b".1."Par" "demand at market j in cases",
"d".2."Par" "distance in thousands of miles",
"f".0."Par" "freight in dollars per case per thousand miles",
"c".2."Par" "transport cost in thousands of dollars per case",
"x".2."Var" "shipment quantities in cases",
"z".0."Var" "total transportation costs in thousands of dollars",
"cost".0."Equ" "define objective function",
"supply".1."Equ" "observe supply limit at plant i",
"demand".1."Equ" "satisfy demand at market j"
/;
```

Option symb
One can write a single identifier from the GDX using the symb option:

```
gdxdump trnsport.gdx symb=x
```

The default output includes the declaration header and a data statement when writing a set or parameter. When writing a variable, the output includes the level and marginal subfields:

```
positive Variable x(i,j) shipment quantities in cases /
'seattle'.'new-york'.L 50,
'seattle'.'chicago'.L 300,
'seattle'.'topeka'.M 0.036,
'san-diego'.'new-york'.L 275,
'san-diego'.'chicago'.M 0.00900000000000001,
'san-diego'.'topeka'.L 275 /;
```

Note the default values are not written (for instance, zero values of the level or lower bound subfields are missing).

Option UelTable
Using the UelTable option one can write a set (named `allUELs` in this example) containing all unique elements of the GDX file:

```
gdxdump trnsport output=allUELs.gms UelTable=allUELs
```

The following set declaration will be part of the output file allUELs.gms:

```
Set allUELs /
  'seattle' ,
  'san-diego' ,
  'new-york' ,
  'chicago' ,
  'topeka' /;
```

Format gamsbas
Using the format option one can write a variable or equation in the gamsbas format, i.e. there will be no declaration of the symbol and the level and marginal subfields will be written as assignments.

```
gdxdump trnsport format=gamsbas
```

`GDXDUMP` writes the following to standard output:

```
x.L ('seattle'.'new-york') = 50 ;
x.L ('seattle'.'chicago') = 300 ;
x.M ('seattle'.'topeka') = 0.036 ;
x.L ('san-diego'.'new-york') = 275 ;
x.M ('san-diego'.'chicago') = 0.00900000000000001 ;
x.L ('san-diego'.'topeka') = 275 ;

z.L  = 153.675 ;
cost.M  = 1 ;
supply.M ('seattle') = Eps ;
demand.M ('new-york') = 0.225 ;
demand.M ('chicago') = 0.153 ;
demand.M ('topeka') = 0.126 ;
```

6.21.3.2 Writing GDX to CSV

Next we specify CSV as the output format and demonstrate some simple features available when writing to CSV.

Writing to CSV
One can write to a CSV file by running the following command:

```
gdxdump trnsport.gdx output=varX.csv symb=x format=csv
```

By doing this, the level subfield of the variable `x` is written as a list to the file varX.csv specified in the output statement. The header row contains the domain set identifiers `i` and `j` of `x` and the field "Val":

```
"i","j","Val"
"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275
```

Option cDim
Using the cDim option one can write the data as a table since the right most dimension is used as the column header:

```
gdxdump trnsport.gdx symb=x format=csv cDim=y
```

We did not specify an output file in this case, so the data will be written to standard output formatted as CSV:

The `GDXDUMP` program writes the following:

```
"i","new-york","chicago","topeka"
"seattle",50,300,0
"san-diego",275,0,275
```

Note the elements new-work, chicago and topeka are written to the header row, since they are elements of the most right dimension `j`.

Customizing the output: CSVAllFields, header, decimalSep, delim, pInfOut etc.
In this section, all subfields of the variable `x` are written to CSV and the representation is customized. The writing of subfields is enabled by CSVAllFields. In addition, the header option is used to overwrite the default header (as seen in the two previous paragraphs). For instance, we want to replace `i` with the term 'canning plants', `j` with the term 'markets' and "Val" with the term 'shipment quantities in cases', while we are fine with the default column labels for the level, marginal etc. subfields, but they will be overwritten, too. Therefore, we must specify them again manually in the `header` option. For some reason, the field separator is changed from comma (default) to semicolon and the decimal separator from period (default) to comma using the delim and decimalSep options. One can define the string written to CSV for special values of GAMS like `+INF` by using the corresponding option pInfOut, which might be useful for further processing of the CSV file.

```
gdxdump trnsport format=csv output=varX.csv symb=x CSVAllFields header="canning plants;markets;shipme
```

GDXDUMP creates the following output:

```
canning plants;markets;shipment quantities in cases;Level;Marginal;Lower;Upper;Scale
"seattle";"new-york";50;0;0;1E+100;1
"seattle";"chicago";300;0;0;1E+100;1
"seattle";"topeka";0;0,036;0;1E+100;1
"san-diego";"new-york";275;0;0;1E+100;1
"san-diego";"chicago";0;0,00900000000000001;0;1E+100;1
"san-diego";"topeka";275;0;0;1E+100;1
```

Note that the fields in the header are not enclosed by double quotes. This problem is addressed in the next section Adding double Quotes to an user defined Header when writing to CSV.

Customizing the output: CSVSetText

The set element text is the *value* of a set element. Often this is left out when defining sets, but for descriptive models often set element text is provided. When dumping a set to a CSV file the set element text is not written to the CSV file by default. The option CSVSetText enables the writing of the text. The model mexss.gms ([MEXSS] from GAMS Model Library) has a set i of steel plants with set element text

```
Set i 'steel plants' / ahmsa     'altos hornos - monclova'
                       fundidora 'monterrey'
                       sicartsa  'lazaro cardenas'
                       hylsa     'monterrey'
                       hylsap    'puebla'                  /;
```

When we create a GDX file and dump set i as CSV with defaults via the following commands

```
gamslib mexss
gams mexss a=c gdx=mexss
gdxdump mexss format=csv output=setI.csv symb=i header="steel_plants"
```

GDXDUMP creates the following output:

```
steel_plants
"ahmsa"
"fundidora"
"sicartsa"
"hylsa"
"hylsap"
```

If we add CSVSetText

```
gamslib mexss
gams mexss a=c gdx=mexss
gdxdump mexss format=csv output=setI.csv symb=i CSVSetText header="steel_plants,real_name"
```

GDXDUMP creates the following output:

```
steel_plants,real_name
"ahmsa","altos hornos - monclova"
"fundidora","monterrey"
"sicartsa","lazaro cardenas"
"hylsa","monterrey"
"hylsap","puebla"
```

6.21.4 Adding double Quotes to an user defined Header when writing to CSV

This sections describes a possible workaround for the following problem: Suppose one wish to write a parameter or variable from GDX to CSV, but with a non default header containing double quotes. For this purpose, one might try to use the `header` option of `GDXDUMP` to specify an user defined header. Though, there is no way to add double quotes enclosing the single fields using the `header` option, but this is a quite common standard in CSV files (for instance, if the fields contain reserved characters).

Initially, define the header you want to write to the CSV file and redirect the line to your final CSV file (which will be created at this point, an already existing file will be overwritten!). Note the double quotes enclosing the field content. Adding the single quotes is necessary in order to run the following statement from GAMS IDE (however, they will not be written to the file), while they must be omitted when using the command prompt.

```
$echo '"canning plants","markets","shipment quantities in cases"' > quotedHeader.csv
```

Afterwards, call `GDXDUMP` by specifying the GDX file, the single symbol `x` we want to write, the output format CSV and redirect the output to the file quotedHeader.csv:

```
$call gdxdump trnsport.gdx symb=x format=csv noHeader >> quotedHeader.csv
```

Appending the output of the recent `GDXDUMP` call to the already existing file quotedHeader.csv adds the data of `x` to the file and creates the following CSV file (note that one must prevent `GDXDUMP` to write the default header by using the command noHeader):

```
"canning plants","markets","shipment quantities in cases"
"seattle","new-york",50
"seattle","chicago",300
"seattle","topeka",0
"san-diego","new-york",275
"san-diego","chicago",0
"san-diego","topeka",275
```

6.22 GDXEncoding

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This converts the unique elements in the GDX file from one [encoding] (https://docs.python.org/3/library/codecs.ht
to another one. The GDX string (a sequence of bytes) is decoded using *encodingIn* (default `latin_1`) into a proper unicode string and afterwards encoded again into a GDX string (byte array) using *encodingOut* (default `utf_8`). If *verbose* is set to 1 (default is 0) the converted unique elements are printed with new and old GDX strings (byte arrays) to the GAMS log. If the number of conversions is necessary in GAMS, a name of a scalar can be passed via *numConv* (default empty) which will hold the number of unique elements converted.

6.22.1 Usage

Command line:

```
gamstool [gdxservice.]GDXEncoding gdxFile [encodingIn=codeIn] [encodingOut=codeOut] [numConv=id gdxOu
```

Compile time:

```
$callTool [gdxservice.]GDXEncoding gdxFile [encodingIn=codeIn] [encodingOut=codeOut] [numConv=id]
```

Execution time:

```
executeTool '[gdxservice.]GDXEncoding gdxFile [encodingIn=codeIn] [encodingOut=codeOut] [numConv=id]
```

Where:

| gdxFile | Name of GDX file. |

The following named parameters are available:

Argument	Description
encodingIn=codeIn	Input encoding of GDX string. Default is latin_1
encodingOut=codeOut	Output encoding of GDX string. Default is utf_8.
numConv=id	GAMS scalar symbol to store the number of actual conversions.
gdxOut=fileOut.gdx	Name of GDX file that contains symbol numConv after execution. Mandatory if called from the command line with argument numConv set, otherwise optional.

6.22.2 Example

```
* UTF-8 encoding
Set c / "côte d'ivoire-3" , "cote d'ivoire-3" /;
Scalar numConv /0/;
$gdxUnload c.gdx c
* Changes the label encoding from UTF-8 to Latin-1:
$callTool gdxservice.gdxEncoding c.gdx encodingIn=utf_8 encodingOut=latin_1 numConv=numConv trace=1
```

The complete example is also part of the GAMS Test Library, see model [**gdxencoding1**] for reference.

6.23 GDXMERGE

The program GDXMERGE combines multiple GDX files into a single GDX file. Symbols with the same name, dimension and type are combined into a single symbol of a higher dimension. The added dimension has the file name of the combined file as its unique element.

6.23.1 Usage

```
gdxmerge filepattern1 filepattern2 .... filepatternN {options}
```

Each file pattern represents a file name or a wildcard representation using ? and ∗. A parameter of the form @filename will process the commands from the text file specified. The result of the GDXMERGE execution will be written to a file called merged.gdx, unless this default is overwritten by the output option.

The .gdx file extension can be omitted. Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory.

6.23.2 Options

The following options can be specified:

id = <*ident1*>, <*ident2*>... (default = none)

Only merge the symbols `ident1`, `ident2`, ...

exclude = <*ident1*>, <*ident2*>... (default = none)

Merge all symbols except for `ident1`, `ident2`` , ...

big = <*integer*>

The size for big symbols.

strict = true|false (default = false)

The program terminates with an error (non-zero return code) if input files specified cannot be found or a file pattern results in no files. An already existing output file also triggers an error if strict is set to true.

output = *fileName* (default = merged.gdx)

The optional output file name.

All symbols with matching type and dimension will be merged. By specifying the parameter `id=ident1` the merge process will only be performed for the identifier(s) specified, while `exclude=ident1` indicates that all symbols should be merged except for the ones specified in the `exclude` list. Note that the two options `id` and `exclude` are mutually exclusive.

By default, the program reads all GDX once and stores all data in memory before writing the merged.gdx file. The `big` parameter is used to specify a cutoff for symbols that will be written one at a time. Each symbol that exceeds the size will be processed by reading each GDX file and only process the data for that symbol. This can lead to reading the same GDX file many times, but it allows the merging of large data sets. The formula used to calculate the cutoff is:

`dimension ∗ totalNumberOfElements`.

The calculated value is doubled for variables and equations.

In addition to the symbols written, a set is added to the GDX file representing all the files processed during the merge operation. The name of the set is `Merged_set_1`, and is made unique by changing the number. The explanatory text for each set element contains the date and time of the GDX file processed.

Note

- The file merged.gdx, or the file specified with the output parameter, will never be used in a merge operation even if the name matches a file pattern.
- Symbols with dimension 20 cannot be merged, because the resulting symbol will have dimension 21 which exceeds the maximum dimension allowed by GAMS.

6.23.3 Examples

6.23.3.1 Merging several GDX Files

In this example, we solve the [**trnsport**] model from the GAMS Model Library using different LP solvers. After each run, we write all symbols to a GDX file and merge the files into one file. The variable `X` is read from the merged file and displayed.

```
$call gamslib trnsport
$call gams trnsport lp=cplex gdx=cplex
$call gams trnsport lp=xpress gdx=xpress
$call gams trnsport lp=conopt gdx=conopt
$call gams trnsport lp=minos gdx=minos
$call gams trnsport lp=snopt gdx=snopt
$call gdxmerge *.gdx

Variable AllX(*,*,*);
$gdxIn merged.gdx
$load AllX=X
$gdxIn

option  AllX:5:1:2;
display AllX.L;
```

The display statement generates the following output in the listing file:

```
----     22 VARIABLE AllX.L  shipment quantities in cases

           seattle     seattle   san-diego   san-diego
          new-york     chicago    new-york      topeka

conopt                300.00000   325.00000   275.00000
cplex      50.00000   300.00000   275.00000   275.00000
minos      50.00000   300.00000   275.00000   275.00000
snopt      50.00000   300.00000   275.00000   275.00000
xpress                300.00000   325.00000   275.00000
```

Note that the different solutions are combined into a single symbol of a higher dimension. The filenames (resp. the solver used) are added as unique elements.
Instead of using the display statement, we can also use the GAMS IDE or GAMS Studio to display the merged.gdx file. The following figure shows the contents of merged.gdx after selecting the level subfield of the variable to be displayed and arranging the display:

This example is also part of the GAMS Data Utilities Library, see model [**GDXMERGEExample17**] for reference.

6.23.3.2 Recursively Merging GDX Files

Some users generate data scenarios for a model in a hierarchical way and store the results in a directory tree: The following example demonstrates how to use three independent scenario scalars a, b, and c with values *low*, *medium*, and *high* to generate 3∗3∗3=27 scenarios, store these in a directory tree, and merge them recursively. The resulting GDX file has for each scenario scalar a new dimension which makes this data especially suitable for viewing in a pivot table:

```
----     39 PARAMETER p  commodity price

                                new-york     chicago      topeka

a_high.b_high.c_high.report        0.25        0.17        0.14
a_high.b_high.c_low .report        0.20        0.14        0.11
a_high.b_high.c_med .report        0.23        0.15        0.13
a_high.b_low .c_high.report        0.25        0.17        0.14
a_high.b_low .c_low .report        0.20        0.14        0.11
a_high.b_low .c_med .report        0.23        0.15        0.13
a_high.b_med .c_high.report        0.25        0.17        0.14
a_high.b_med .c_low .report        0.20        0.14        0.11
a_high.b_med .c_med .report        0.23        0.15        0.13
a_low .b_high.c_high.report        0.25        0.17        0.14
a_low .b_high.c_low .report        0.20        0.14        0.11
a_low .b_high.c_med .report        0.23        0.15        0.13
...
```

The GAMS program that creates the directory structure and runs the model for all 27 scenarios looks as follows:

```
set a,b,c;
parameter
    ascen(a<) / a_low 0.9, a_med 1.0, a_high 1.1 /
    bscen(b<) / b_low 0.9, b_med 1.0, b_high 1.1 /
    cscen(c<) / c_low 0.9, c_med 1.0, c_high 1.1 /;

loop((a,b,c),
   put_utility 'shell' / 'mkdir ' a.tl:0 '\' b.tl:0 '\' c.tl:0
   put_utility 'exec'  / 'gams t.gms lo=2 -idir "%gams.wdir%" -curdir ' a.tl:0 '/' b.tl:0 '/' c.tl:0
);
```

The program creates the directory tree (by creating the leave nodes of the tree), changes into the leave directory (`curDir`) and calls GAMS (using `iDir` to point to the starting location). The program `t.gms` is almost an identical copy of the [**trnsport**] model with the last few lines changed:

```
* ...
a(i) = %a%*a(i);
b(j) = %b%*b(j);
c(i,j) = %c%*c(i,j);

solve transport using lp minimizing z;

scalar tmodelstat;
tmodelstat = transport.modelstat;
parameter p(j) 'commodity price';
p(j) = demand.m(j);
execute 'test -d results || mkdir results';
execute_unload 'results/input', a, b, c;
execute_unload 'results/report', x, z, p, tmodelstat;
```

This code saves the results of a run in to GDX files `input.gdx` and `report.gdx` in a `results` subdirectory. After the program has executed the 27 runs of `t.gms` it executes some Python code that walks recursively through the directory tree and merges GDX files it finds in a `results` folder in the leaf nodes percolates the results up in the tree by calling `gdxmerge` on each high node level. The Python function `os.walk` with argument `topdown=False` ensure the walking of the tree in the correct order. The complete example can be found in the GAMS Model Library in model [**scenmerge**]:

```
embeddedCode Python:
import os
gd = -1
for root, dirs, files in os.walk(os.path.normpath(r"C:\Users\test\allscen\ ".rstrip()), topdown=Fals
  if 'results' in dirs:
    if gd == -1:
      gd = root.count(os.path.sep)
      gdr = root;
    elif not gd == root.count(os.path.sep):
      raise NameError('results subdirs found at different depths in the tree: ' + root + ' and ' + g
    cmd = 'cd "' + root + '" && gdxmerge results' + os.path.sep + '* output=' + root.split(os.path.s
    print('Merging results in ' + root)
    if not 0 == os.system(cmd + '>' + os.devnull):
      raise NameError('problems running: ' + cmd)
  elif root.count(os.path.sep) < gd:
    input = ''
    for d in dirs:
      if os.path.isfile(os.path.join(root, d, d + '.gdx')):
        input = input + ' ' + os.path.join(d, d + '.gdx')
    if len(input):
      cmd = 'cd "' + root + '" && gdxmerge' + input + ' output=' + root.split(os.path.sep)[-1]
      print('Merging' + input + ' in ' + root)
      print(cmd)
      if not 0 == os.system(cmd + '>' + os.devnull):
        raise NameError('problems running: ' + cmd)
print('All done. Final result file: ' + root.split(os.path.sep)[-1] + '.gdx')
endEmbeddedCode
```

6.24 GDXMRW

Interfacing GAMS and MATLAB. This document briefly describes GDXMRW (GDX-Matlab Read/Write), a suite of utilities to exchange data between GAMS and MATLAB. The software gives MATLAB users the ability to use all the optimization capabilities of GAMS, and allows visualization of GAMS models directly within MATLAB. The most recent version of GDXMRW is included as part of the current GAMS Distribution.

Author

Steven Dirkse (sdirkse@gams.com), GAMS Development Corp.

Michael C. Ferris (ferris@cs.wisc.edu), Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton Street, Madison, Wisconsin 53706

Jagdish Ramakrishnan (jramakrishn2@wisc.edu), Wisconsin Institute for Discovery, University of Wisconsin - Madison, 330 North Orchard Street, Madison, Wisconsin 53715

Date

August 1, 2014

Attention

GDXMRW is deprecated (see GAMS 38 GDXMRW release notes). Please use **GAMS Transfer Matlab** instead.

6.24.1 Introduction

Optimization is becoming widely used in many application areas as can be evidenced by its appearance in software packages such as Excel and MATLAB. While the optimization tools in these packages are useful for small-scale nonlinear models (and to some extent for large linear models), the lack of a capability to compute automatic derivatives makes them impractical for large scale nonlinear optimization. In sharp contrast, modeling languages such as GAMS and AMPL have had such a capability for many years, and have been used in many practical large scale nonlinear applications.

On the other hand, while modeling languages have some capabilities for data manipulation and visualization (e.g., Rutherford's GNUPLOT), specialized software tools like Excel and MATLAB are much better at these tasks.

This paper describes a link between GAMS and MATLAB. The aim of this link is two-fold. Firstly, it is intended to provide MATLAB users with a sophisticated nonlinear optimization capability. Secondly, the visualization tools of MATLAB are made available to a GAMS modeler in an easy and extendable manner so that optimization results can be viewed using any of the wide variety of plots and imaging capabilities that exist in MATLAB.

In order to enable this link between GAMS and MATLAB, we have implemented MATLAB callable functions that can efficiently import and export data to and from GAMS through GDX files. The simplest read and write functions, `irgdx` and `iwgdx`, deal with indexed parameters. Without getting into their precise meaning, these parameters are essentially indexed in each dimension by the sequence of integers from 1 to the size of that dimension. As an example of the read function, the following command in MATLAB will store the matrices `A` and `B` into the caller workspace after reading from the file `foo1.gdx`:

```
>> irgdx('foo1','A','B');
```

Similarly, the call

```
>> iwgdx('foo2','C','D');
```

would write the matrices `C` and `D` (located in the MATLAB workspace) as indexed parameters into a file named `foo2.gdx`. Note that, many of the examples found in this manual are also part of the GAMS data utilities models library, referred to here as **datalib**. The statements above can be found in datalib example `gdxmrw_intro02_init`. For reading and writing more complex data such as sets, variables, equations, and non-indexed parameters, we can use the more sophisticated functions `rgdx` and `wdgx`. Further descriptions of these functions together with advanced use of `irgdx` and `iwgdx` are detailed in the rest of this paper.

In Section Data Transfer, we discuss the data transfer utilities that allow importing and exporting data between MATLAB and GDX files: `irgdx`, `iwgdx`, `rgdx`, `wgdx`. In Section Extended use, we give a few examples of the MATLAB and GAMS interface. In APPENDIX A - Configuring GDXMRW, we provide information about configuring GDXMRW and testing the utilities. In APPENDIX B - Utility functions: gdxWhos and gdxInfo, we describe additional utility functions, `gdxWhos` and `gdxInfo`, that allow viewing contents of GDX files in the MATLAB console. Finally, in APPENDIX C - Calling GAMS model from MATLAB, we discuss the `gams` function that with a single call initializes a GAMS model with MATLAB data, executes GAMS on the model, and returns results back into MATLAB.

6.24.2 Data Transfer

This paper describes a suite of tools for exchanging data between GAMS and MATLAB. This data exchange is accomplished via the GDX (i.e. the GAMS Data eXchange) interface and API. There are many advantages to using GDX, including platform independence, space and time efficiency in storing and accessing data, and a guarantee that all GDX data contains no duplicates and is free from any logical or syntax errors that might prevent it from being read into GAMS. The GDX interface is well tested, available to the public, and is the basis for most if not all of the GAMS data utilities.

In this section we discuss four MATLAB routines. The first two, `irgdx` and `iwgdx`, are the subject of sections irgdx and iwgdx and are used to quickly and simply read indexed parameters from a GDX file into MATLAB and vice versa. For an example use of these functions, a generic quadratic program function that mimics MATLAB's `quadprog` function is provided in datalib example `gdxmrw_qp3`. The `rgdx` and `wgdx` routines (sections rgdx and wgdx) are more sophisticated versions of the first two that can read and write more general GDX data, e.g. sets, variables, equations, and non-indexed parameters. To understand the structure of this GDX data, the material in section Indexing with labels (UELs) is essential.

6.24.2.1 irgdx

`irgdx` is a specialized MATLAB function to do an efficient import of an indexed parameter from a GDX file. The `irgdx` call will not read non-indexed parameters; it can only read data recognized as an indexed data type. If a GDX file consists of both indexed and non-indexed data, `irgdx` can still read the indexed data. To see whether particular parameters in a GDX file are indexed, the `gdxWhos` function described in APPENDIX B - Utility functions: gdxWhos and gdxInfo can be used.

The syntax and functionality of `irgdx` closely resembles that of the MATLAB `load` function to import MATLAB formatted data, i.e data stored in MAT files. In Basic syntax and Load symbols into output structure we describe the basic modes of operation where the data read from GDX are either stored in the caller workspace or into a MATLAB structure on the left-hand side of an assignment. In MATLAB sparse form, we describe an optional specification that allows read results to be stored in MATLAB sparse form. Finally, in Subsection Renaming, we describe a simple syntax that allows the renaming of variables/parameters without the need to make expensive copies.

Basic syntax

The basic syntax for `irgdx` is

```
irgdx('gdxFileName', 'sym1', 'sym2', ...);
```

The above call will read the indexed parameters named sym1, sym2, and so on from the GDX file whose name is specified in the first argument and store the results in the MATLAB caller workspace. Only the GDX name argument is mandatory. If only this argument is specified (i.e. no symbol names are provided) then *all* of the indexed parameters stored in the GDX file will be loaded into the workspace. The GDX name can be specified with or without the `'.gdx'` extension. Note that all of the input arguments must be specified in string form.

As an example of the above syntax, we can read the scalar a0, the vector a1, and the matrix a2 from the file `idx1_.gdx` with the following command:

```
>> irgdx('idx1_','a0','a1','a2');
```

The three symbols are stored as MATLAB variables a0, a1, and a2 in the caller workspace. If we want to read all the symbols from the file `idx1_.gdx`, we simply call `irgdx` with only the file name:

```
>> irgdx('idx1_');
```

Load symbols into output structure

An alternative to storing the results in the MATLAB caller workspace is to return the results in a MATLAB structure. Here, the syntax is:

```
s = irgdx('gdxFileName','sym1','sym2',...);
```

This call will return the indexed parameters named `sym1`, `sym2`, and so on in the structure s, with fields `s.sym1`, `s.sym2`, and so on containing the values of the respective indexed parameters. The input arguments must all be in string form. Only the GDX name is mandatory; if only this argument is specified, all of the indexed parameters stored in the GDX file will be returned in the output structure. The GDX name can be specified with or without the `.gdx` extension.

For example, the following call returns the values of parameter a2 in the structure s:

```
>> s = irgdx('idx1_','a2');
>> s.a2

ans =

    50    50
    50    50
```

To return all the indexed parameters in the fields of the structure `s`, we would simply call `irgdx` with only the GDX name:

```
>> s = irgdx('idx1_');
```

See datalib example `gdxmrw_irgdx01_init`.

MATLAB sparse form

By default, `irgdx` will store results in MATLAB dense form. However, it is possible to use MATLAB sparse form for parameters with dimension less than or equal to 2; MATLAB's sparse storage scheme cannot handle matrices with more than 2 dimensions. Using sparse storage will allow for efficient import of especially large sparse indexed parameters. Note that in this context *sparse* does not refer to data represented in the form [i,j,...,val] but rather to MATLAB's internal sparse storage scheme. To store a particular symbol in MATLAB sparse form, one can simply append ':s' to that symbol's name in the `irgdx` call. For example, the following call will store the parameter a2 into the caller's workspace in sparse form:

```
>> irgdx('idx1_','a2:s');
```

Note that if we want to store *all* the indexed parameters from a specified file in MATLAB sparse form, we would need to specify each symbol individually in order to append the ':s' to each symbol's name.

Renaming

The details of this section are rather involved; it can be skipped without loss of continuity.

While an `irgdx` call will typically store results in the caller workspace or create field names in the output structure using the symbol names from the GDX file, it is possible to rename the outputs in MATLAB without making potentially expensive copies. To rename a symbol, we can take advantage of MATLAB's default copy-on-write mechanism: when an assignment occurs, the values assigned are not actually copied until one of the values (the original or the copy) is actually changed. For example, after executing the following two lines of code, the variable `a2_new` points to the same memory location as a2 and is *not* a separate copy of a2:

```
>> irgdx('idx1_','a2');
>> a2_new = a2;
```

If however the value of `a2_new` were changed after executing the above lines, MATLAB would then need to make a separate copy, at the cost of memory and time. Now, the command

```
>> clear a2;
```

immediately following the above code would complete the rename of `a2` to `a2_new` as desired. Once `a2` is cleared, we can change `a2_new` without needing to make a copy. Similarly, we can do a rename when we have an output structure:

```
>> s = irgdx('idx1_','a2');
>> s.a2_new = s.a2;
>> s = rmfield(s,'a2');
```

To convince ourselves that this code indeed does not require a copy, we can run the commands in the debug output format. That is, immediately after the assignment of `s.a2` to `s.a2_new` in the above code (before using the rmfield function), we can run the following commands:

```
>> format debug;
>> s.a2

ans =


Structure address = 8c1eda0
m = 2
n = 2
pr = 75fa88a0
pi = 0
    50    50
    50    50

>> s.a2_new

ans =


Structure address = 8c1eda0
m = 2
n = 2
pr = 75fa88a0
pi = 0
    50    50
    50    50
```

Note that the pointer pr is the same for both outputs, implying that MATLAB does not make a separate copy prior to a write. A similar test can also be done when parameters are stored in workspace variables rather than a structure.

6.24.2.2 iwgdx

`iwgdx` is a function that creates a GDX file of indexed parameters from MATLAB data. Since `iwgdx` writes only indexed parameters to GDX, the input MATLAB data is simplified: no labels are required or implied. The input matrices can be stored in the usual (dense) scheme or using the MATLAB sparse scheme: `iwgdx` detects and handles the two cases automatically. Note that matrices represented in `[i,j,...,val]` form are not acceptable as `iwgdx` input.

While the `iwgdx` function is patterned after the `save` function in MATLAB, there are important differences. Firstly, if `only` the file name is specified, the `save` function stores all variables in the caller workspace, while `iwgdx` creates an empty GDX file. The behavior of `iwgdx` in this case also differs from that of `irgdx`, which reads all parameters when only the file name is specified. Secondly, `iwgdx` has an additional pass by reference and renaming syntax, in which the parameter string name and values are passed as two consecutive arguments. In Subsection Basic syntax, we describe the basic syntax for `iwgdx`. In Subsection Pass by reference syntax , we introduce the pass by reference and renaming syntax. Finally, in Subsection Dimensionality specification, we describe an optional dimensionality specification that allows symbols to be written with a larger dimensionality than the default.

Basic syntax

The basic syntax for `iwgdx` is:

```
iwgdx('fileName','sym1','sym2',...);
```

The above call will write the values of the variables sym1, sym2, and so on in the MATLAB caller workspace into a GDX file with the name given by the first input argument. The file name can be specified with or without the '.gdx' extension. If a GDX file with the same name already exists, it will be overwritten. Note that a call with a single input argument (the GDX name) will raise a warning and will create an empty GDX file. This contrasts to the MATLAB save function, which writes all of the workspace variables when none are specified. In the basic form described here, all of the `iwgdx` input arguments are strings.

To illustrate, suppose the matrices a2 and a3 exist in the MATLAB caller workspace. We can write the results of these matrices into a file test.gdx with the following call:

```
>> iwgdx('test','a2','a3');
```

If the file test.gdx already exists, it will be overwritten. Specifying only the file name will create an empty GDX file:

```
>> iwgdx('test');
Warning: an empty gdx file was created.
```

Pass by reference syntax

When using the basic `iwgdx` syntax, a copy of the workspace variable to be written to GDX is made as part of the call. This is unavoidable: the MATLAB executable (MEX) interface only provides functions that return *copies* of workspace variables, most likely to avoid overwriting or corrupting data. However, an alternative, more efficient approach is possible by passing the numeric data as an argument. Essentially, the pointer to the data values are passed by reference and no copy is made. This could be especially useful when writing a large amount of data. An additional benefit of this syntax is the ability to write GDX parameters with different names than used in the MATLAB caller workspace. For example:

```
>> iwgdx('test','s1',s1Mat,'s2',s2Mat);
```

In the above command, `s1` and `s2` are strings holding the GDX parameter names, while `s1Mat` and `s2Mat` are matrices containing the values to store. We can also use a mix of the basic syntax with the pass by reference syntax. For example:

```
>> iwgdx('test','anew',a,'b','c',c);
```

This call would save matrix `a` as anew in GDX without making a copy, would internally make a copy of `b` while saving `b` to GDX, and would save `c` to GDX without renaming or making a copy.

We can do a timing test to see the performance difference between the basic syntax and the pass by reference syntax. The following commands create a 5000x5000 dense matrix consisting of random values in the workspace.

```
>> clear;
>> randMat = rand(5000);
```

First, we do a timing test using the basic syntax:

```
>> delete test.gdx; % delete file if it exists
>> tic;
>> iwgdx('test','randMat');
>> t1 = toc

t1 =

    5.3346
```

Now, we do a timing test using the pass by reference syntax:

```
>> delete test.gdx; % delete file if it exists
>> tic;
>> iwgdx('test','randMat',randMat);
>> t2 = toc

t2 =

    5.1636
```

For the above example, the improvement in time performance is about 3%. Of course, the pass by reference call will also use only the half the memory compared to the basic call. We can expect further improvement in performance time and memory use for even larger data.

Dimensionality specification

It is possible to specify the number of dimensions (i.e. dimensionality) for the symbol written to GDX by appending `:n` to the variable/symbol name, where `n` is the desired dimensionality of the GDX symbol. This feature is motivated by two quirks of the MATLAB environment. Firstly, all scalars and vectors are stored in MATLAB as 2-dimensional matrices. For example:

```
>> s0 = 100
>> s1 = [ 100 ]
```

results in two variables with identical storage schemes: 1x1 matrices. In the MATLAB environment this isn't a problem: things behave as you would expect them to, and the `isscalar`, `isvector`, and `ismatrix` calls are available to interrogate MATLAB about how it views variables. However, in GDX there is a distinction between a scalar (something with 0 dimensions), a parameter with 1 dimension and length 1, and a 1x1 parameter. We need to be able to create GDX files containing any or all of these.

Secondly, MATLAB "flattens" all variables by removing trailing dimensions whose size is 1. For example, after executing

```
>> B = ones(2,2,2)
>> C = ones(2,2,2,1,1)
```

`B` and `C` will be identical 3-dimensional MATLAB arrays, yet we need to be able to create a 2x2x2x1x1 indexed parameter in GDX.

By default, `iwgdx` creates GDX parameters that are consistent with the MATLAB view of the data passed in. For example, executing

```
>> d0 = ones(1,1)
>> d1 = ones(3,1)
>> d2 = ones(3,3)
>> iwgdx('dd','d0','d1','d2')
```

results in indexed parameters with 0, 1, and 2 dimensions, respectively, being written to GDX. However, if we want to write d0 as a vector or matrix or higher-dimensional array, we need to use the optional syntax to specify the dimensionality of the resulting symbol in GDX. For example:

```
>> d0 = pi;
>> p1 = d0;
>> d1 = pi*ones(3,1);
>> p2 = d1;
>> d2 = pi*ones(3,3,1)
>> iwgdx('pp','d0','p1:1','d1','p2:2','d2','p3:3',d2)
```

will write the following symbols to GDX:

- The 0-dimensional symbol `d0`. This is the default behavior, consistent with the MATLAB view that `d0` is a scalar.
- The 1-dimensional symbol `p1`. The scalar variable `p1` was promoted to a 1-dimensional symbol by adding an additional singleton dimension.

- The 1-dimensional symbol `d1`. This is the default behavior, consistent with the MATLAB view that `d1` is a vector.
- The 2-dimensional symbol `p2`. The vector variable `p2` was promoted to a 2-dimensional symbol by adding an additional singleton dimension.
- The 2-dimensional symbol `d2`. This is the default behavior, consistent with the MATLAB view that `d2` is a matrix, i.e. the final singleton dimension is just removed.
- The 3-dimensional symbol `p3`. The matrix variable `d2` was promoted to a 3-dimensional symbol by adding an additional singleton dimension. Independently, call-by-reference was used to rename `d2` to `p3`.

Note that the dimensionality specified must always be a promotion, i.e. the dimensionality of the resulting GDX symbol is only increased from what it would be by default.

See datalib example `gdxmrw_iwgdx01_init`.

6.24.2.3 Indexing with labels (UELs)

To this point we have been looking at functions to read (`irgdx`, Section irgdx) and write (`iwgdx`, Section iwgdx) *indexed* parameters from and to GDX. Indexed parameters are convenient since their structure is essentially identical to that of MATLAB matrices. This structure can be encapsulated very simply as the number of dimensions and the size or extent of each dimension. Using indexed parameters, we do not need to be very concerned with the structure of the data since little structure exists. However, there is much more to GDX data than indexed parameters. If we want to read and write GDX data in more generality we will need to understand how labels or strings are used to reference GDX data and how these labels are organized within GDX.

In general, GAMS data is referenced with labels instead of with numbers, so that one references `demand('chicago')` instead of `demand(2)` in a GAMS model. These labels are also called Unique Elements, or UELs. The collection of UELs used in a model or in a GDX file is ordered internally and often referred to as the *universe* of UELs. For efficiency, it is not necessary to use labels internally in GDX or when using the GDX API. Instead, a correspondence or mapping between integers and labels is established initially and/or built up as labels are introduced, so that integers can be used in place of the UELs that reference the data. It is important to keep in mind that the labels are the key thing in referencing GDX data: when integer maps are used the integers are only used to efficiently represent the labels.

A similar scheme mapping integers to labels is used in the `rgdx` and `wgdx` routines for reading and writing general GDX data. In addition to the actual data values being stored, there will be a mapping passed to allow integers to be mapped to labels and vice versa. In some cases, where no mapping is passed and data are written to GDX, a default mapping (`1` to `'1'`, `2` to `'2'`, etc.) may be used. When reading, the default mapping to use when passing back data is the universe mapping from the GDX, but it is possible to apply a filter. A filter can reduce the amount of data returned and also change the order of that data.

6.24.2.4 rgdx

`rgdx` is a MATLAB utility to import data from a GDX file. It takes structural input and returns data back in the form of a structure. This is a very flexible routine as it gives user control over the output data structure. `rgdx` can read a set/parameter/equation/variable from a GDX file and display results in either full/dense or sparse form. A user can also perform a filtered read to read only certain specific elements of a symbol. It can also perform compression to remove extra zeros.

This routine can take up to two arguments. The first argument is a string input containing the GDX file name. It can be with or without the `'.gdx'` file extension. If you call this routine with only the GDX file name as an argument then the `'uels'` field of output structure will be the global UEL of the GDX file and the rest of the fields of the output structure will be NULL. The second argument is a structure input containing information regarding the desired symbol. The syntax for this call looks like this:

```
x = rgdx('fileName', structure);
```

As an example, we read a 3D parameter, `'test3'` from `'sample.gdx'`. Here we display this parameter in full format but without redundant zeros:

```
>> s.name = 'test3';
>> s.form = 'full';
>> s.compress = true;
>> x = rgdx('sample', s)

x =

    name: 'test3'
    type: 'parameter'
     dim: 3
     val: [4x2x2 double]
    form: 'full'
    uels: {{1x4 cell}  {1x2 cell}  {1x2 cell}}

>> x.val

ans(:,:,1) =

     3     4
     4     5
     5     6
     6     7


ans(:,:,2) =

     4     5
     5     6
     6     7
     7     8

>> x.uels{1}

ans =

    '1'    '2'    '3'    '4'

>> x.uels{2}

ans =

    'j1'    'j2'

>> x.uels{3}

ans =

    'k1'    'k2'
```

In the following subsections we will explain the input and output structures. Please note that except for the `'name'` and `'uels'` fields, all other string fields take case insensitive input. All boolean fields can also be entered as string values as well.

Input structure

To read a symbol from a GDX file we just need to know its name in string format. Thus, the only mandatory field of the input structure is 'name'. e.g.

```
>> s.name = 'test3';
```

There are several other optional fields of the input structure that give user more control over the output structure. These optional fields are as follows:

1. form
   This field represents the form of the output data. Output data can be either in `'full'` or `'dense'` form or it can be in `[i, j,.., val]` sparse form. We will label `[i, j,.., val]` as 'sparse'. A user can enter it as string input with value `'full'` or `'sparse'`. e.g.

   ```
   >> s.form = 'full';
   ```

   By default the data will be in 'sparse' format. Note that this sparse format differs from MATLAB's internal sparse storage scheme that we referred to in prior sections for the `irgdx` and `iwgdx` functions.

2. compress
   By default the uels in the output structure will be a global UEL of the GDX file and the `'val'` field data will be indexed to this UEL. The `rgdx` routine allows a user to remove rows and columns with all zeros from the 'val' data matrix and re-indexes the uels accordingly. This is called compression of the data. This can be achieved by setting compress as true in the input structure. Valid values for this field are true and false, either in logical form or in string form. e.g.

   ```
   >> s.compress = 'true';
   ```

   However, we note that compressing the data can be dangerous because the size of the matrix that is read can be incorrect. Essentially, all zero rows and columns are removed, including those that might actually be part of the data values in the symbol matrix.

3. uels
   This input field is used to perform a filtered read, i.e. output data matrix will contain values only corresponding to the entered uels. Filtered read is very useful if user just wants certain specific set of data. Uels should be entered in cell array form. It has to be in 1xN form with each column being a cell array representing the uels for that dimension. Each column can have strings, doubles or combinations of both. It also allows a user to enter double data in shorthand notation or a 1 x N matrix. For example, in the previous example we can perform a filtered read to get data corresponding to only the `'1'`, `'3'` elements of the first index of the parameter `'test3'`. The following function is handy when one needs to generate a UEL listing for the input structure:

   ```
   >> guel = @(s,v) strcat(s,strsplit(num2str(v)));
   ```

   Using the above function, we can create a listing of strings consisting of the string s appended to each number in the array v. Thus, instead of using the command

   ```
   >> s.uels = {{'1','3'},{'j1','j2'},{'k1','k2'}};
   ```

   we can conveniently use the command

   ```
   >> s.uels = {guel('',[1,3]),guel('j',1:2),guel('k',1:2)};
   ```

   The benefit of using the latter command will be more apparent when creating many elements in the listing. Now, as an example, suppose we would like to do a filtered read on the parameter called `test2` in `sample.gdx`. We could use the following commands:

```
>> s.name = 'test3';
>> s.form = 'full';
>> s.compress = false;
>> s.uels = {guel('',[1,3]),guel('j',1:2),guel('k',1:2)};
>> x = rgdx('sample',s)

x =

    name: 'test3'
    type: 'parameter'
     dim: 3
     val: [2x2x2 double]
    form: 'full'
    uels: {{1x2 cell}  {1x2 cell}  {1x2 cell}}

>> x.val

ans(:,:,1) =

     3     4
     5     6


ans(:,:,2) =

     4     5
     6     7
```

Here it should be noted that we turned off compression while performing the filtered read. This is necessary because the filtered read will give data in accordance with the entered uels and the output uels will be the same as the input uels; thus compression is not possible.

4. field
   This field is required when variables or equations are to be read from a GDX file. Sets and parameters in the GDX file do not have any field value but variables and equations have 5 fields namely, level, marginal, lower, upper, and scale. Thus, it may be useful to enter field as an input when reading an equation or a variable. A user can enter it as a string with valid values being `'l/m/up/lo/s'`. e.g.

   ```
   >> s.field = 'm';
   ```

   By default, the output will be the level value of a variable or an equation.

5. ts
   This represents the text string associated with the symbol in the GDX file. If a user sets this field to be `'true'`, then the output structure will have one more string field `'ts'` that contains the text string of the symbol. e.g.

   ```
   >> s.ts = true;
   ```

6. te
   GAMS allows a modeler to enter text elements for a set. Similarly to the `'ts'` field, if a user sets `'te'` to be true in the input structure, then the output structure will contain one more field representing the text elements for that symbol. Please note that text elements only exist for `'sets'`. e.g.

   ```
   >> s.te = true;
   ```

Output Structure

As mentioned earlier, output of the `rgdx` routine will be in structure form. This structure is very similar to the input structure. To get information regarding any symbol, we always need to display its basic characteristics, such as its name, type, value, uels, form, etc. An output structure will always have these fields:

1. name
   It is same as that entered in the input structure name field, i.e., the symbol name in the GDX file.
2. val
   It represents the the value matrix of the symbol. To save MATLAB memory by default it will be in `'sparse'` format. e.g.

```
>> s = rmfield(s, 'form');
>> s

s =
        name: 'test3'
    compress: 0

>> x = rgdx('sample', s)

x =

    name: 'test3'
    type: 'parameter'
     dim: 3
     val: [16x4 double]
    form: 'sparse'
    uels: {{1x8 cell}  {1x8 cell}  {1x8 cell}}
```

   Here `val` is a 16x4 double matrix. As it is a parameter; thus the last column of the sparse matrix will represent the value and the rest (i.e. the first three columns) will represent its index. Please note that in the case of a `'set'`, the number of columns in the sparse matrix will be equal to its dimension, i.e., it will not have a column representing its value. Here, the presence of each row in the output `'val'` field corresponds to the existence of a set element at that index. When a `'full'` matrix output is specified, a 1 represents existence of a set element and a 0 otherwise.
3. form
   It represents the format in which the `'val'` field is being displayed. As mentioned earlier it can be either in `'full'` or `'sparse'` form.
4. type
   While reading a symbol from a GDX file it is often very useful to know its type. The `rgdx` routine is designed to read set, parameter, variable and equation. This field will store this information as a string.
5. uels
   This represents the unique element listing of the requested symbol in the form of a cell array. It is a 1 x N cell array, where N is the dimension of the symbol. Each column of this array consists of string elements. By default, the output uels will be the same as the global uel of the GDX file, but it can be reduced to element specific local uels if compress is set to be true in the input structure. If a user is using a filtered read, i.e. calling `rgdx` with input uels, then the output uels will be essentially the same as the input uels.
6. dim
   It is a scalar value representing the dimension of the symbol.

   Apart from these necessary fields there are a few additional fields as well. They are as follows:

7. field
   If we are reading variables or equations, then it becomes useful to know which field we have read, i.e., `l/m/up/lo/s`. This information is displayed via this field in the form of a string.

8. ts
   It display the explanatory text string associated with the symbol. This field only exists in the output structure if the `'ts'` field is set as `'true'` in the input structure.

9. te
   It is an N dimensional cell array representing the text elements associated with each index of the set. This field only exists in the output structure if the `'te'` field is set as true in the input structure and the symbol is a set.

See datalib example `gdxmrw_rgdx01_init`.

6.24.2.5 wgdx

`wgdx` is a MATLAB routine to create a GDX file containing data from MATLAB. Similar to the `rgdx` routine, it takes a structure input and can write multiple symbols into a single GDX file with one call. The first argument is the file name of the GDX file to be created in string format; it can be with or without the `'.gdx'` file extension. The rest of the arguments are structures, each containing data for different symbols to be written into the GDX file. The syntax for the call is:

```
>> wgdx('fileName', s1, s2 ...);
```

If the GDX file already exists in the MATLAB current directory, `wgdx` will overwrite it; otherwise a new file will be created. After a successful run, it doesn't return anything back into MATLAB. Most of the fields of its input structures are the same as those of the `rgdx` output structure. In the example below, we use the `wgdx` routine to create `foo.gdx` containing a set `'l'` and a parameter `'par'`.

```
>> s.name = 'l';
>> s.uels = {{'i1', 'i2', 'i3'}, {'j1',  'j2'}};
>> c.name = 'par';
>> c.type = 'parameter';
>> c.val = eye(3);
>> c.form = 'full';
>> c.ts = '3 x 3 identity';
>> wgdx('foo', s, c)
```

The equivalent code in GAMS to create the above set `'l'` is:

```
Set
   a / i1*i3 /
   b / j1*j2 /
   l(a,b);

l(a,b) = yes;
```

In the next subsection we will explain the input fields in detail.

Input Structure

The necessary fields in an input structure to represent a symbol are given below:

1. name
   It is a string representing the name of the symbol.

2. val
   It represents the value matrix of the parameter or set. It can be entered in either full or sparse format, whichever is convenient to the user; the corresponding format must be specified in the `'form'` field. By default the value matrix is assumed to be in sparse format.

3. type
   It is a string input to specify the type of the symbol. The `wgdx` routine can write a set or parameter into the GDX file. In the previous example, we didn't specify the type for structure `'s'` because by default it is assumed to be a set.

4. form
   This is a string input representing the format in which the val matrix has been entered. By default it is assumed that the data is specified in sparse format.

5. uels
   Similar to the `rgdx` uels field, this represents the local unique element listing of the symbol in an 1 x N cell array form. Each column of this cell array can contain string or double or both. Again we can make use of a handy function to create a UEL listing as mentioned previously in the section about `rgdx`:

   ```
   >> guel = @(s,v) strcat(s,strsplit(num2str(v)));
   ```

   As an example, we create a set using this function and the `wgdx` routine:

   ```
   >> s.name = 'l';
   >> s.uels = {guel('i',1:20),guel('j',1:40)};
   >> wgdx('foo',s);
   ```

   The above code will create `foo.gdx` that contains a 2 dimensional set `'l'` with set elements for each of the specified uels. If a user would like to create a set with set elements only at specified indices, one could do so by entering the indices in the val field. e.g.

   ```
   >> s.val = [1, 1; 2, 1; 3, 6; 4, 7];
   >> wgdx('foo',s);
   ```

   Now the above would result in a set with only elements existing in the indices specified in the `val` field. If a user enters a structure with only two fields, name and uels, as in the example given in the introduction of this section (structure s), then `wgdx` will create a full set corresponding to the global uels.

   The optional fields are:

6. dim
   This field is useful when a user wants to write a zero dimensional or 1 dimensional data in full format. As every data matrix in MATLAB is at least 2D, it becomes necessary to indicate its dimension for writing purposes.

7. ts
   This is the text string that goes with the symbol. If nothing is entered then `'MATLAB data from GDXMRW'` will be written in the GDX file.

See datalib example `gdxmrw_wgdx01_init`.

6.24.3 Extended use

In this section, we will discuss a few examples of the MATLAB and GAMS interface. We will give a simple example of a nonlinear optimization problem that would benefit from this capability and describe the steps that are needed in order to use our interface in this application.

- Special values
  Following example shows how special values are handled by this interface. It can be seen that `rgdx` can retrieve all these values from GDX file and display them appropriately in MATLAB.

```
>> s.name = 'special';
>> s.form = 'full';
>> s.compress = true;
>> x = rgdx('sample', s)

x =

    name: 'special'
    type: 'parameter'
     dim: 1
     val: [4x1 double]
    form: 'full'
    uels: {{1x4 cell}}

>> x.val

ans =

                -Inf
                 NaN
   3.141592653589793
                 Inf
```

- Variables and Equations
  In an optimization problem, we are not only interested in level value of variables and equations but also in their marginal values, lower and upper bounds. This interface gives its user ability to read any of these values into MATLAB. By default `rgdx` and `gams` routines will read the level value of equations and variables but this can be changed very easily by using `'field'` in input structure. In `gams` call user can also specify this in `'$set matout'` statement. e.g.

```
$set matout "'matsol.gdx', x.m, dual.lo=dl ";
```

  In this case the marginal value of variable `'x'` will be read and lower bound of dual variable will be read and stored in `'dl'`.

- Text string and Text elements
  GAMS allows its user to enter text string and explanatory text elements and all GDX file contain this information as well. Following example shows how to get these text elements in MATLAB.

```
>> s1.name = 'el';
>> s1.te = true;
>> s1.ts = true;
>> s1.compress = true

s1 =

        name: 'el'
          te: 1
```

```
          ts: 1
    compress: 1

>> z = rgdx('sample', s1)

z =

    name: 'el'
    type: 'set'
     dim: 2
     val: [3x2 double]
    form: 'sparse'
    uels: {{1x2 cell}  {1x2 cell}}
      ts: 'This is 2D set with text elements'
      te: {2x2 cell}

>> z.te

ans =

    'element1'    'element2'
    '2.j1'                []

>> z.val

ans =

     1     1
     1     2
     2     1
```

- String elements
  One piece of information that may be needed within MATLAB is the modelstat and solvestat values generated by GAMS for the solves that it performed. This is easy to generate, and an example is given in `do_status.m`. This example is generated by taking the standard gamslib trnsport example, and adding the following lines to the end:

```
$set matout "'matsol.gdx', returnStat, str ";

Set
   stat / modelstat, solvestat      /
   str  / 'grunt', '%system.title%' /;

Parameter returnStat(stat);
returnStat('modelstat') = transport.modelstat;
returnStat('solvestat') = transport.solvestat;
execute_unload %matout%;
```

  Note that the relevant status numbers are stored in GAMS into the parameter returnStat which is then written to matsol.gdx and read back into MATLAB using the `rgdx` call.

```
>> gamso.output = 'std';
>> gamso.form = 'full';
>> gamso.compress = true;
>> s = gams('trnsport')

s =

     1
     1
```

See datalib example gdxmrw_ext01_init.

- Advanced Use: Plotting
  One of the key features of the GAMS/MATLAB interface is the ability to visualize optimization results obtained via GAMS within MATLAB.
  Some simple examples are contained with the program distribution. For example, a simple two dimensional plot with four lines can be carried out as follows. First create the data in GAMS and export it to MATLAB using the gams routine (see APPENDIX C - Calling GAMS model from MATLAB for a detailed description of this routine).
  We make an assumption that the user will write the plotting routines in the MATLAB environment. To create the plot in MATLAB, the following sequence of MATLAB commands should be input (saved as do_plot.m).

```
gamso.output = 'std';
gamso.compress = true;
gamso.form = 'full';
[a,xlabels,legendset,titlestr] = gams('simple');
figure(1)
% Plot out the four lines contained in a;
% format using the third argument
plot(a,'+-');
% only put labels on x axis at 5 year intervals
xtick = 1:5:length(xlabels{1});
xlabels{1} = xlabels{1}(xtick);
set(gca,'XTick',xtick);
set(gca,'XTickLabel',xlabels{1});
% Add title, labels to axes
title(titlestr{1});
xlabel('Year -- time step annual');
ylabel('Value');
% Add a legend, letting MATLAB choose positioning
legend(legendset{1},0);
% match axes to data, add grid lines to plot
axis tight
grid
```

  The data is created using the following gams code.

```
$title Examples for plotting routines via MATLAB

$set matout "'matsol.gdx', a, t, j, sys ";

Set
   sys / '%system.title%' /
   t   / 1990*2030        /
   j   / a, b, c, d       /;

Parameter a(t,j);
a("1990",j) = 1;

loop(t, a(t+1,j) = a(t,j)*(1 + 0.04*uniform(0.2,1.8)););

Parameter year(*);
year(t) = 1989 + ord(t);

* Omit some data in the middle of the graph:
a(t,j)$((year(t) > 1995)*(year(t) <= 2002)) = na;

execute_unload %matout%;
```

The following figure is an example created using this utility (and the MATLAB command print -djpeg simple).

See datalib example GDXMRWPlotting01.

MATLAB supports extensive hard copy output or formats to transfer data to another application. For example, the clipboard can be used to transfer meta files in the PC environment, or encapsulated postscript files can be generated. The help print command in MATLAB details the possibilities on the current computing platform.

Scaling of pictures is also most effectively carried out in the MATLAB environment. The following code is an example of rescaling printed out. Note that the output of this routine is saved as a jpeg file "rescale.jpg".

```
do_plot;
fpunits = get(gcf,'PaperUnits');
set(gcf,'PaperUnits','inches');
figpos = get(gcf,'Position');
pappos = get(gcf,'PaperPosition');
newpappos(1) = 0.25;
newpappos(2) = 0.25;
newpappos(3) = 4.0;
% get the aspect ratio the same on the print out
newpappos(4) = newpappos(3)*figpos(4)/figpos(3);
set(gcf,'PaperPosition',newpappos),
print -djpeg100 rescale.jpg
set(gcf,'PaperPosition',pappos);
set(gcf,'PaperUnits',fpunits);
```

Other examples of uses of the utility outlined in this paper can be found in the "m" files:

```
do_ehl
plotngon
```

that are contained in the distribution.

6.24.4 Acknowledgements

The authors would like to thank Alexander Meeraus and Michael R. Bussieck of GAMS corporation for constructive comments on the design and improvement of this tool. Thanks to Rishabh Jain for work on an earlier version.

6.24.5 APPENDIX A - Configuring GDXMRW

6.24.5.1 Installation

This section describes the installation procedure for all types of machines. The following section describes the testing procedure for verifying a correct installation.

First of all, you need to install both MATLAB and GAMS on your machine. For brevity, we will assume that the GAMS system (installation) directory is (for Windows)

```
C:\GAMS
```

and for non-Windows systems:

```
/usr/local/gams
```

All of the utilities come as a part of the GAMS distribution, so to use them you have only to add the GAMS directory to the MATLAB path. One way to do this is from the MATLAB command prompt, as follows:

```
>> addpath 'C:\GAMS'; savepath;
```

OR this can be done by following these steps:

1. Start MATLAB
2. Click on 'File' tab.
3. Now click on 'Set Path'
4. Click on 'Add Folder'
5. Select GAMS directory and click 'OK'.
6. Save it and then close it.

6.24.5.2 Testing

The GAMS system comes with some tests that you should run to verify the correct configuration and operation of the GDXMRW utilities. In addition, these tests create a log file that can be useful when things don't work as expected. To run the tests, carry out the following steps.

1. Create a directory to run the tests in, e.g.

   ```
   % mkdir \tmp
   ```

2. Extract the test models and supporting files from the GAMS test library into the test directory.

   ```
   % cd \tmp
   % testlib gdxmrw03
   % testlib gdxmrw04
   % testlib gdxmrw05
   % testlib gdxmrw06
   % testlib gdxmrw07
   ```

3. Execute the GAMS files `gdxmrw03`, `gdxmrw04`, and `gdxmrw06`. The files `gdxmrw03` and `gdxmrw04` test that the `rgdx` and `wgdx` routines are working properly, and `gdxmrw06` test `irgdx` and `iwgdx`. In addition to calling MATLAB in batch mode, they verify that the data are read and written as expected and give a clear indication of success or failure.

4. The GAMS file `gdxmrw05` tests the `gams` MATLAB routine. Like the other tests, it can be run in batch mode. You can also run it interactively by starting MATLAB, making `tmp` the current directory, and running the script `'testinst.m'`.

   ```
   >> testinst
   ```

   In addition to messages indicating success or failure, this test produces a log file `testinstlog.txt` that will be useful in troubleshooting a failed test.

5. The GAMS file `gdxmrw07` helps in basic testing if you find that GDXMRW is not functioning at all.To use this, open Matlab, make `tmp` the current directory and run gdxmrwSupport. You can then look over the outputs and identify if and where any errors or failures are occurring. Re-run the commands that result in failures via cut-and-paste, and examine the tests and the output carefully: perhaps a solution will present itself to you. If the issue persists, send the screen output and the resulting log file gdxmrwSupport.txt (located in the created folder tmp_gdxmrw) to GAMS Support (support@gams.com).

6.24.6 APPENDIX B - Utility functions: gdxWhos and gdxInfo

In this section, we'll describe two utilities, `gdxWhos` and `gdxInfo`, provided with GDXMRW that can be called from the MATLAB command prompt. The `gdxWhos` function is patterned loosely after the MATLAB function `whos` used to query `.mat` files. It provides information about symbols in a specified GDX file. The only input argument is the name of a GDX file, which can be with or without the `'.gdx'` extension. With no output argument, `gdxWhos` lists the symbols in the specified GDX file in the MATLAB command prompt. When used with an output argument, information about the symbols in the GDX file is returned as an array of structures. This meta-data can be used in a variety of ways and is especially useful when programming.

An example GDXWhos call without an output argument is shown below:

```
>> gdxWhos('idx1_.gdx');
Symbol info of GDX idx1_.gdx
  Index Type       Dim       NRecs  Name
      1 Parameter    0           1  a0
      2 Parameter    1           3  a1(5)
      3 Parameter    2           4  a2(2,2)
      4 Parameter    3           6  a3(3,5,2)
      5 Parameter   10         256  a10(3,5,2,2,2,2,2,2,2,2)
```

The above output indicates that there are five indexed parameters in the file `'idx1_.gdx'`. The way we can recognize an indexed parameter is by noticing numbered arguments of the parameters under the `'Name'` column. For non-indexed data, the arguments will appear either as the name of a set or by a '$*$'. For example, the first three symbols are non-indexed and the last symbol is indexed in the output from this call:

```
>> gdxWhos('fake.gdx');
Symbol info of GDX fake.gdx
  Index Type       Dim       NRecs  Name
      1 Set          1           3  i(*)
      2 Parameter    1           3  a(i)
      3 Set          1           3  d_i_m__3(*)
      4 Parameter    1           3  aa(3)
```

The GDXInfo function is patterned after the `gdxdump` utility in GAMS. It lists and dumps all of the data values for each symbol in the specified GDX file to the MATLAB console. The only input argument is the name of a GDX file, which can be with or without the `'.gdx'` extension. It accepts no output arguments. It is especially useful to see the entire contents of smaller GDX files. To use the function, we could simply enter a command such as `gdxInfo('idx1_')` in the MATLAB command prompt, and the entire data contents would be displayed in the console. Further information about what is displayed can be found in the GAMS documentation of the `gdxdump` utility.

6.24.7 APPENDIX C - Calling GAMS model from MATLAB

Until now we have discussed the data Import/Export utility between MATLAB and GAMS. In this section, we will discuss a new MATLAB utility `'gams'` that initializes a GAMS model with MATLAB data then executes GAMS on that model and bring the results back into MATLAB. This `'gams'` routine is based on the same design as `rgdx` and `wgdx` but instead it does everything in one call. This routine can take multiple input arguments and can return multiple output arguments. Its standard syntax is as follows:

```
>>  [x1, x2, x3] = gams('model', s1, s2.., c1, c2..);
```

Here note that the first argument of `gams` is the GAMS model name plus any user specific command line settings. If a user wants to solve the given model (in this case found in `qp.gms`) using a different solver then it can be done by adding that solver to the GAMS model name as `"qp nlp=baron"`. This feature allows a user to change the execution time behavior of the model.

The rest of the input arguments of GAMS are structures. Their positioning is not important. These structures are of two kinds; one is similar to the input structure of `wgdx` and the other structure has just two string fields, name and val. This latter structure is used to set or overwrite values in the model using the `"$set"` variables syntax of GAMS. We will explain it in detail a later section.

The first step is to generate a working GAMS model. For example, we can set up a simple model file to solve a quadratic program that minimizes $\frac{1}{2}$ $x^T Qx + c^T x$ subject to $Ax \geq b$ and $x \geq 0$.

The GAMS model for this quadratic problem is as given in the following code.

```
$set matout "'matsol.gdx', x, dual, obj, returnStat ";

Set
   i / 1*2 /
   j / 1*3 /;

Alias (j1,j);

Parameter
   Q(j,j1) / 1  .1  1.0
             2  .2  1.0
             3  .3  1.0 /
   A(i,j)  / 1  .1  1.0
             1  .2  1.0
             1  .3  1.0
             2  .1 -1.0
             2  .3  1.0 /
   b(i)    / 1 1.0
             2 1.0      /
   c(j)    / 1 2.0      /;

Variable obj;

Positive Variable x(j);

Equation cost, dual(i);

cost..    obj =e= 0.5*sum(j, x(j)*sum(j1, Q(j,j1)*x(j1))) + sum(j, c(j)*x(j));

dual(i).. sum(j, A(i,j)*x(j)) =g= b(i);

Model qp / cost, dual /;

$if exist matdata.gms $include matdata.gms

solve qp using nlp minimizing obj;

Set stat / modelStat, solveStat /;

Parameter returnStat(stat);
```

```
returnStat('modelStat') = qp.modelstat;
returnStat('solveStat') = qp.solvestat;

execute_unload %matout%;
```

This GAMS qp model can be executed directly at the command prompt using the following command

```
gams qp (for Unix/Linux)
```

or

```
gams.exe qp (for Windows)
```

or the user can simply hit the run button in the GAMSIDE. The optimal value is 0.5. In order to run the same model within MATLAB and return the solution vector x back into the MATLAB workspace, no change is required to the GAMS file. In MATLAB, all you have to do is to execute the following command:

```
>>  x = gams('qp');
```

This command will first collect the input structure data and create 'matdata.gdx' and 'matdata.gms' that contains include statements for the symbols written in a file matdata.gdx. In the previous example there is no structural input, so an empty 'matdata.gdx' file will be created and 'matdata.gms' will have just have a load statement for the GDX file but no load statements for any symbol. This is done to prevent any undesirable loading of data in the main model if there had already existed a 'matdata.gdx' or 'matdata.gms file'. After creating these two files then the 'gams' routine will execute "gams qp" using a system call. When this model is executed, another file 'matsol.gdx' will be created because of the execute_unload statement in the last line of the model. Here it should be noted that any model that you want to execute using the MATLAB gams routine should contain something like

```
$set matout "'matsol.gdx', x, dual, obj, returnStat ";
```

either as the first line, or somewhere near the start of the model file. This is a standard gams $set statement, setting the value of the local variable 'matout'. The reason to have this statement near the start of the gms file is that the gams routine searches the file from the beginning for "$set matout" in the gms file. As these files can be very large, it is wise to have this statement near the start of the file. In this statement 'fileName' is the gdx file name that will be created containing symbols 'x1', 'x2', etc. These symbols can then be exported to MATLAB. The last line of the model should always be

```
execute_unload %matout%;
```

The purpose of setting the first and last line of the model in this manner is to specify what data the user wants to export to MATLAB in a "header" of the model. As MATLAB does not give any information about the output arguments except the number of expected arguments, we have to specify what data to export to MATLAB in the GAMS model with minimum modification to the existing model. In the previous example, there is only one output argument, thus the gams routine will get data for its first element from the output gdx file and store it in the MATLAB output argument.

If there is more than one output argument, e.g.,

```
>>  [x, u] = gams('qp');
```

then the gams routine will read the output gdx file and store its first element information of the GDX file as the first output argument of MATLAB, i.e. 'x', and the second element information of the GDX file in the second output argument of MATLAB, i.e. 'u' and so on. If the number of MATLAB output arguments is greater than the number of elements in the GDX file then gams will throw an error.

See datalib example gdxmrw_qp4.

6.24.7.1 Input Structure

As mentioned earlier, the gams routine takes input arguments in structured form. It allows two different types of structure input. One contains the symbol data similar to the wgdx input structure, to be exported to the GDX file. The other structure will just have two string fields 'name' and 'value'. Example use:

```
>> s.name = 'Q';
>> s.val = eye(3);
>> s.form = 'full';
>> m = struct('name','m','val','2');
>> [x] = gams('qpmcp',s, m);
```

In this example both 's' and 'm' are structures but 'm' has only two fields and both are strings. The gams routine will use the 's' structure to create a 'matdata.gdx' file and 'm' to modify the execution command line to include "--m=2" at the end i.e. a command that when executed will be "gams qpmcp --m=2"

The structure 's' is the same as the input structure for wgdx but with two important differences. Firstly, it can be seen in the above example that 's' doesn't have a 'type' field. In wgdx we assume the type to be 'set' by default, but in the gams routine the type is assumed to be 'parameter' by default. The second change is an optional additional input field (in addition to those given in Section Input Structure) called "load".

- load
  It is a string input representing how the corresponding data will be loaded into the GAMS program. Depending on the value of the global option "gamso.input" (see next section) the input data will be read into GAMS in different ways. Suppose the input structure 's' has a 'name' field called 'foo'. By default (where gamso.input = 'compile'), the file matdata.gms will

  ```
  $loadR foo
  ```

  The GAMS parameter (or set) foo will be replaced by the data that is in the 'matdata.gdx' container called 'foo'. If the data has been initialized before in the model, this will replace that initial data with the new data from 'matdata.gdx'. The option can also be explicitly set using

  ```
  s.load = 'replace'
  ```

  There are two other compile time load options, namely 'initialize' and 'merge'. The first is only valid if the parameter values have not been initialized in the GAMS file, otherwise an error is thrown. It uses the GAMS syntax

  ```
  $load foo
  ```

  The merge option is valid when the GAMS file being run has already initialized the parameter values. The new values in the MATLAB structure 's' are merged into the parameter simply overwriting existing values with the new values given. Explicitly, the 'matdata.gms' file contains the statement

  ```
  $loadM foo
  ```

  to direct GAMS accordingly.
  Finally, if gamso.input = 'exec', the loading will occur at execution time. In this case, s.load = 'initialize' is not a valid input, the default setting is s.load = 'replace' which carries out

  ```
  execute_load "matdata.gdx" foo
  ```

  and the alternative setting s.load = 'merge' carries out

  ```
  execute_loadpoint "matdata.gdx" foo
  ```

  In this way, the data is loaded at execution time and performs an appropriate replace or merge.

6.24.7.2 Global input to change default behavior

Until now we have seen how to specify different input to the gams routine and in this section we will see how to change the default behavior of a gams call. This can be done by creating a structure 'gamso' in the current workspace and adding different fields to that structure. There are currently nine fields that can be set in that structure that affect the behavior of the program. Except the uels field, all other string fields take case insensitive data. These are as follows:

- gamso.output
  By default, output of the gams routine will be in structure form but it might be the case that a user is only interested in the data matrix, i.e., the val field of that structure. This can be done by setting gamso.output as 'std'. This will give only the value matrix as output. If this is not set to 'std' then output will be in the structure form described in the wgdx section.

  ```
  >> gamso.output = 'Std';
  >> x = gams('qp nlp=baron')

  x =
      0.5000
  ```

- gamso.input
  By default, the interface updates data at compile time. Thus, if execution time updates are made to the parameters before the line '$include matdata.gms' these may override the data that is provided in 'matdata.gms' (i.e. from the command line). This may not be desirable. If you wish to perform execution time updates to the data, you should set gamso.input to 'exec'.

- gamso.write_data
  If this is set to 'no', then all parameters on the call to gams are ignored, except the program name. This is useful for dealing with large datasets. Consider the following invocation:

  ```
  x = gams('largedata','A');
  y = gams('resolve','A');
  ```

  The first call generates a file 'matdata.gms' containing the elements of the matrix A for use in the largedata.gms program. The second call rewrites a new 'matdata.gms' file that again contains A. If we wish to save writing out A the second time we can use the following invocation:

  ```
  x = gams('largedata','A');
  gamso.write_data = 'no';
  y = gams('resolve','A');
  clear gamso;
  ```

  or the equivalent invocation:

  ```
  x = gams('largedata','A');
  gamso.write_data = 'no';
  y = gams('resolve');
  clear gamso;
  ```

- gamso.show
  This is only relevant on a Windows platform. This controls how the 'command box' that runs GAMS appears on the desktop. The three possible values are:
  - 'minimized' (default): The command prompt appears iconified on the taskbar.
  - 'invisible' : No command prompt is seen.
  - 'normal' : The command prompt appears on the desktop and focus is shifted to this box.

- `gamso.path`
  This option is used to specify fully qualified path for the gams executable. This is very useful if you have multiple versions of GAMS installed on your system and want to make sure which version you are running for the `gams` call. e.g.

  ```
  >> gamso.path = 'C:\Program Files\GAMS23.4GAMS.exe';
  ```

The output of `gams` is similar to `rgdx` but unlike the `rgdx gams` routine it doesn't take input specific to a particular symbol. Thus it becomes important to implement a way to change the default behavior of the output. This can be achieved by adding following field to the global structure `'gamso'`. All these fields behave similar to that described in `rgdx` and take the same input as of `rgdx`.

- `gamso.compress`
- `gamso.form`
- `gamso.uels`
- `gamso.field`

This is a global option however.

6.25 GDXRename

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This renames the labels in the GDX file using a two dimensional `mapSet`. The renaming of the labels only affects the string stored for each label, and does not change the data order for the symbols in the GDX file. Because no data is changing in the GDX file, only the strings for the labels are changed and applied to the GDX file directly and no new GDX file is written. This replaces the labels in the GDX file by the ones in the `mapSet`.

6.25.1 Usage

Command line:

```
gamstool [gdxservice.]GDXRename GDXFile mapSet [reverse=0|1]
```

Compile time:

```
$callTool [gdxservice.]GDXRename [-reverse=0|1] GDXFile mapSet
```

Execution time:

```
executeTool '[gdxservice.]GDXRename [-reverse=0|1] GDXFile mapSet';
```

Where:

Argument	Description
GDXFile	Name of GDX file.
mapSet	Set of labels to be used for renaming mapSet(*,*).

The following named parameter is available:

Argument	Description
reverse=0 or 1	Determines if mapSet with record a.b leads to a replace of a by b or to a replace of b by a (default: 0).

6.25.2 Example

```
Set c / r, g, b, y /;
Parameter A(c) / r 1, g 2, b 3, y 4 /;
$gdxOut color.gdx
$unload c A
$gdxOut

Set map(c,*) / r.red, g.green, b.blue, y.yellow /;
$callTool gdxservice.GDXRename color.gdx map
```

The complete example is also part of the GAMS Test Library, see model [**gdxrename1**] for reference.

6.26 GDXRRW

Interfacing GAMS and R.

Author

Michael C. Ferris [(ferris@cs.wisc.edu)] (ferris@cs.wisc.edu), Computer Sciences Department, University of Wisconsin - Madison, 1210 West Dayton Street, Madison, Wisconsin 53706

Steven Dirkse (sdirkse@gams.com) GAMS Development Corp.

Attention

GDXRRW is deprecated (see GAMS 41 GDXRRW release notes). Please use GAMS Transfer R instead and contact support with any issues about the transition.

GDXRRW is a suite of utilities to import/export data between GAMS and R (both of which the user is assumed to have already) and to call GAMS conveniently from R. The software gives R users the ability to use all the optimization capabilities of GAMS, and allows visualization and other operations on GAMS data directly within R.

The GDXRRW tool is unique among the GDX interface utilities in that it is an R extension made available as an R package. As such, it is run as part of an R session or script, not as part of a GAMS run, and it follows the usual R package conventions.

1. **Installation**: The R software is designed to be easily extended. Thousands of extension packages are freely and conveniently available online and can be installed easily using a simple, standard procedure. The GDXRRW package is one of these. The latest version is available on GitHub in both source and binary form, along with a FAQ list, some hints and tips on common problems and solutions, and other helpful content. For convenience, the source and binary packages are also available in the gdxrrw directory of the GAMS distribution.

2. **Documentation**: The GDXRRW documentation is installed as part of the GDXRRW package and is available from within R in the usual way. See the GDXRRW GitHub page for details and hints.

3. **Examples, tests, and data**: Like many R packages, GDXRRW comes with examples and data that help a user get started using the package. The R help system and the GDXRRW GitHub provide pointers to these. GDXRRW GitHub page also provides hints on finding and running the thousands of lines of tests that come with the GDXRRW package.

6.27 GDXVIEWER

6.27.1 Overview

GDXVIEWER is a tool to view and convert data contained in GDX files.

Attention

GDXVIEWER is deprecated (see GAMS 42 GAMS IDE and GDXVIEWER release notes).

Besides inspecting a GDX file, gdxviewer allows you to export to a large number of data formats, including ASCII text, CSV, HTML, XML, database, and spreadsheet formats.

This tool is designed as an interactive Windows program, but it can also be operated through command line parameters.

6.27.2 Requirements

GDXVIEWER runs only on PC's running Windows (95/98/NT/XP).The DLL **GDXCCLIB.DLL** needs to be in the same location as **GDXVIEWER.EXE**. If XLS files are saved, MS Excel needs to be present. If MDB database files are saved, MS Access needs to be present.

If GDXCCLIB.DLL is not found in the same directory as the executable gdxviewer.exe, the following window will be shown:

A simple way to make sure that GDXVIEWER has access to the GDXCCLIB.DLL dynamic load library is to place **gdxviewer.exe** in the GAMS system directory, e.g. c:\program files\GAMS21.3.

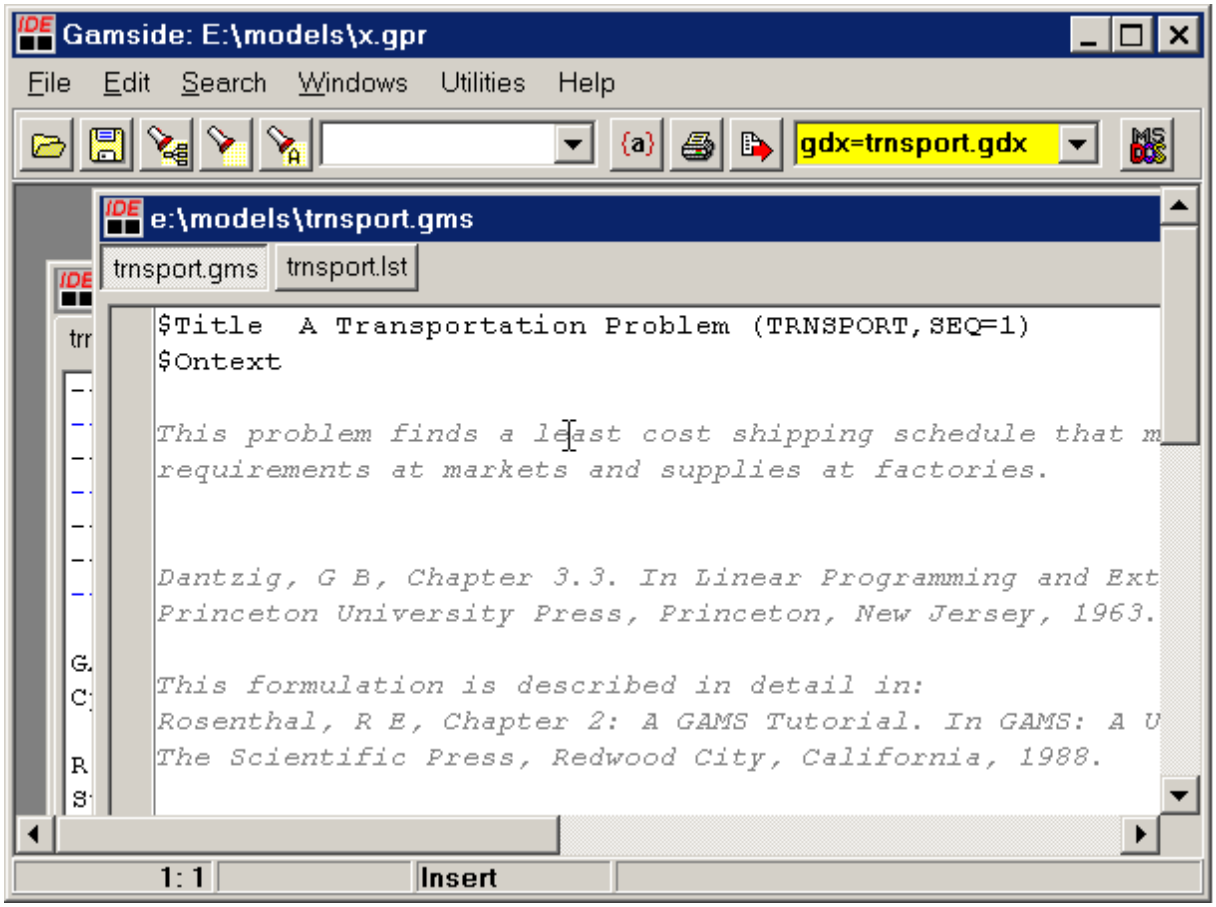
6.27.3 Creating GDX files

GDX files are binary data files. They can contain sets, parameters (including scalars), equations and variables. These files can be generated by a number of tools: by GAMS itself, by utilities such as MDB2GMS, SQL2GMS, GDXXRW.

To save all data from a GAMS model into a GDX file you can use the **GDX=fln** command line parameter:

```
C:\> gams trnsport.gms GDX=trnsport.gdx
```

From the IDE you can specify the command line parameter `GDX=trnsport.gdx` in the parameter edit box:



To selectively place identifiers in a GDX file you can use the **execute_unload** statement:

```
Model transport / all /;
solve transport using lp minimizing z;

display x.l, x.m;

execute_unload 'results.gdx', i, j, x;
```

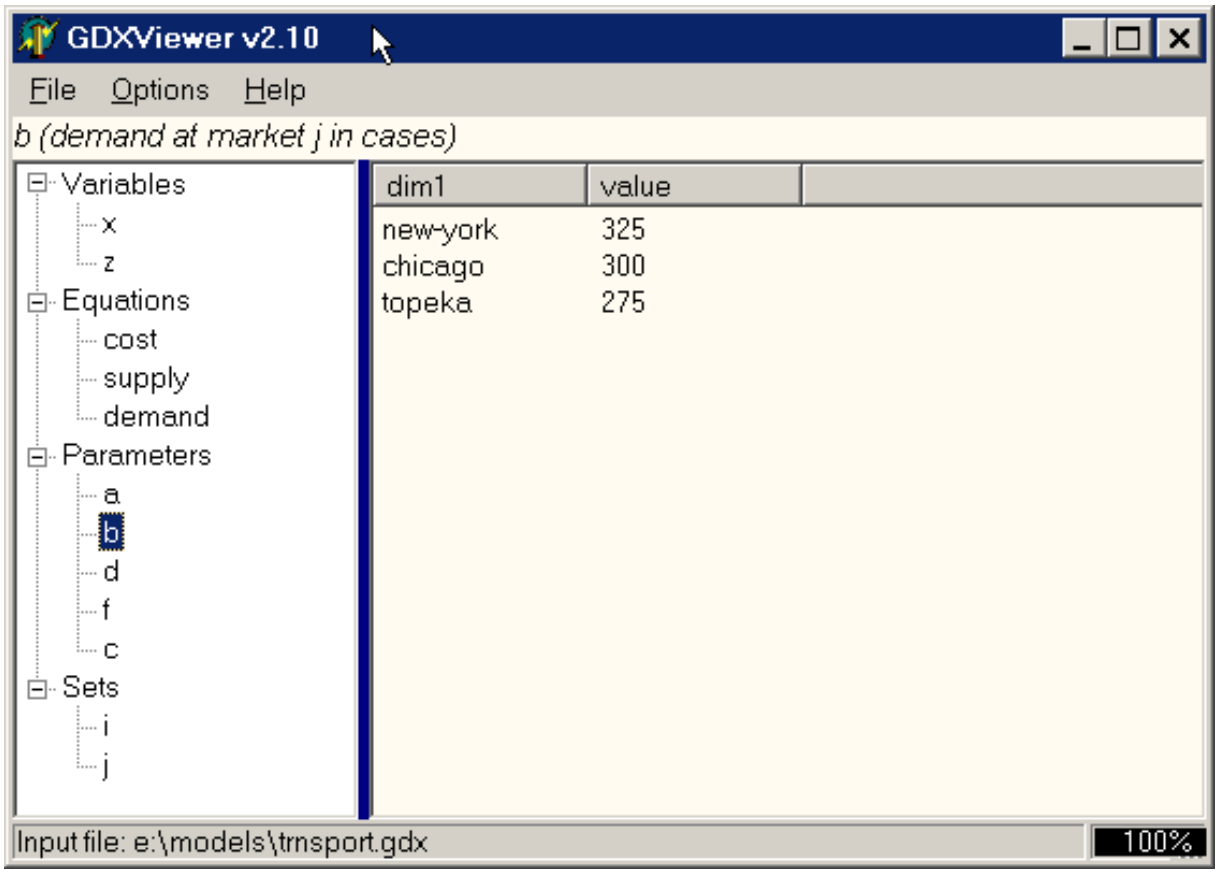
In this example the sets **i** and **j** and the variable **x** are saved to the GDX file **results.gdx**.

Other ways to create GDX files include:

- The MDB2GMS tool can be used to convert data stored in MS Access tables to GDX files
- The SQL2GMS tool can read data from virtually any SQL database (including any ODBC accessible database) and can create GDX files.
- The tool GDXXRW allows data from an Excel spreadsheet to be stored in a GDX file.
- $GDXOUT allows you to write data to a GDX file during GAMS compile time. This is not as useful as execute_unload but may have its use in special cases.
- You can write your own program to write a GDX file. There is an API and bindings for different languages such as Delphi, Kylix, VB6, VBA, VB.NET, C/C++, C#, Java, Fortran.

6.27.4 Viewing GDX files

After loading a GDX file in **gdxviewer** the content of the file is displayed in list view. The left-hand side of the window shows the index of the GDX file organized in a tree structure. When clicking on an identifier, the right-hand-side will display the actual data for the identifier.



When variables are shown, more information is available, such as bounds (lower and upper bounds) and marginals.

The GDX file can be loaded interactively using the `File|Open` menu, or it can be launched from the command line:

```
C:\> gdxviewer e:\models\trnsport.gdx
```

The command line specification can also be used to launch gdxviewer from within a GAMS model as in:

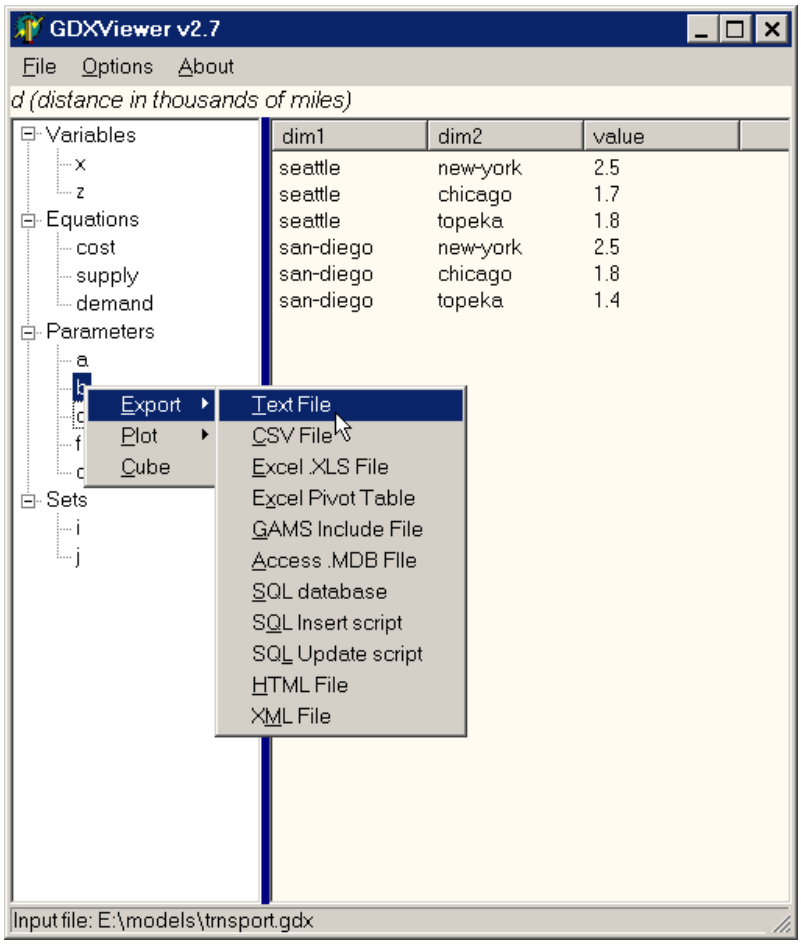
```
Model transport / all /;
solve transport using lp minimizing z;

display x.l, x.m;

execute_unload 'results.gdx', i, j, x;
execute '=gdxviewer results.gdx';
```

In this case **gdxviewer.exe** was located in the GAMS system directory, such that **execute** had no problems in finding it.

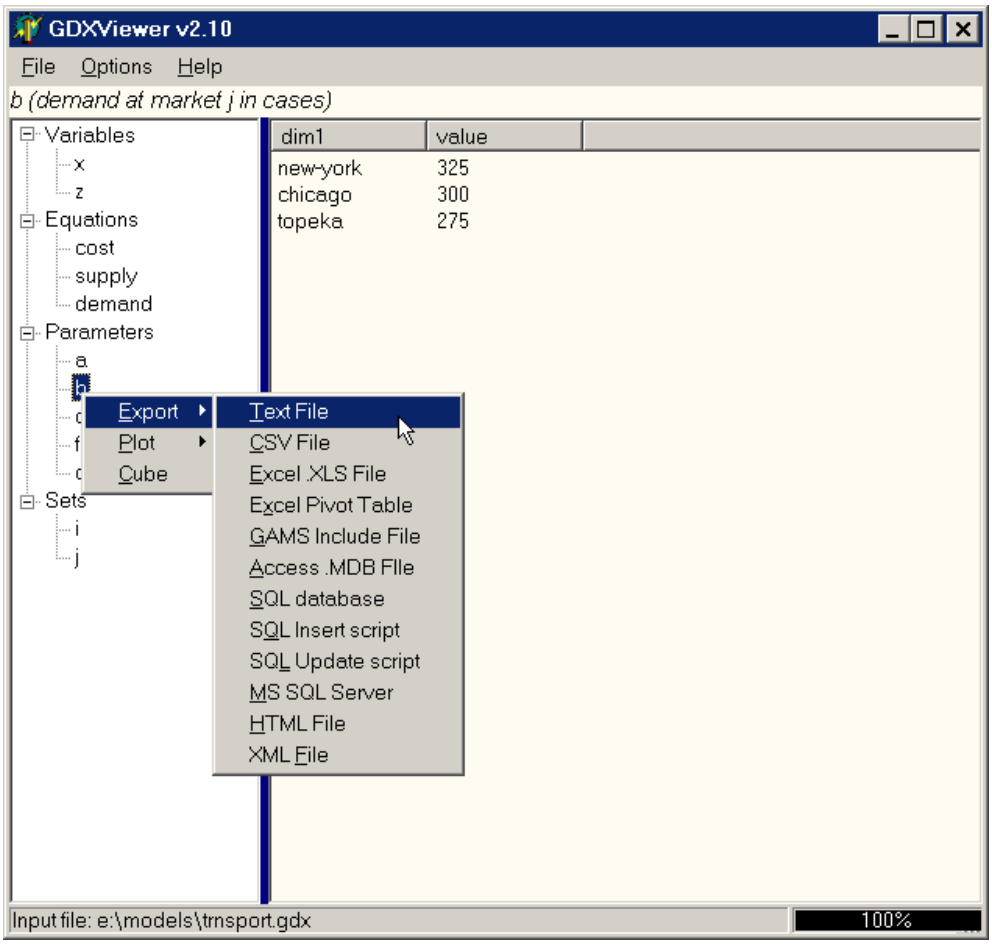
Note: there are alternative tools to view GDX files. The GAMS IDE has a built-in GDX file viewer (use `File|Open`) and there is a command line utility called GDXDUMP.

6.27.5 Exporting an identifier

When the right mouse button is clicked on an identifier a pop-up menu is presented that allows you to export an identifier to a number of target formats.



The same operation can be invoked from the `File|Export` menu:

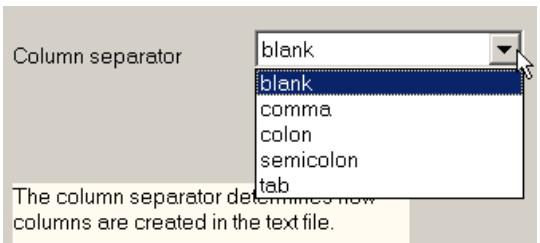


6.27.6 Exporting to a Text File

The text file export facility (`File`|`Export`|`Text File`) will write a GAMS identifier to a standard ASCII text file. Such a text file can look like:

```
seattle new-york 2.5
seattle chicago 1.7
seattle topeka 1.8
san-diego new-york 2.5
san-diego chicago 1.8
san-diego topeka 1.4
```

The separator symbol can be set using the menu `Options`|`Configuration`|`Text File`:



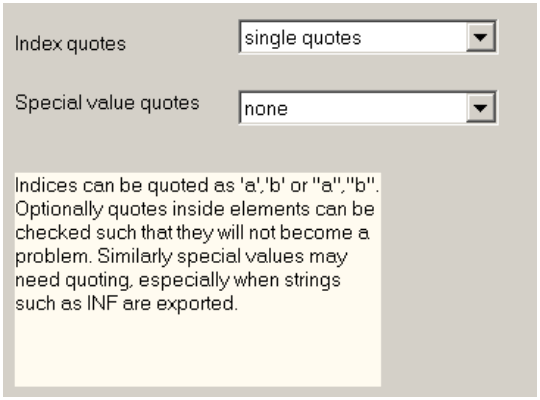
Other options that involve the format of the text file being written are: Options|Configuration|Export and Options|Configuration|Special Values. Currently there are no facilities to write fixed format text files. If you need to write fixed format text files you can use the GAMS PUT statement.

6.27.7 Exporting to a CSV files

Comma-separated Values (`File`|`Export`|`CSV File`) is a popular format to exchange data between applications. An example of such a file is:

```
'new-york',325
'chicago',300
'topeka',275
```

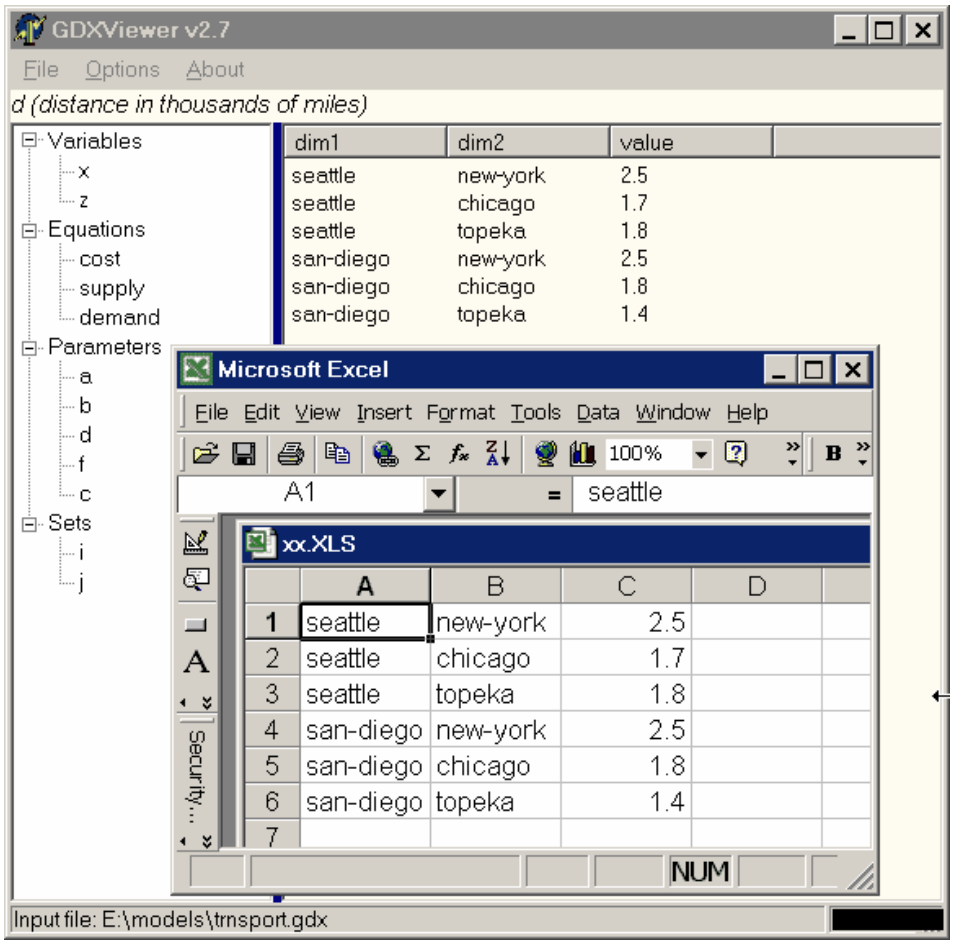
Strings are surrounded by quotes and each field is separated by a comma. The precise format can be specified using the menu `Options`|`Configuration`|`CSV File`:



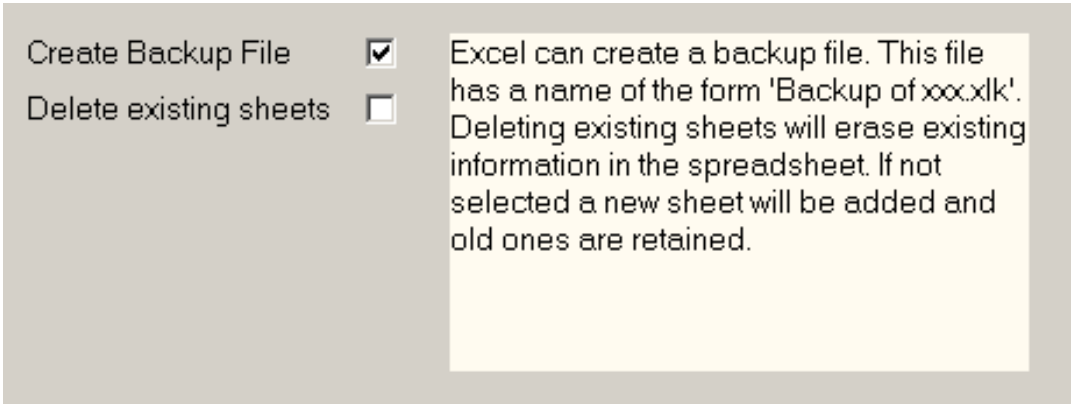
Other options that involve the format of the CSV file being written are: Options|Configuration|Export and Options|Configuration|Special Values.

6.27.8 Exporting to an XLS file

A GAMS identifier can be exported directly to an MS Excel spreadsheet using `File`|`Export`|`Excel XLS File`:



There are a few options available for this operation. Under `Options|Configuration|Excel` the following settings can be changed:

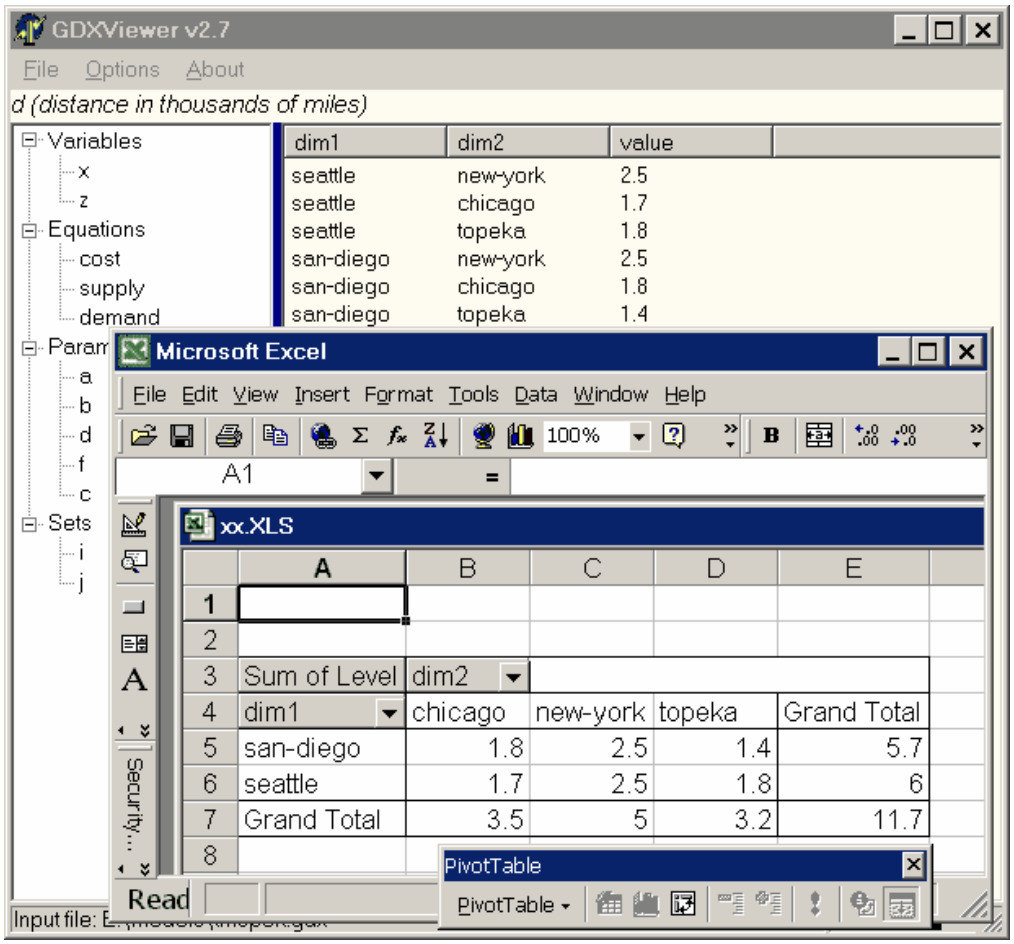


Other options that involve the format of the CSV file being written are: Options|Configuration|Export and Options|Configuration|Special Values. Exporting to Excel is only available if you have Microsoft Excel installed on your machine.

Note: We can write all symbols in the gdx file to Excel by specifying `ID=*` on the command line.

6.27.9 Exporting to an XLS Pivot Table

We can export to an XLS file and create a Pivot Table automatically (`File|Export|Excel Pivot Table`):



Pivot tables are a very convenient way to analyze multi-dimensional data.

The following options are available: Options|Configuration|Excel, Options|Configuration|Export and Options|Configuration|Special Values.

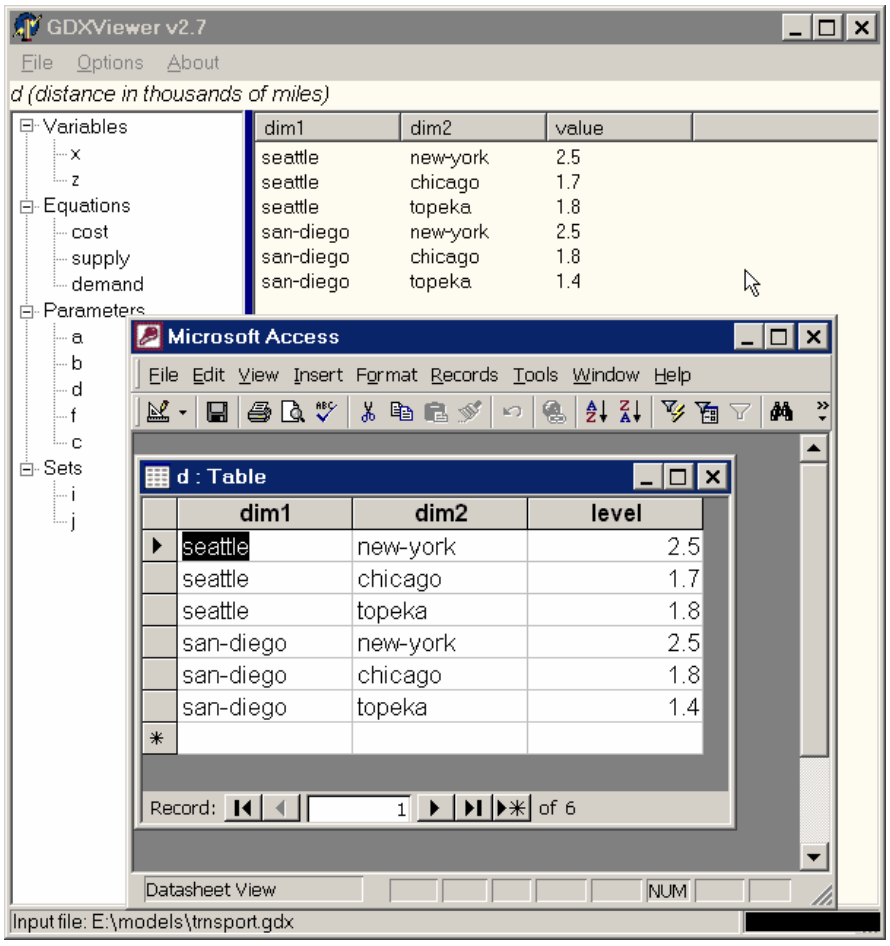
6.27.10 Exporting to a GAMS Include Files

The option `File|Export|GAMS` Include file will export an identifier to a GAMS include file format. An example of such an exported include file can look like:

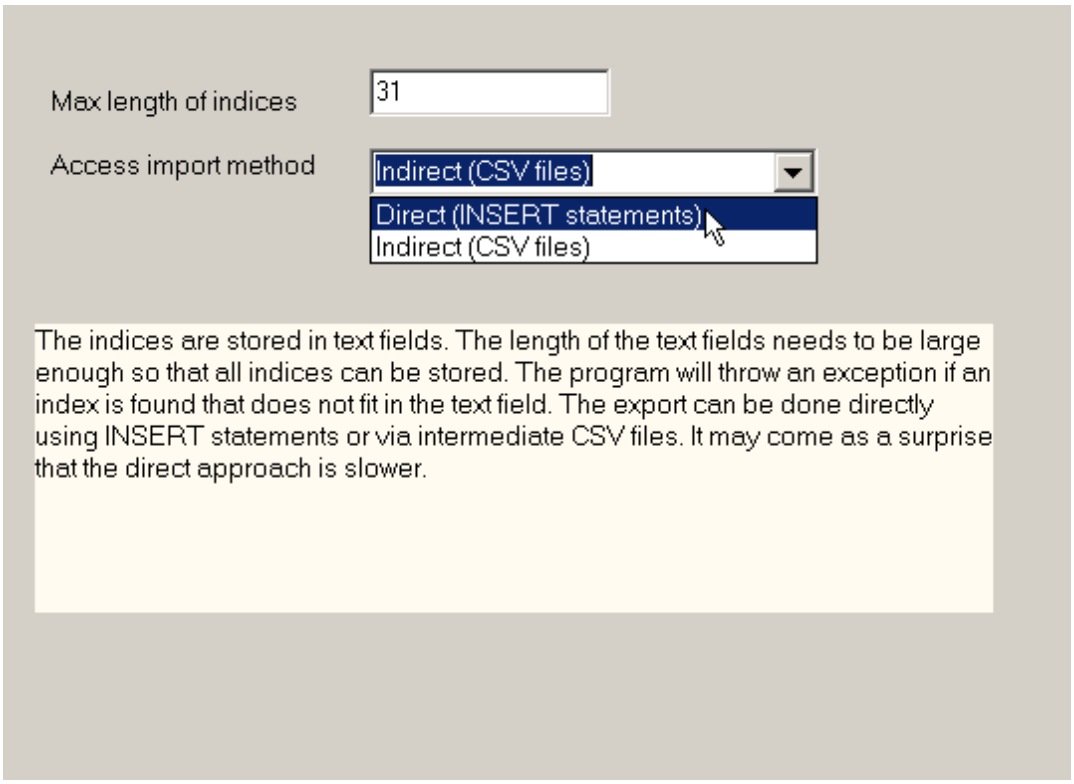
```
Parameter d 'distance in thousands of miles'
            / seattle.new-york 2.5, san-diego.new-york 2.5
              seattle.chicago  1.7, san-diego.chicago  1.8
              seattle.topeka   1.8, san-diego.topeka   1.4 /;
```

6.27.11 Exporting to an Access Tables

GDXVIEWER can export data directly to a table in an Access database using `File|Export|Access (MDB or ACCDB) File`. The name of the table will be the name of the parameter. If the table already exists, GDXVIEWER will try to create a new table with a slightly different name (e.g. d2, d3,...).



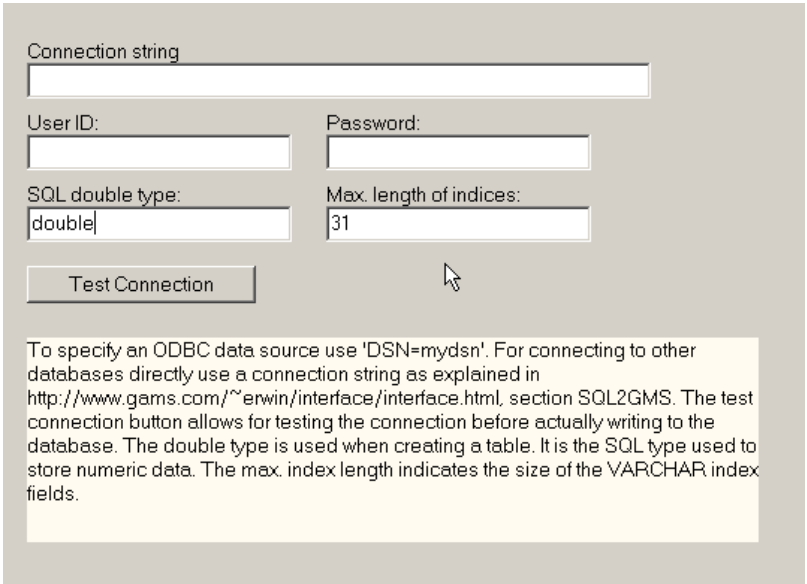
An option `Options|Configuration|Access` allows you to set the length of the text fields where the GAMS indices are stored. This length is used when creating the table.



A feature added in version 2.9 is the possibility to use intermediate CSV (comma separated value) files instead of using direct SQL INSERT statements. The CSV files can be read into Access using a bulk operation and is therefore faster for large datasets. When using CSV files make sure double quotes are used (if single quotes are used they will become part of the data). The temporary CSV files will be written to the Windows TEMP directory (e.g. `C:\WINDOWS\TEMP`). When the import is done, these scratch files will be removed automatically. If you want to look at the CSV files that are being fed into Access, export the data to a CSV file.

6.27.12 Exporting to an SQL Table

It is possible to export data to SQL databases through ADO which includes all databases accessible through ODBC. The configuration information can be specified in `Options`|`Configuration`|`SQL Database`.



The **Test Connection** button will allow you to check the configuration and see if the database can be connected to.

The SQL data for double precision number is no always the same for each database. E.g. for MS Access you can use **double** while for MS SQL server you can use **float**.

When exporting data a new table is created with the name of the identifier. If such a table already exists, names like **name2**, **name3**, are tried.

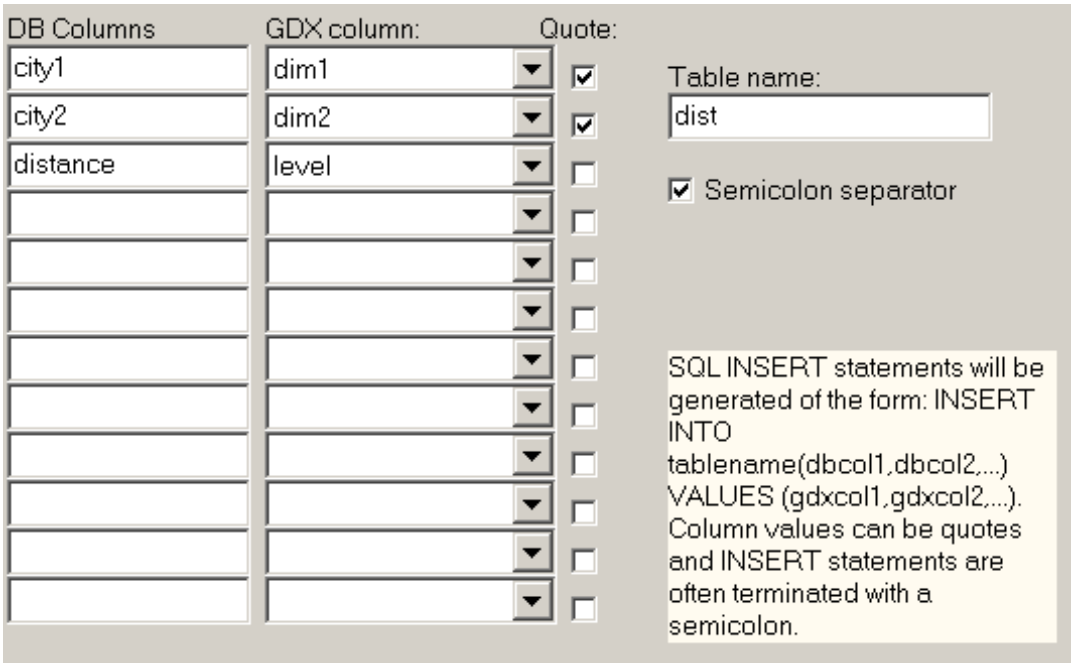
6.27.13 Exporting to MS SQL Server

We can export to Microsoft SQL Server through the standard SQL export facility. However a special facility called BULK INSERT is only available through the specialized SQL Server export tool. BULK INSERT writes a TAB delimited text file to the Windows TEMP directory and subsequently calls BULK INSERT to load that file. This way is often much faster that using individual INSERT statements for each record.

6.27.14 Exporting to SQL Insert script

An SQL script with INSERT statements like:

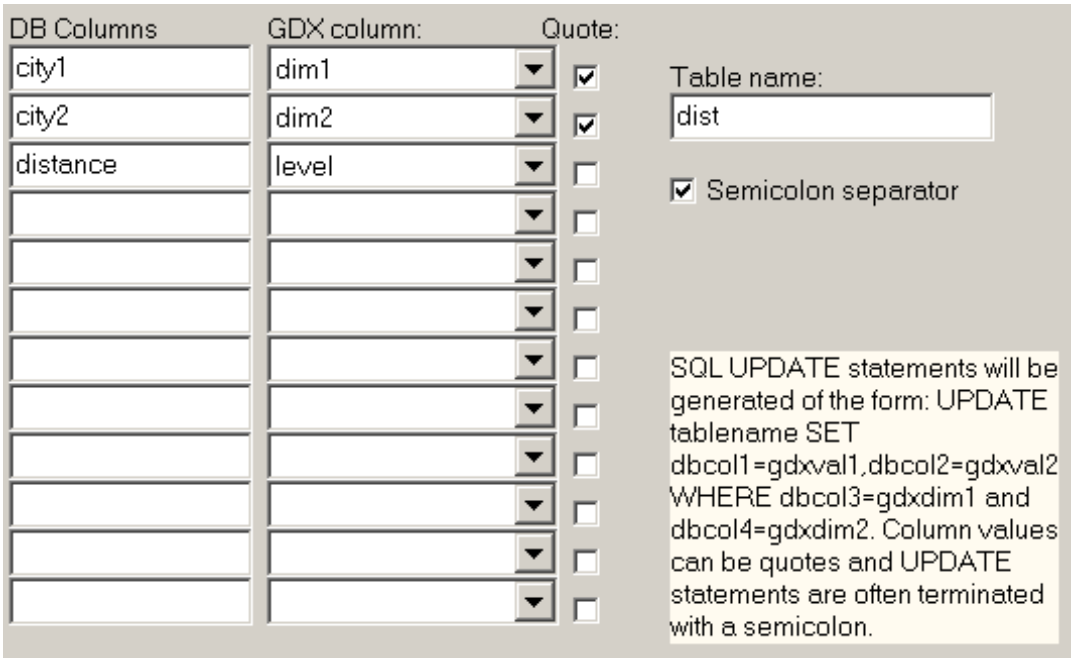
can be generated with `File`|`Export`|`SQL Insert script`. The following settings in `Options`|`Configuration`|`SQL Insert` were used:



6.27.15 Exporting to SQL Update script

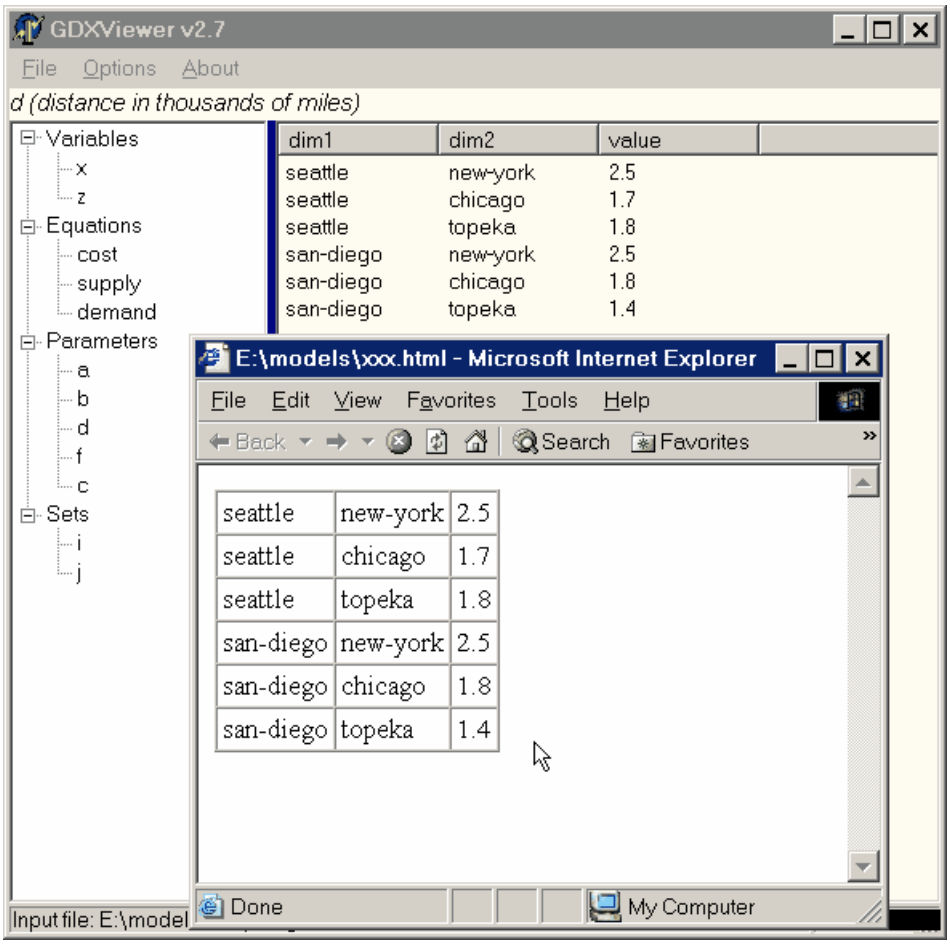
An SQL script with UPDATE statements like:

can be generated with `File`|`Export`|`SQL Update script`. The following settings in `Options`|`Configuration`|`SQL Update` were used:

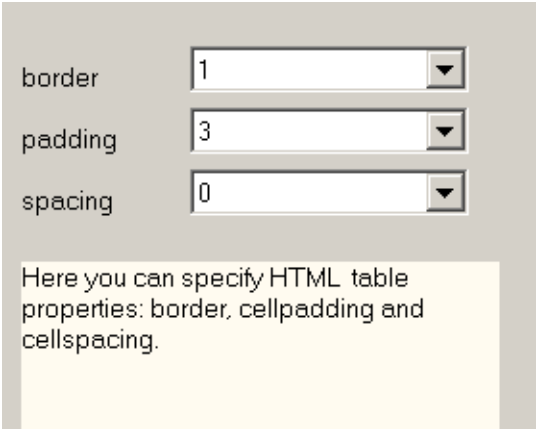


6.27.16 Exporting HTML

GDXVIEWER can write an identifier to an HTML file using `File`|`Export`|`HTML` File.

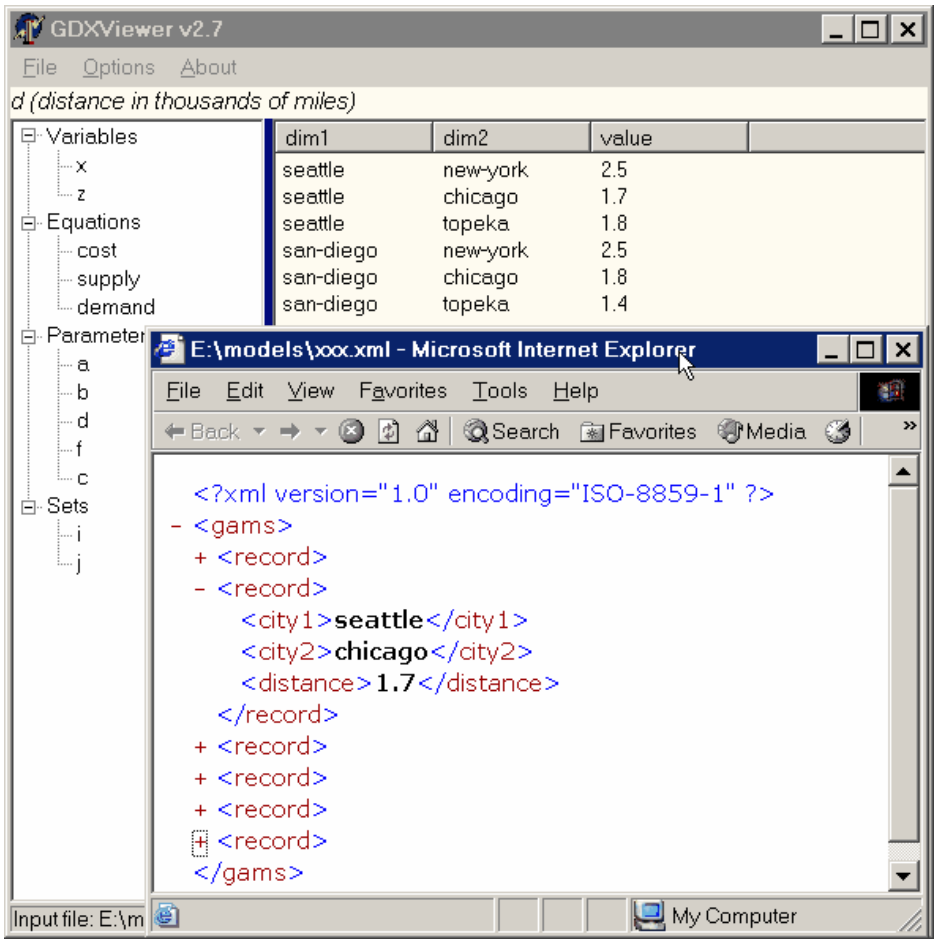


The options relevant to this format are specified in `Options`|`Configuration`|`HTML`.

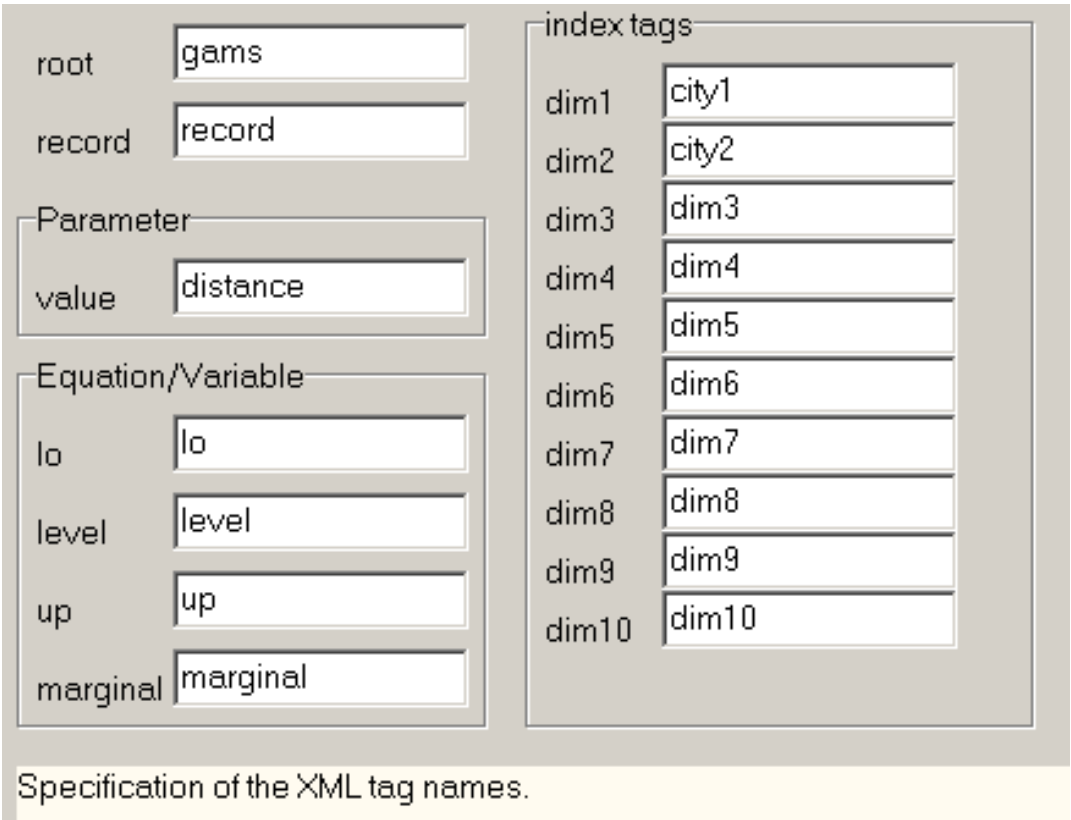


6.27.17 Exporting XML

GDXVIEWER can write an identifier to an XML file using `File`|`Export`|`XML File`.



The XML tags can be specified in `Options`|`Configuration`|`XML`.



6.27.18 Exporting fields

The menu `Options`|`Configuration`|`Export` allows you to set which fields are exported.

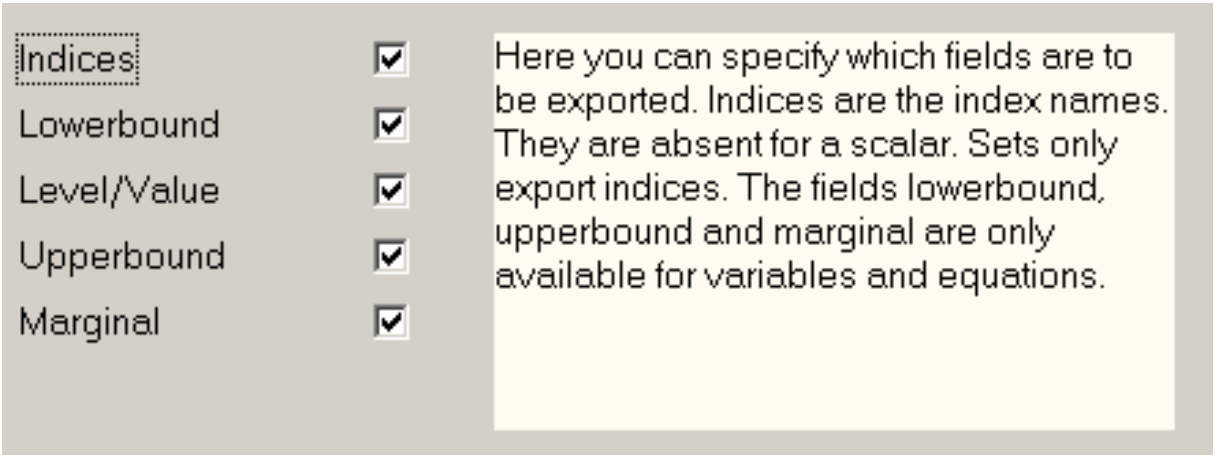


Figure 6.53 400px

The following table gives the possibilities for exports:

	set	scalar	parameter	variable	equation
Indices	+		+	+	+
Lower bound				+	+
Level/Value		+	+	+	+
Upper bound				+	+
Marginal				+	+

6.27.19 Special Values

GAMS data can assume so called special values: `-INF`, `+INF`, `EPS`, `NA`, and `UNDF`. The meaning of these special values is as follows:

Value	Description
-INF	Minus infinity. Mostly used for non-binding lowerbounds.
+INF	Plus infinity. Mostly used for non-binding upperbounds.
EPS	Mostly used for marginals where it can indicate non-basic but numerically zero.
NA	Not available. Not often used.
UNDF	Undefined. Not often used.

When exporting GAMS identifiers we need to map such values to strings that the receiving program can understand. E.g. we could map `{INF` to `{1.0e10` and `+INF` to `+1.0e10`. A good choice for `EPS` would be `0.0`.

The mapping can be specified in `Options`|`Configuration`|`Special Values`:

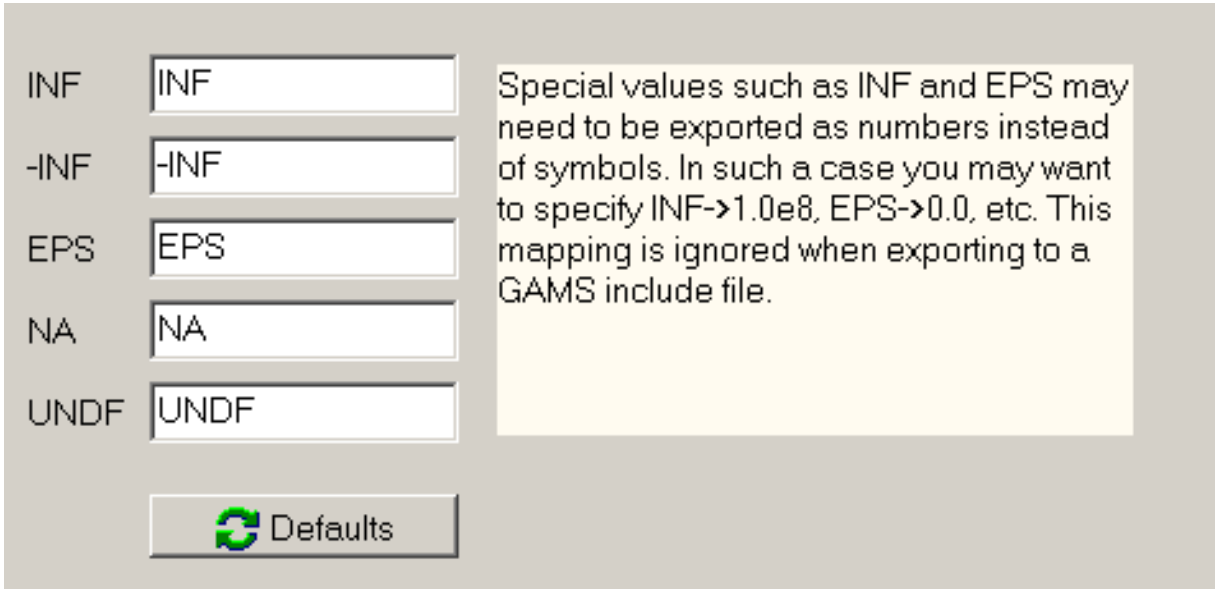
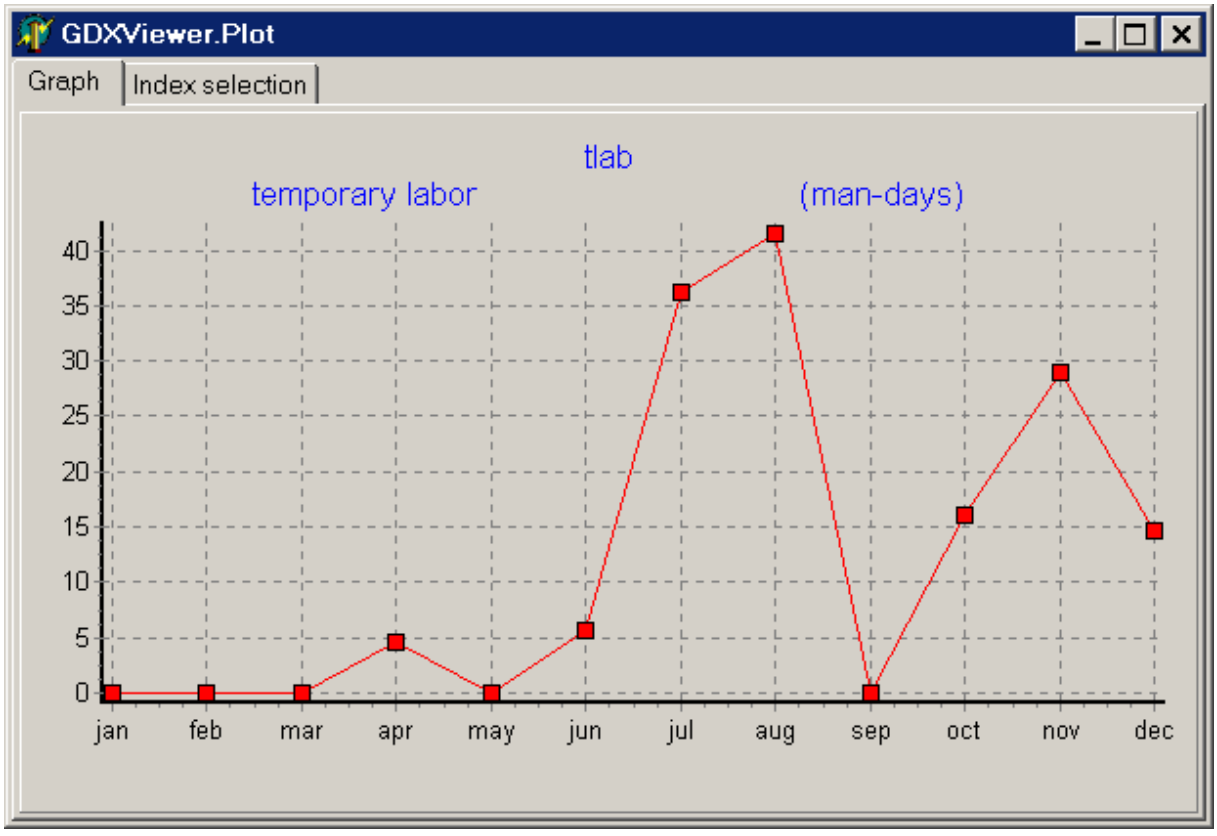


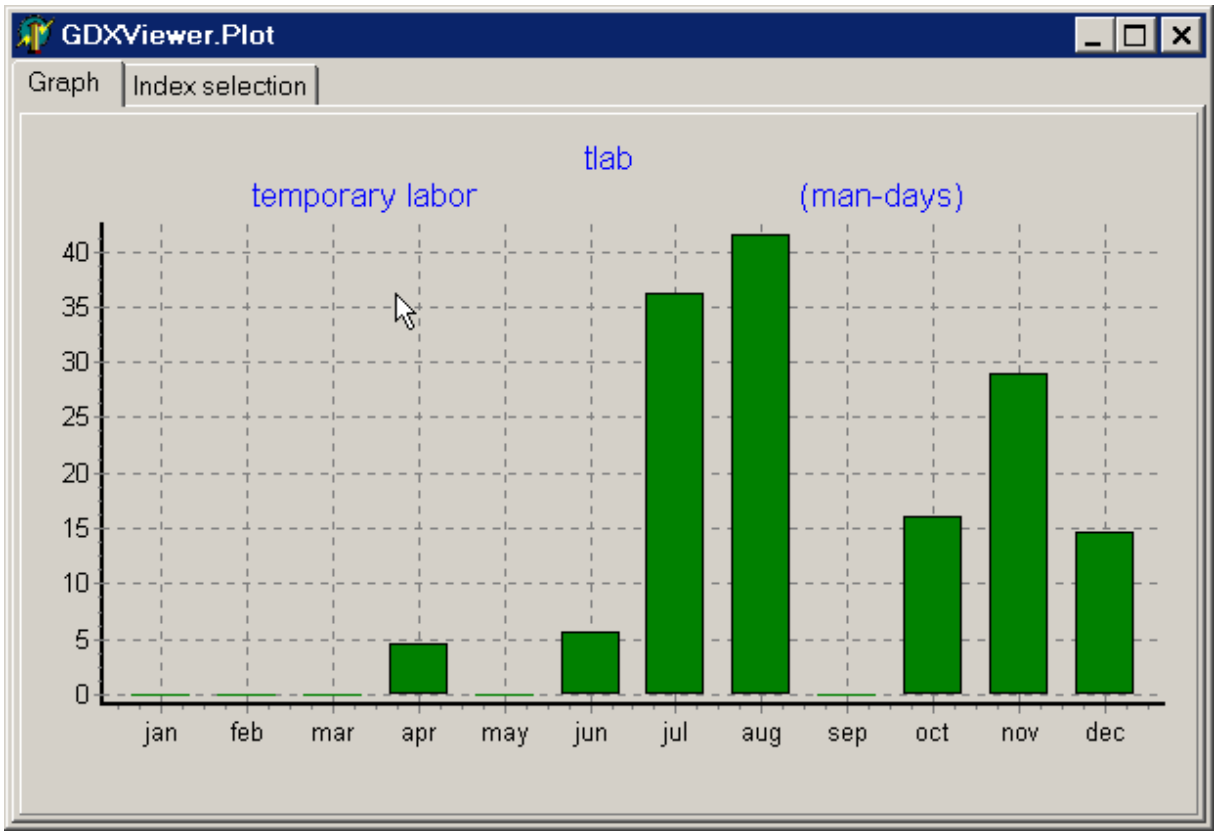
Figure 6.54 400px

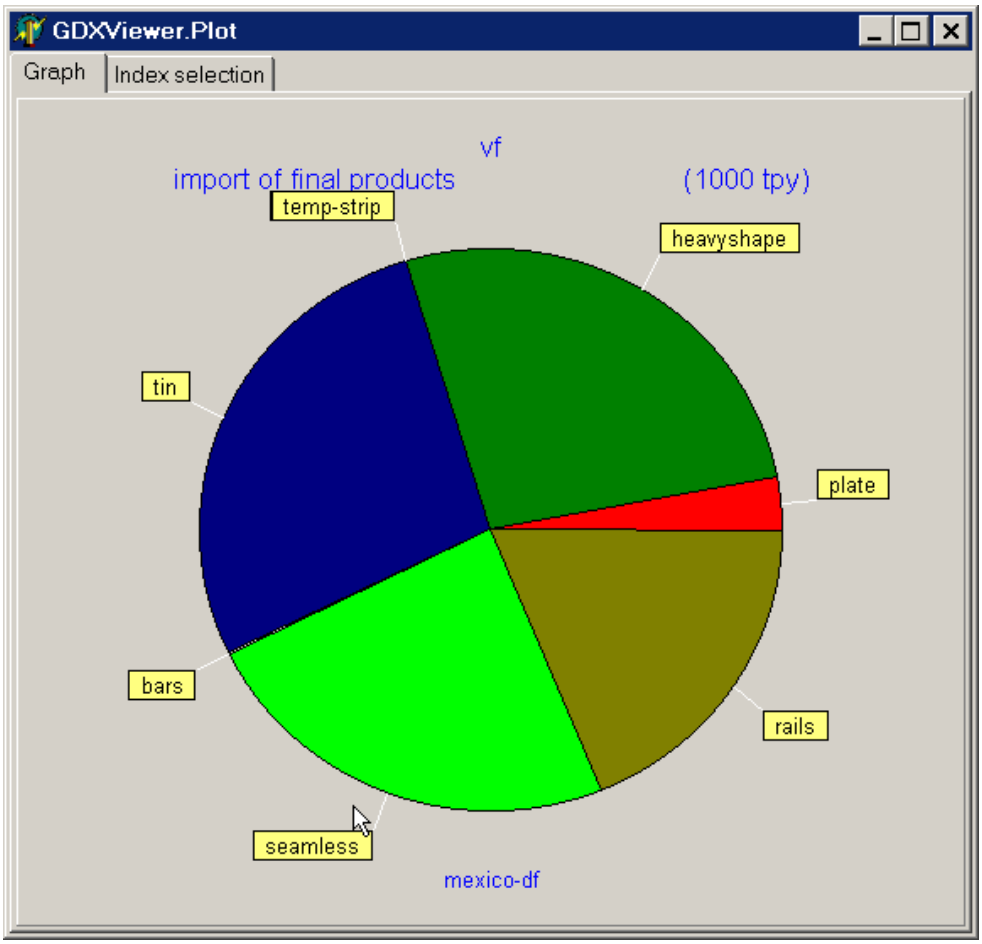
When we export to a GAMS include file all special values are understood, so the mapping is not used. The defaults button will reset the mapping to their default values.

6.27.20 Plotting Data

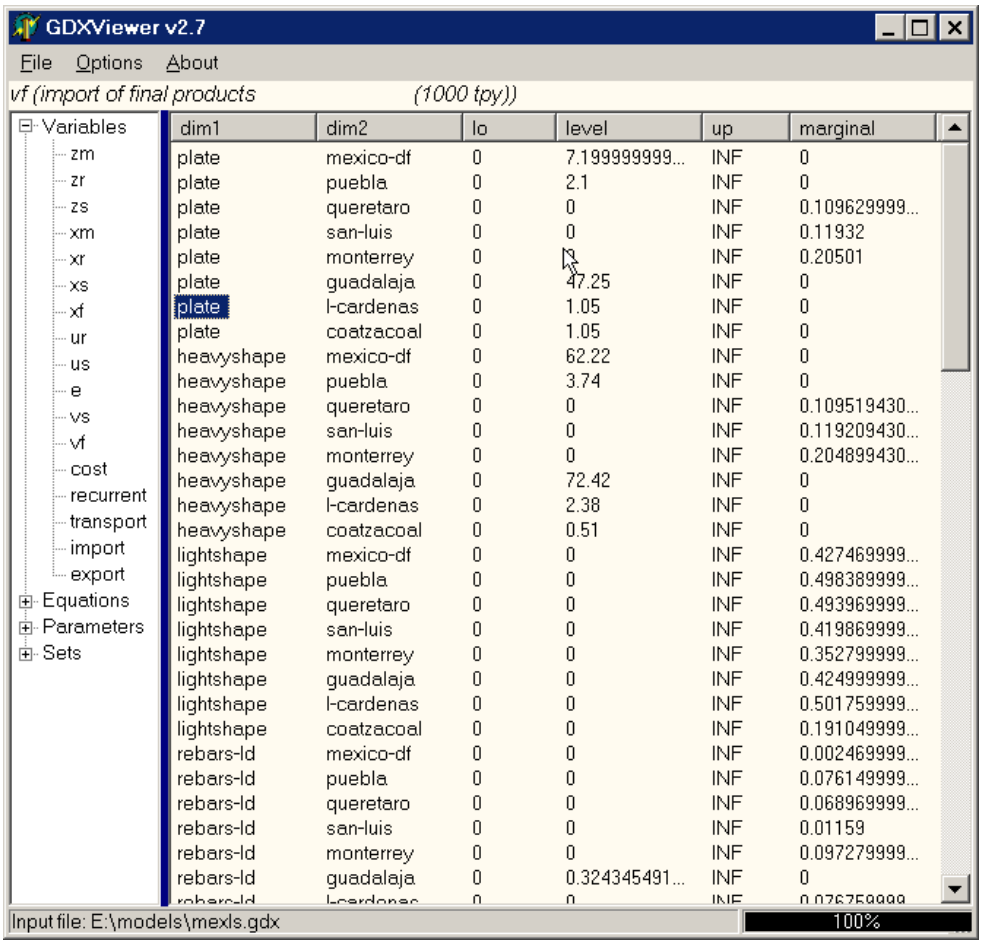
GDXVIEWER has a built-in facility to quickly plot data. It includes LINE, BAR and PIE charts, examples are shown below. The plots can be made through the menu `File|Plot`.



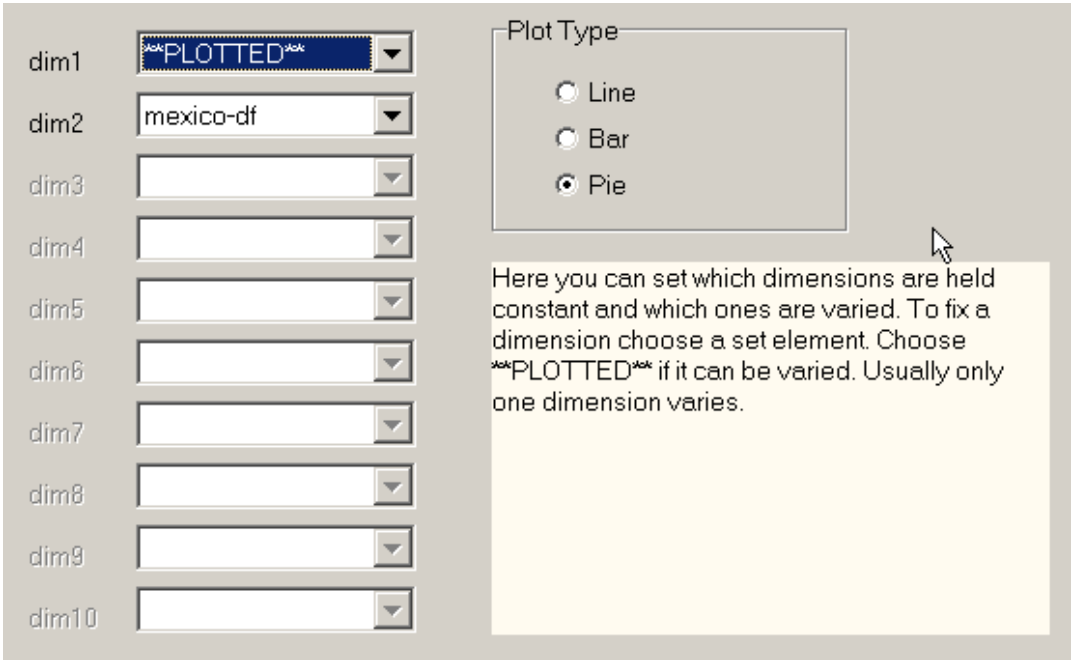

GDXViewer.Plot
Graph
Index selection
tlab
temporary labor
(man-days)
40
35
30
25
20
15
10
5
0
jan
feb
mar
apr
may
jun
jul
aug
sep
oct
nov
dec



For multi-dimensional data it may be needed to take a "slice" of the data to make meaningful graphs. In the example above we plotted a two dimensional quantity **vf** which looks like:

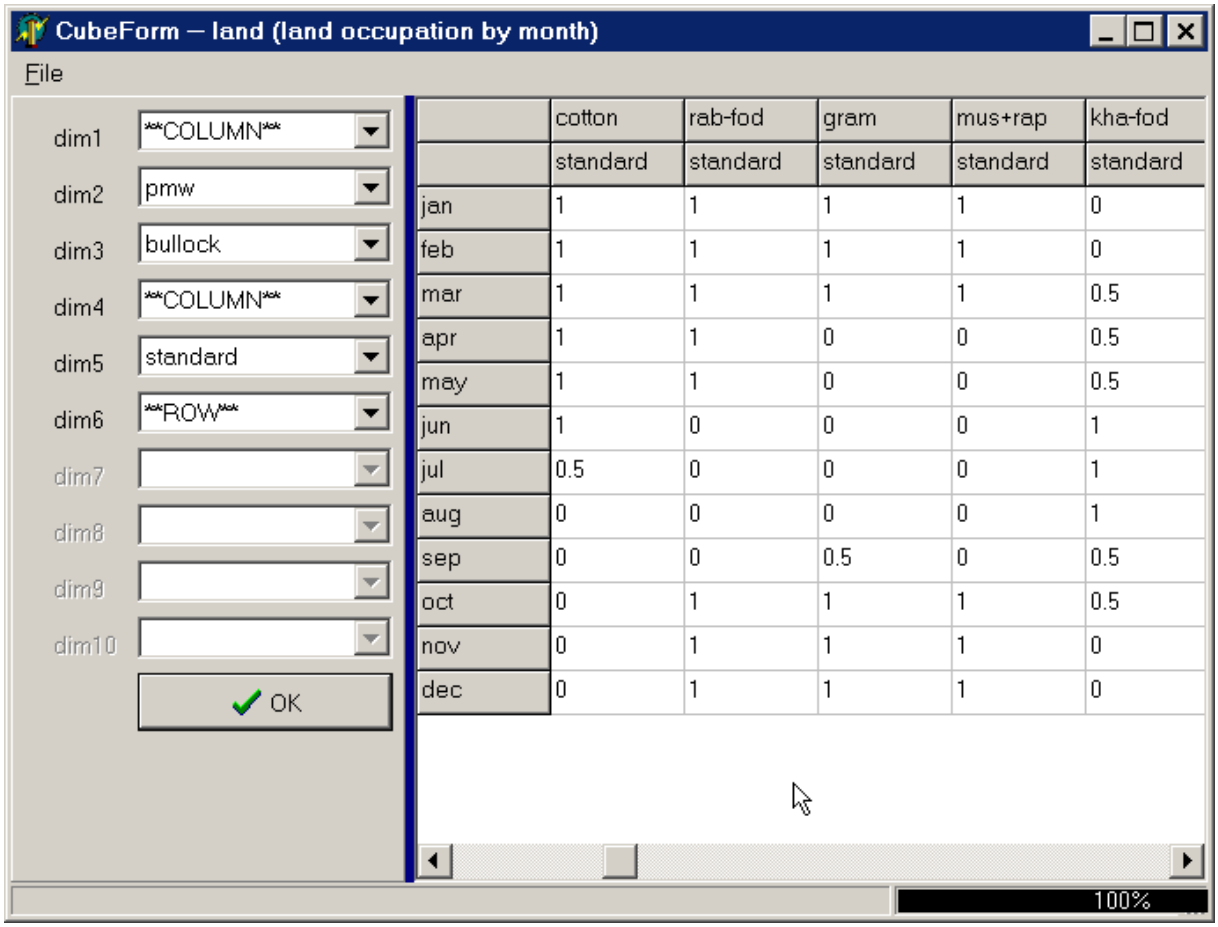


In this case we want to plot a pie graph of **vf(∗,'mexico-df')** which can be specified in the index-selection tab:



6.27.21 Cube View

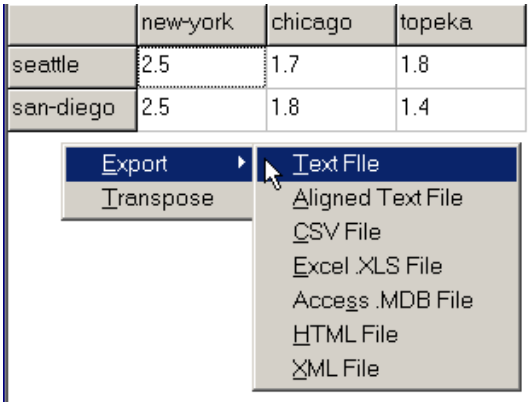
GDXVIEWER has a Cube View which allows to select rows and columns in a flexible way. In the example below we show a six dimensional variable where three dimensions are fixed, one dimension is chosen for the rows and two dimensions are chosen for the columns.



Below are some of the possibilities using parameter **d(i,j)** from the **trnsport.gms** model:

6.27.22 Exporting cubes

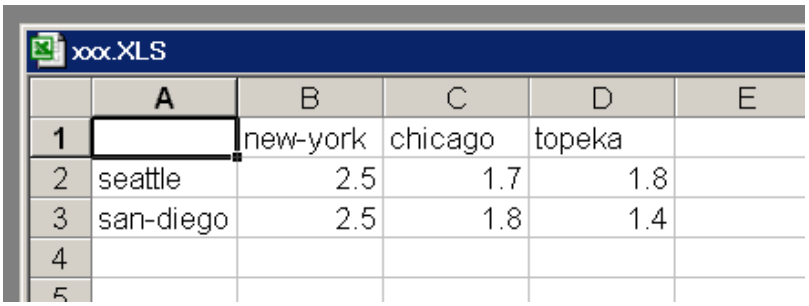
After creating a cube view, we can export that configuration by a right mouse click:



Exporting a cube will only export the selected slice (if certain dimensions are held fixed) and depending on the target format it will preserve the layout, e.g. an exported aligned text file can look like:

```
          new-york chicago topeka
  seattle      2.5     1.7    1.8
san-diego      2.5     1.8    1.4
```

Simarly, the XLS file can look like:



6.27.23 Commandline operation

The GDXViewer utility from version 2.3 accepts several command line parameters, so it can be used in a batch environment. When running in batch mode, the same configuration and option settings are used as for the interactive system and they can be changed by running GDXviewer interactively using the Options menu (the settings are saved in an INI file). It is advised to first run the program interactively until the results are as intended.

- **Single parameter** A single parameter is the filename of the GDX file. GDXViewer will load this file, and will continue to run interactively. Example: .

  ```
  Gdxviewer.exe test.gdx
  ```

- **XLS writing** To write an XLS file, one can use the syntax `i=inputfile.gdx xls=outputfile.xls id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (”). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is: @verbatim execute_unload 'd:\tmp\result.gdx',x; execute 'gdxviewer.exe i=d:\tmp\result.gdx xls=d:\tmp\result.xls id=x'; \endverbatim - Text file writing To write a text file, one can use the syntax <tt>i=inputfile.gdx txt=outputfile.txt id=x</tt>. If a path or filename contains blanks, the name can be surrounded by quotes (”). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

  ```
  execute_unload 'd:\tmp\result.gdx',x;
  execute 'gdxviewer.exe i=d:\tmp\result.gdx txt=d:\tmp\result.txt id=x';
  ```

- **CSV file writing** To write a CSV file, one can use the syntax `i=inputfile.gdx csv=outputfile.csv id=x`. If a path or filename contains blanks, the name can be surrounded by quotes (”). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is: @verbatim execute_unload 'd:\tmp\result.gdx',x; execute 'gdxviewer.exe i=d:\tmp\result.gdx csv=d:\tmp\result.csv id=x'; \endverbatim - HTML file writing To write an HTML file, one can use the syntax <tt>i=inputfile.gdx html=outputfile.html id=x</tt>. If a path or filename contains blanks, the name can be surrounded by quotes (”). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx html=d:\tmp\result.html id=x';
```

- **XML file writing** To write an XML file, one can use the syntax `i=inputfile.gdx xml=outputfile.xml id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is: @verbatim execute_unload 'd:\tmp\result.gdx',x; execute 'gdxviewer.exe i=d:\tmp\result.gdx xml=d:\tmp\result.xml id=x'; \endverbatim - GAMS include file writing To write a GAMS include file, one can use the syntax <tt>i=inputfile.gdx inc=outputfile.inc id=x</tt>. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

  ```
  execute_unload 'd:\tmp\result.gdx',x;
  execute 'gdxviewer.exe i=d:\tmp\result.gdx inc=d:\tmp\result.inc id=x';
  ```

- **Access MDB file writing** To write a Access MDB file, one can use the syntax `i=inputfile.gdx mdb=outputfile.mdb id=x`. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is: @verbatim execute_unload 'd:\tmp\result.gdx',x; execute 'gdxviewer.exe i=d:\tmp\result.gdx mdb=d:\tmp\result.mdb id=x'; \endverbatim - Excel Pivot Table writing To write a file containing a pivot table, one can use the syntax <tt>i=inputfile.gdx pivot=outputfile.xls id=x</tt>. If a path or filename contains blanks, the name can be surrounded by quotes ("). The 'id' parameter indicates the variable or parameter to export from the GDX file. A complete example is:

  ```
  execute_unload 'd:\tmp\result.gdx',x;
  execute 'gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
  ```

- **SQL Database Table writing** To write a table to an SQL database, first interactively configure the connection to the database. The Export SQL Database option allows you to see if a connection succeeded and if the correct database was accessed. The configuration information is written to the SQLVIEWER.INI configuration file. The information in this file is used also when performing a batch command-line operation. The syntax is: `i=inputfile.gdx sql id=x`. A complete example is:

  ```
  execute_unload 'd:\tmp\result.gdx',x;
  execute 'gdxviewer.exe i=d:\tmp\result.gdx sql id=x';
  ```

If you need to access several different databases, you can copy the file `SQLVIEWER.INI` (located in the directory where `SQLVIEWER.EXE` is placed). To tell `GDXVIEWER` to read a different INI file, you can say:

```
execute_unload 'd:\tmp\result.gdx',x;
execute 'gdxviewer.exe i=d:\tmp\result.gdx ini=copy.ini sql id=x';
```

GDXVIEWER uses the MS Access and MS Excel applications as COM Object to write files in XLS (both XLS and PIVOT commands) or MDB format. Those applications may write to `C:\My Documents` in case no full path is specified. Other formats use the default GAMS working directory. In case when running under the IDE this is the location of the project file (∗.GPR).

If a path or file name contains a blank, then it is possible to surround the name by double quotes as in:

```
execute_unload 'result.gdx',x;
execute 'gdxviewer.exe i=result.gdx csv="c:\my documents\result.csv" id=x';
```

Under windows 98 ME the call

```
execute 'gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

will cause GAMS to continue while GDXVIEWER is executing. If we use:

```
execute '=gdxviewer.exe i=d:\tmp\result.gdx pivot=d:\tmp\result.xls id=x';
```

GAMS will wait until **gdxviewer.exe** is terminated before executing more statements. This situation is different under other operating systems such as XP and NT.

6.27.24 Notes

GDX

GDX stands for **Gams Data** Exchange. It is a binary file format to get data in and out of GAMS.

6.28 GDXXRW

`GDXXRW` is a utility to read and write Excel spreadsheet data. `GDXXRW` can read multiple ranges in a spreadsheet and write the data to a GDX file, or read from a GDX file, and write the data to different ranges in a spreadsheet.

Note

- GDXXRW is available on Windows only. This is an important factor to consider when moving an existing code to GAMS Engine.
- GDXXRW can also be used to read csv files. Please be aware that Excel files have a row and column number limit and that rows and columns in csv files beyond this limit may be ignored.

6.28.1 Usage

```
gdxxrw inputFile {outputFile} {options} [symbols]
```

Options and symbol specifications can also be read from a text file; the use of an option file is indicated by preceding the file name with a `@` (At sign.). When reading from a text file, lines starting with an asterisk (∗) will be ignored and act as a comment.

Options and symbol specifications can also be read from an area in a spreadsheet; see index below.

Files without a full path name are assumed to be in the current directory when using a command prompt. When using the GAMS IDE, these files are assumed to be in the current project directory. The use of file names with embedded blanks is allowed as long as the file name is enclosed in double-quotes (").

Note

A libinclude tool is available to see if Excel is installed, to close an Excel file etc. See Windows Only Tools (win32) for more details. To read data from an Excel file without Excel installed see XLSDUMP.

6.28.2 Options

Describing the actions to be taken by GDXXRW requires passing a number of options and symbol specifications to the program. The ability of GDXXRW to process multiple actions in a single call makes the option passing and interpretation more complex.

There are four kinds of options:

1. **Immediate** Immediate options are recognized and processed before any other actions are taken and can only be specified once. Examples are:

   ```
   input= output= trace=
   ```

2. **Global** Global options are interpreted from left to right and affect every action that follows. The same option can be used multiple times to affect the actions that follow. Examples are:

   ```
   skipEmpty= epsOut=
   ```

3. **Symbol** A symbol definition introduces a new action for reading or writing a symbol. Examples are:

   ```
   par= set= dSet=
   ```

4. **Symbol attributes** Attributes specify additional information for the last symbol defined. Examples are:

   ```
   dim= cDim= merge clear
   ```

6.28.2.1 Immediate Options

Immediate options are recognized independent of their position on the command line. They are global and they can only be specified once.

Option	Default	Description
input	none	Specify the input filename (required).
output	inputFileName	Specify the output filename.
log	none	Specify the log filename.
logAppend	none	Appending the log information to the file specified.
index	none	Indicates reading the options and symbols directly from the spreadsheet.
password	none	Password for an encrypted input file.
rWait	0	Delay after opening a spreadsheet.
checkDate	disabled	Write GDX file only if the input file is more recent than the GDX file.
useRC	disabled	Use Row-Column notation to specify cells and ranges.
reCalc	N	Controls if recalculations of cells inside Excel are executed after writing to the spreadsheet.
trace	1	Controls the amount of information written to the log.
maxDupeErrors	0	Maximum number of duplicate records allowed for a symbol.
updLinks	0	Updating of cells that refer to other spreadsheets.
runMarcos	0	Execution of Excel Auto macros.

Some more detailed remarks on the immediate options:

input = *fileName* (required, default = none)

Synonym: i

Either use the keywords `input` or `i` to specify the input file name anywhere on the command line or just specify the input file name without keyword at the first position behind `GDXXRW`. The file extension of the input file is required and determines the action taken by the program.

The extension .gdx for the input file will read data from a GDX file and write data to a spreadsheet. The extension .xls, .xlsx, .xlsm, or .xlsb for the input file will read a spreadsheet and write the data to a .gdx file. In addition to the .xls, .xlsx, .xlsm, or .xlsb input file extension, the following file extensions are also valid for spreadsheet input files: .wk1, .wk2, .wk3 and .dbf.

A file sharing conflict will arise when writing to a spreadsheet with the target file open in Excel. Either close the file in Excel before executing `GDXXRW`, or mark the spreadsheet as a shared workbook in Excel. To change the shared status of a workbook, use the Excel commands available under: `Tools`|`Share Workbook`.

Writing to a shared workbook can be painfully slow; simply closing the file and reopen the file after `GDXXRW` has finished is often a better option.

output = *fileName* (default = inputFileName)

Synonym: o

When an output file is not specified, the output file will be derived from the input file by changing the file extension of the input file and removing any path information. The file type, i.e. the file extension, depends on the installed version of Excel. Versions prior to Excel 2007 use the .xls file extension, later version use .xlsx, .xlsm, and .xlsb. Excel 2007 can write .xls files, but in that case the output file has to be specified with an .xls file extension.

log = *fileName* (default = none)

Specifies the filename of the log file. When omitted, log information will be written to standard output. When using `GDXXRW` in a GAMS model that is started from the GAMS IDE, the output will be written to the IDE process window.

logAppend = *fileName* (default = none)

Using `logAppend` will add the log information to the end of the file specified. If the file does not exist yet, a new one will be created.

index = *Excel Range*

The `index` option is used to obtain the global options, symbols and symbol attributes specified by reading them from the spreadsheet directly. The parameters are read using the specified range, and treated as if they were specified directly on the command line. The first three columns of the range have a fixed interpretation: dataType, Symbol identifier and dataRange. The fourth and following columns can be used for additional parameters. The column header contains the keyword when necessary, and the cell content is used as the parameter value. See Reading Spreadsheet using the Index Option for instance.

password = *string* (default = none)

Specifies a password for a protected spreadsheet file.

rWait = *integer* (default = 0)

Introduce a delay (in milliseconds) after opening a spreadsheet before accessing the data. This parameter can be used to work around an issue we encountered that Excel indicated it was not ready. The issue can arise during the data exchange with Excel Sheets that contain macros, plots or pivot tables where `GDXXRW` attempts to access a sheet while Excel is busy updating macros, graphs, and pivot tables.

checkDate (disabled by default)

When specified, no data will be written if the output file already exists and the file date for the output file is more recent than the file date for the input file. Provides a simple check to update the output file only if the input file has changed to save resources.

useRC (disabled by default)

Specify that all cell and range references use RC notation. So, instead of specifying the range Sheet1!A1:D6, one specifies Sheet1!R1C1:R6C4. When tracing is enabled, ranges will be reported in RC notation. This is a global option and applies to all cell references.

reCalc = *flag* (default = `N`)

Enable or disable the recalculations of cells inside Excel after writing to the spreadsheet. If there are many formulas in the spreadsheet the recalculation of cells can become very expensive and slowing down the writing process. By default, the recalculation is disabled and can be enabled via this option.

trace = *integer* (default = 1)

Sets the amount of information written to the log (for a better debugging). Higher values will generate more output. Valid range is `0..4`.

0 Minimal information is included in the output
1 Message appears telling about each `GDXXRW` call indicating input file, output file and execution time
2 Message appears giving the level 1 output plus a listing for each symbol specified, indicating the type, sheet name, dimension, data range and the range of the row and column headers
3 Message appears giving the level 2 output plus cell ranges affected by reading, writing and clearing
4 Message appears giving the level 3 output plus cell addresses, and numerical or string values for every cell worked with

maxDupeErrors = *integer* (default = 0)

Sets the maximum number of duplicate records that is allowed when reading a spreadsheet and writing to a GDX file. The duplicate records for each symbol will be reported in the logfile, and if their accumulated number does not exceed the maximum specified using this option, the GDX file will not be deleted. This is a global option and applies to each symbol read from the spreadsheet.

The option is demonstrated in Reading Set from Lists with Duplication.

updLinks = *integer* (default = 0)

Specifies how links in a spreadsheet should be updated. The valid range is `0..3`.

0 Doesn't update any references
1 Updates external references but not remote references
2 Updates remote references but not external references
3 Updates both remote and external references

runMacros = *integer* (default = 0)

This option controls the execution of the 'Auto_open' and the 'Auto_close' macros when opening or closing a spreadsheet. Valid range is `0..3`.

0 Doesn't execute any macros
1 Executes Auto_open macro
2 Executes Auto_close macro
3 Executes Auto_open and Auto_close macro

6.28.2.2 Global Options

The following options affect the symbols that follow the option. They remain in effect unless they are redefined and used again for another symbol.

Option	Default	Description
acronyms	0	Controls the handling of acronyms.
cMerge	0	Controls the handling of merged Excel ranges.
dSetText	`N`	Read explanatory text for set elements of domain sets.
epsOut	`Eps`	String to be used when writing the value for 'Epsilon'.
filter	0	Set the Excel filter for symbols written to Excel.
incRC	`N`	Include Excel row and column indices when a symbol is written to the GDX file.
mInfOut	`-Inf`	String to be used when writing the value for 'Negative infinity'.
NaIn	none	String to be used when reading a value for 'Not available'.
nameConv	`N`	Controls the interpretation of an Excel range.
NaOut	`NA`	String to be used when writing the value for 'Not available'.
pInfOut	`+Inf`	String to be used when writing the value for 'Positive infinity'.
resetOut	disabled	Reset the output strings for special values to their defaults.
squeeze	`Y`	Controls writing of default values of sub-fields of variables and equations resp. the handling of zero values within parameters when reading from spreadsheet.
skipEmpty	1	Number of empty row or column cells to ignore before the next empty row or column indicates the end of a block when reading from spreadsheet using the top left cell specification.
UndfOut	`Undf`	String to be used when writing the value for 'Undefined'.
allUELs	`Y`	Controls the handling of UELs without associated values in the data range.
zeroOut	0	String to be used when writing the value for 'Zero'.

Some more detailed remarks on the global options:

acronyms = *integer* (default = `0`)

A non-zero value indicates that acronyms can be expected and should be processed.

If no acronym processing takes place, reading an identifier in the data section of a sheet will generate an error. Writing an acronym to a sheet will write the internal numerical representation of the acronym.

Processing acronyms:

When reading a spreadsheet, an identifier in the data section of the sheet will be interpreted as an acronym and will be written to the GDX file.

When writing to a spreadsheet, a data tuple containing an acronym will be stored using the corresponding identifier of the acronym.

cMerge = *integer* (default = `0`)

Option indicating how to read an empty cell that is part of a merged Excel range. See Reading merged Excel Ranges with cMerge. Possible values and their interpretation are:

0 Leave the cell empty
1 Use merged value in row and column headers only
2 Use merged value in all cells

dSetText = *flag* (default = `N`)

Valid only when reading a spreadsheet.

This controls the reading of explanatory text for set elements of domain sets. By default, no text is read for domain sets. If this option is activated this is changed. If an element appears more than once, the first one defines the explanatory text read.

epsOut = *string* (default = `Eps`)

String to be used when writing the value for 'Epsilon'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet and Writing Parameter to Spreadsheet including Zero Values.

filter = *integer* (default = `0`)

Adds basic Excel filter to the columns of the spreadsheet to display only those values matching some conditions. Using this option when reading an Excel file will result in an error. Specifying `filter=1` will set an Excel filter for the row of labels that are closest to the data values. When there are multiple rows in a column header (`cDim > 1`) we can specify `filter=x` where `x` is a number of the range `2..cDim`, indicating to use a row farther away from the data values. See also Writing to Spreadsheet using a Filter.

incRC = *flag* (default = `N`)

Valid only when reading a spreadsheet.

Include Excel row and column indices when a symbol is written to the GDX file. For example, when we write a parameter `P` with indices `I` and `J`, without this option it will be written as `P(I, J)`. When `incRC` is enabled, the parameter will be written as `P(Excel_Rows, I, Excel_Columns, J)`. Note that the sets `Excel_Rows` and `Excel_Columns` will be added to the GDX file automatically.

mInfOut = *string* (default = `-Inf`)

String to be used when writing the value for 'Negative infinity'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

NaIn = *string* (default = none)

String to be used when reading a value for 'Not available'; this string is recognized in addition to the string 'NA' and is not case-sensitive. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

nameConv = *flag* (default = `N`)

Synonym: nC

The naming convention parameter is used to change the interpretation of an Excel range that does not contain an '!' (exclamation mark). For details see Excel Ranges below.

NaOut = *string* (default = `NA`)

String to be used when writing the value for 'Not available'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

pInfOut = *string* (default = `+Inf`)

String to be used when writing the value for 'Positive infinity'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

resetOut (disabled by default)

Reset the output strings for special values to their defaults. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

squeeze = *flag* (default = `Y`)

Synonym: sq

Writing to a spreadsheet:

The `squeeze` option affects the writing of sub-fields of variables and equations. A value for the field that is the default value for that type of variable or equation will not be written to the spreadsheet. For example, the default for `.l` (level value) is `0.0`, and therefore zero will not be written to the spreadsheet. When we set `squeeze=n`, all values will be written to the spreadsheet.

The `squeeze` option for writing data is demonstrated in Reading Data from Spreadsheet and writing Data t and Writing Parameter to Spreadsheet including Zero Values.

Reading a spreadsheet:

When the `squeeze` option is enabled, zero values for parameters will not be written to the GDX file. When the `squeeze` option is disabled, zero values will be written to the GDX file. In either case, empty cells, or cells containing blanks only, will never be written to the GDX file.

The `squeeze` option for reading data is demonstrated in Reading Special Values from Spreadsheet and writi

skipEmpty = *integer* (default = `1`)

Synonym: sE

The `skipEmpty` option can be used when reading a spreadsheet and the range is specified using the top left corner instead of a block range. The value defines the number of empty row or column cells to ignore before the next empty row or column signals the end of a block. Valid values are `0..n`. If the range is specified using a block range, `skipEmpty` will be ignored. Blank rows or columns will be skipped automatically.

Note that `skipEmpty` is also valid when using the merge resp. clear options in order to write data to spreadsheet (in a specific order determined by matching row and column labels already stored in the spreadsheet).

See Skipping Empty Rows and Columns or Reading Multi-dimensional Parameter from Spreadsheet for instance.

UndfOut = *string* (default = `Undf`)

String to be used when writing the value for 'Undefined'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

allUELs = *flag* (default = `Y`)

Valid only when reading a spreadsheet.

When enabled, all unique elements found in a range will be entered in the GDX file. When disabled, only those unique elements that are used in conjunction with a value will be entered in the GDX file.

zeroOut = *string* (default = `0`)

String to be used when writing the value for 'Zero'; by default this is '0'. This option is demonstrated in Reading Special Values from Spreadsheet and writing to Spreadsheet.

6.28.2.3 Symbols

To write data to a spreadsheet or to a GDX file, one or more symbols and their associated Excel range need to be specified. See also Excel Ranges.

The general syntax for a symbol specification is:

dataType=*symbolName* {symbolAttributes}

dataTyp	Description

Among the symbolAttributes, one specifies the dataRange, the dimensions of the symbol and some additional symbolOptions in general.

dataType

dataTyp	Description
par	Declare the symbol as parameter and define a individual name when reading from spreadsheet, or specify a parameter from a GDX file when writing to spreadsheet.
equ	Specify a sub-field of a equation from a GDX file when writing to spreadsheet.
var	Specify a sub-field of a variable from a GDX file when writing to spreadsheet.
set	Declare the symbol as set and define a individual name when reading from spreadsheet, or specify a set from a GDX file when writing to spreadsheet.
dSet	Declare the symbol as domain set and define a individual name when reading from spreadsheet, or specify a domain set from a GDX file when writing to spreadsheet.
text	Write the text specified to spreadsheet. One can also create hyperlinks, using the link resp. linkID statement.
textID	Write the explanatory text of an identifier stored in the GDX file to spreadsheet.
hText	Write sections of the text specified to different cells in the horizontal direction (row).
vText	Write sections of the text specified to different cells in the vertical direction (column).

par = *GAMS_Parameter*

Specify a GAMS parameter to be read from a GDX file and written to spreadsheet, or to be read from a spreadsheet and written to a GDX file.

When writing to a spreadsheet, special values such as `Eps`, `NA` and `Inf` will be written in ASCII. When reading data from a spreadsheet, the ASCII strings will be used to write the corresponding special values to the GDX file.

This datatype is used in the most examples. Reading parameters is demonstrated in Reading Parameter from Spreadsheet or Reading Multi-dimensional Parameter from Spreadsheet. Writing parameters from GDX to spreadsheet is demonstrated in Writing Parameter to Spreadsheet.

equ = *GAMS_Equation*
var = *GAMS_Variable*

A sub-field of a variable or equation can be written to a spreadsheet and should be specified as part of the symbolName. The fields recognized are `.l` (level), `.m` (marginal), `.lo` (lower bound), `.up` (upper bound), `.prior` (priority), and `.scale` (scale). The sub-field names are not case-sensitive. See also Reading Data from Spreadsheet and writing Data to Spreadsheet after Solve.

A sub-field of a variable or equation cannot be read from a spreadsheet and written to a GDX file.

set = *GAMS_Set* [values = *valueType*]

In GAMS we can define a set by specifying all its elements. In addition, each tuple can have an associated text. To read a set from a spreadsheet, the values option is used to indicate if there is any data, and if there is, if the data should be interpreted as associated text or as an indicator whether the tuple should be included in the set or not.

Reading sets is demonstrated in Reading Set from Spreadsheet or Reading Set Elements associated with Data o (focusing on the `values` option) for instance. Writing sets from GDX to spreadsheet is demonstrated in Writing Set to Spreadsheet.

valueType	Interpretation
auto	Based on the range, row and column dimensions for the set, the program decides on the valueType to be used. This is the default for `values`.
noData	There is no data range for the set; all tuples will be included.
YN	Only those tuples will be included that have a data cell that is not empty and does not contain '0', 'N' or 'No'.
sparse	Only those tuples will be included that have a data cell that is not empty. The string in the data cell will be used as the associated text for the tuple.
dense	All tuples will be included. A string in the data cell will be used as the associated text for the tuple.

Due to backward compatibility, `valueType=string` or `all` are also recognized and are synonyms for `valueType=dense`. The following table summarizes which valueType will be used when reading a set, if a valueType was not specified:

Range specification	rDim = 0 Or cDim = 0	rDim > 0 And cDim > 0
Top left corner only	dense	YN
A block, but the data range is empty	dense	YN
A block, and there is a data range	dense	YN

When writing to a spreadsheet, the entire set is written to the spreadsheet and the writing of the associated text is governed by the values option:

valueType	Interpretation
auto	If `rDim=0` or `cDim=0`, auto means string, otherwise auto means YN.
noData	Neither associated text nor 'Y' is written for a set element.
YN	A 'Y' is written for a set element.
string	The associated text is written for a set element. If no text is stored with set element the cell will be empty.

Due to backward compatibility, `valueType=dense`, `sparse` or `all` are also recognized and are synonyms for `valueType=string`.

dSet = *GAMS_Set*

A domain set is used to read the domain of a set from a spreadsheet row or column. Either the row or the column dimension (`rDim` or `cDim`) should be set to '1' to specify a row or column for the set, resulting in a one-dimensional set. Duplicate labels in the range specified do not generate an error message. For instance, see also Reading Set from Lists with Duplication.

Note that reading explanatory text of set elements is not supported by `dSet`. In order to read explanatory text, use set instead. If there are duplicate set element labels in your data, use the `set` symbol specification while increasing the value of the immediate option maxDupeErrors to oppress an error message when reading duplicates.

text = *"String of characters"* {dataRange}
textID = *Identifier* {dataRange}

Write the text to the cell specified in the DataRange. In addition, `textID` will write the explanatory text of the Identifier in the cell to the right of the dataRange.

A Text directive can be followed by a link=*Address* or linkID=*identifier* directive. Using `link` will create a hyperlink to an external page or to a cell in the spreadsheet, while `linkID` will create a hyperlink to the top left corner of the symbol specified. See Writing to Spreadsheet adding Text and Hyperlinks for instance.

hText = *"String of characters"* {dataRange}
vText = *"String of characters"* {dataRange}

Write a string of characters in the horizontal direction for `hText` or vertical direction for `vText`. Text for the next cell is indicated by a comma. In order to write a comma as part of the text, the comma needs to be preceded by a backslash. See Writing Set to Spreadsheet for instance.

Symbol Attributes

The following options apply to the symbol preceding the option, and only affect that symbol:

Parameter	Default	Description
dataRange	Cell A1 of the first sheet	Specify the Exel range of the symbol for reading from spreadsheet or for writing to spreadsheet.
dim	2 when reading from spreadsheet Defined by the symbol dimension stored in the GDX file when writing to spreadsheet	Total dimension of the symbol. Please also refer to section More about dimensions.
cDim	`1`	Column dimension of the symbol. Please also refer to section More about dimensions.
rDim	`dim-1`	Row dimension of the symbol. Please also refer to section More about dimensions.
merge	disabled	When enabled, the data will be written in a specific order determined by matching row and column labels already stored in the spreadsheet.
clear	disabled	In addition to the effect of `merge`, already existing values in the data range of the spreadsheet are removed before writing.
colMerge	0	Determines the columns for which non-empty content of the previous cell will be used as content for the empty cell of a column.
intAsText	`Y`	Determines the cell format when writing unique elements that are a proper integers to spreadsheet.
ignoreRows/Cols	none	Specify rows and columns to be ignored when reading from spreadsheet.

Some more detailed remarks on the symbol attributes:

dataRange

rng = *Excel Range*

The Excel Range for the data for the symbol. Note that an empty range is equivalent to the first cell of the first sheet.

dimensions

dim = *integer*

The total dimension for the symbol.

cDim = *integer*

Column dimension: the number of rows in the data range that will be used to define the labels for the columns. The first `cDim` rows of the data range will be used for labels.

rDim = *integer*

Row dimension: the number of columns in the data range that will be used to define the labels for the rows. The first `rDim` columns of the data range will be used for the labels.

More about dimensions:

The sum of `cDim` and `rDim` determine the dimension of the symbol: `dim = cDim + rDim`.

Reading data from a GDX file and writing to a spreadsheet:
In this case, the dimension of the symbol is stored in the GDX file and therefore known. `cDim` and/or `rDim` can be omitted. If both `cDim` and `rDim` are omitted, the program assumes that `cDim=1` and `rDim=dim-1`.

Reading a spreadsheet and writing data to a GDX file:
In this case, the dimension of the symbol is not known. If neither `cdim` nor `rdim` are known, both default to `1` (hence the default value for `dim` is `2`). If `dim` and either `cdim` or `rdim` are known, the missing dimension is calculated using `dim = cDim + rDim`. If only `cdim` or `rdim` are known, but `dim` is not, the missing dimension is set to `0`.

symbolOptions

The options below are only valid when reading a GDX file and writing to a spreadsheet.

By default, writing data to a spreadsheet will include the row and column labels in addition to the data. The row and column labels will appear in the same order as they appear in the GDX file.

merge (disabled by default)

Using the `merge` option assumes that the row and column labels are in the spreadsheet already. For each value read from the GDX file, the location of the row and column labels is used to update the spreadsheet. Using the `merge` option will force the data to be presented in a given order using the row and column labels. Spreadsheet cells for which there is no matching row/column pair will not be changed. The matching of labels is not case-sensitive. See also Writing to Spreadsheet with merge Option Example.

Note that the skipEmpty option value affects the reading of the row and column labels from spreadsheet in case of top left range specification (while `skipEmpty` is ignored in case of block range specification).

Warning: The `merge` or `clear` option will clear the Excel formulas in the rectangle used, even if the cells do not have matching row/column headings in the GDX file. Cells containing strings or numbers are not affected.

clear (disabled by default)

The `clear` option is similar as the `merge` option, except that the data range will be cleared before any data is written. See also Writing to Spreadsheet with clear Option Example.

colMerge = *integer* (default = 0)

The number of columns that will use a previous value in that column if the cell is empty. Can only be used when reading from a spreadsheet. See Reading empty Cells with colMerge.

intAsText = *flag* (default = `Y`)

Unique elements that are a proper integer can be written as text or as an integer value. The default is `Y`, which will write the unique element as a string. Note that this impacts the sorting order and can be used when using an Excel filter on a data range.

Ignoring Rows and Columns when reading from a spreadsheet

ignoreRows = *rownr, rownr, rownr:rownr*
ignoreColumns = *colnr, colnr, colnr:colnr*

Row numbers are represented by integers. Column numbers are represented by Excel column numbers, like A, CD, IV etc, or by integers.

Note

- Ignoring rows or columns is only allowed when reading a spreadsheet.
- The specification of ignored rows or columns follows the symbol specification and only applies to that symbol.
- When ignoring a column that would be part of an index if the column was not ignored, the range for the index will be extended for each column that is ignored. The same holds for ignored rows that are part of an index.

See also Ignoring Rows and Columns.

6.28.2.4 Syntax Elements

The most options are specified by using an integer, a string or a flag. Note that the options `useRC`, `resetOut`, `checkDate`, `merge` and `clear` are enabled or disabled by simply adding the keyword to your `GDXXRW` statement.

Element	Description
integer	An unsigned integer
string	A string of characters; a string can be quoted with single or double quotation marks.
flag	True values: `1`, `Y` or `Yes` False values: `0`, `N` or `No` (not case-sensitive)

6.28.2.5 Excel Ranges

An Excel Range is specified using the standard Excel notation: `SheetName!CellRange`.

When the `SheetName!` is omitted, the first sheet will be used. A CellRange is specified by using the `TopLeft:BottomRight` cell notation like `A1:C12`. When `:BottomRight` is omitted, the program will extend the range as far down and to the right as possible. (Using '..' in stead of ':' is supported.)

Excel also allows for named ranges; a named range includes a sheet name and a cell range. Before interpreting a range parameter, the string will be used to search for a pre-defined Excel range with that name. See Reading Parameter from Spreadsheet using pre-defined Excel Named Ranges for instance.

When writing to a spreadsheet and a sheet name has been specified that does not exist, a new sheet will be added to the workbook with that name. Reading a spreadsheet and using an unknown range or sheet name will result in an error.

The following table summarizes all possible input combinations and their interpretation:

Input	Sheet used	Cell(s) used	Condition
	First sheet	A1	
!	First sheet	A1	
Name	First sheet	Name	When nc=0
Name	Name	A1	When nc=1
Name!	Name	A1	
!Name	First sheet	Name	
Name1!Name2	Name1	Name2	

The term `nc=` refers to the nameConv option.

6.28.3 Return Codes

On success, `GDXXRW` will return 0 as error code. However, there might be an error which will be signaled with a specific return code in addition to an error message.

Return Code	Interpretation
0	No error

Return Code	Interpretation
1	Cannot write log
2	GDX error
3	No input file
4	Input file not found
5	Bad parameter
6	Read error
7	Problem loading GDX DLL
8	Symbol not found
9	Dimension different
10	Types different
11	Bad UELs
12	Bad output file
13	Problem opening Excel
14	Problem writing to Excel
15	Problem reading from Excel
16	Duplicate entry
17	Cannot add sheet
18	Bad cell value
19	Dimension conflict
20	Data exceeds range
21	Exceeds range or memory problem
22	Deprecated
23	Program aborted
24	Merge range empty
25	Too many columns skipped
26	Too many rows skipped

6.28.4 Warning

When executing `GDXXRW` twice and redirecting output to the same log file may result in a fatal error.

For example:

```
gdxxrw step1 parameters > logfile
gdxxrw step2 parameters > logfile
```

The execution of step2 may fail, because Excel will close the logfile in step1 in a delayed fashion, but return control to `GDXXRW` immediately. Using the log or logAppend parameter will avoid this problem.

6.28.5 Reading from Spreadsheet - Examples:

6.28.5.1 Reading Set from Spreadsheet

Assuming we want to read set elements from the first sheet of the spreadsheet file exampleData.xlsx and write the data to exampleData.gdx.

Either of the following two statements below reads the second row of set elements from the spreadsheet above:

```
gdxxrw exampleData.xlsx  set=i1  rng=readingSets!A2:C2 cDim=1
gdxxrw exampleData.xlsx dSet=i1a rng=readingSets!A2:C2 cDim=1
```

When the output file is not specified, the output file will be derived from the input file by changing the file extension of the input file and removing any path information. Since all elements in the second row are unique, there is no need of increasing the maxDupeErrors parameter to avoid an error message when defining the symbol as set. By specifying the symbol directly as a dSet (domain set) in the second statement, duplicate labels would be removed without throwing an error. We set cDim to one so that the first row of the range is used for the labels of the set.

On the other hand if we want to read set elements listed in a column:

Either of the following two statements reads column A of set elements from the spreadsheet above:

```
gdxxrw exampleData.xlsx  set=j1  rng=readingSets!A35:A37 rDim=1
gdxxrw exampleData.xlsx dSet=j1a rng=readingSets!A35:A37 rDim=1
```

Besides the range we also had to change the parameter rDim to indicate that the first column of the range is to be used for the labels.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample18**] for reference.

6.28.5.2 Reading Set and Explanatory Text

Suppose we want to read the set elements in the ninth row and their associated text in the tenth row of the following spreadsheet:

We can read the set elements and their explanatory text by executing the command:

```
gdxxrw exampleData.xlsx set=i3 rng=readingSets!A9:E10 cDim=1
```

To read the explanatory text, we simple include the tenth row within the range of the symbol `i3` and specify `cDim=1`. By doing this, the first row of the range, i.e. the elements of the ninth row, will be used as the set elements, while the tenth row will be interpreted as their associated text (depending on the values option specified. By default, the `values` option is set to `dense` in this example, i.e. all elements will be included and the string in the associated data cell will be used as explanatory text. See also Reading set elements associated with Data or Text for instance.).

Note here the explanatory text of the set element `skipme2` is just a `Y` as it has a blank entry for the explanatory text.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample18**] for reference.

6.28.5.3 Reading Set Elements associated with Data or Text using the values Option

When reading set elements from spreadsheet, the values option can be used to control whether elements and associated text are included in the set or not. We use the data displayed in the spreadsheet below to demonstrate the `yn`, `dense`, `sparse` and `noData` specifications:

The set element names are stored in the first row, the associated data cells in the second row.

1. values=yn
Run the following command to load those element names associated with nonzero data or yes without storing the data as explanatory text:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1 values=yn
```

The set `A` will contain the elements `a`, `b`, `e`, `f`, `h`, `i`, `k` and `m`, since the elements `c`, `d`, `g`, `j` and `l` are associated with a zero, a blank or a no resp. N (case insensitive).

2. values=dense
This option must be specified, if we want to read all elements while using the strings in the data cells as explanatory text.

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1 values=dense
```

3. values=sparse
To read in all elements having a non-empty data cell while interpreting the string in the data cell as explanatory text, run the following command:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1 values=sparse
```

The set `A` will contain all elements except for `j`, since the associated data cell is empty.

4. values=noData
This option must be used, if we want to read all set elements while ignoring the data range. Especially, the data cells will not be interpreted as explanatory text. To read the elements from the range A1:M1, one could run the following command:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M1 cDim=1
```

While all elements `a` - `m` will be included this way, the second row will be automatically interpreted as explanatory text (even though we specified only the first row within the `rng` statement). This might be not desirable at all in some situations, e.g. when reading the city names in the next example Reading Set from Data Tables, we do not want to have the numbers 5000, 6000 and 0 to be explanatory text for the city names. Run the following command to include all elements in your set without interpreting the cells in the second row as explanatory text:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M1 cDim=1 values=noData
```

5. values=auto (default)
The second table within the description of the values options indicates which value type will be used by default based on the `rng`, `cDim` and `rDim` specifications for the set. For instance, when running the following command:

```
gdxxrw exampleData.xlsx set=A rng=readingWithValues!A1:M2 cDim=1
```

the value type used is `dense`, since we specified a block range with a data row and `rDim` equals zero.

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample9]** for reference.

6.28.5.4 Reading Set from Data Tables

One may wish to load set elements from a data table. Given a spreadsheet segment like the following:

We can take the set across the top containing the elements cleveland, chicago and dallas with any of the following commands:

```
gdxxrw exampleData.xlsx  set=i6  rng=readingSets!B20:D20 cDim=1 values=noData
gdxxrw exampleData.xlsx dSet=i6a rng=readingSets!B20:D20 cDim=1
gdxxrw exampleData.xlsx  set=i6c rng=readingSets!B20:D21 cDim=1 values=noData
```

Note the usage of the values option in order to avoid reading the numbers as explanatory text. See also Reading set elements associated with Data or Text. However, this can also be avoided by declaring the symbol as a domain set using the symbol declaration dSet.
We can also take a set vertically from column A as follows:

```
gdxxrw exampleData.xlsx dSet=j4 rng=readingSets!A21:A23 rDim=1
```

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample18**] for reference.

6.28.5.5 Reading Set from Lists with Duplication

One may wish to extract set elements from a spreadsheet where there is no unique list of elements that can be read but rather a list where some names are repeated. In the example below note that in rows 26 and 27 there are set element names but they are duplicated:

One can read this using dSet as follows:

```
gdxxrw exampleData.xlsx dSet=i7 rng=readingSets!B26:E26 cDim=1 dSet=i8 rng=readingSets!B27:E27 cDim=
```

Both domain sets will be read within a single `GDXXRW` call. The `rng` and `cDim` specifications affect only the symbol that they are following directly.

It may be favored in some situation to use the set symbol instead (e.g. for reading explanatory text). To oppress an error message when reading sets with duplication, one must specify a sufficient large number within the maxDupeErrors option.

```
gdxxrw exampleData.xlsx maxDupeErrors=4 set=i7 rng=readingSets!B26:E26 cDim=1 values=noData set=i8 r
```

For the data in this example, four is a sufficient large number since there are two duplicates for the first and two duplicates for the second set within each range. Note the usage of the values option in order to avoid reading 'cleveland' as explanatory text for the elements of set `i7` and to avoid reading the numbers as explanatory text for the elements of `i8`. See also Reading set elements associated with Data or Text for more informations about the `values` option.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample18**] for reference.

6.28.5.6 Reading Parameter from Spreadsheet

Assuming we want to read parameter `data1` from the file Test1.xlsx and write the data to Test1.gdx.

The following statement reads parameter `data1` from the spreadsheet above (using the par data type):

```
gdxxrw Test1.xlsx par=data1 rng=A1:D3 cDim=1 rDim=1
```

The sheet name in a range can be omitted when it refers to the first sheet. The elements in the first row and first column of the data range will be used as labels for the two dimensional parameter `data1` by defining `cDim=1` and `rDim=1` (see also cDim resp. rDim).

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample5]** for reference.

6.28.5.7 Reading Parameter from Spreadsheet with Duplication

The same data as in the previous example, but organized differently. We use the dSet symbol specification instead of set to read set `I` (in column A) and set `A` (in column B), since there are duplicate entries in column A resp. column B.

The following statement reads parameter `data2` from the spreadsheet above:

```
gdxxrw Test1.xlsx par=data2 rng=EX2!A1 rDim=2 dSet=I rng=EX2!A1 rDim=1 dSet=A rng=EX2!B1 rDim=1
```

By setting `rDim=2` for the parameter `data2` we indicate to use the first two columns of the data range as the labels for the parameter values. Since the sheet does not contain further data, one can specify the ranges using the top left cell notation without hesitation.
When using a few symbols, the command line can become too long to be practical. In such case, use a text file to hold the parameters. A parameter file can contain multiple lines to increase readability and a line starting with a '*' will be ignored.

```
* file example6.txt
par =data2 rng=EX2!A1 rDim=2
dSet=I     rng=EX2!A1 rDim=1
dSet=A     rng=EX2!B1 rDim=1
```

An option file is indicated by preceding the file name with a @ (At sign.).

```
gdxxrw Test1.xlsx @example6.txt
```

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample6]** for reference.

Note

- An option file can contain multiple lines to increase readability.
- When reading parameters from a text file, lines starting with an asterisk (*) will be ignored and act as a comment.
- An option file can also be written during the execution of a GAMS model using the GAMS Put Facility and the subsequent `GDXXRW` commands must use execute command so the put file is written before it is to be read (this would happen when using the compile time command `$call` to run `GDXXRW`).
- Using an option file can be useful in reducing execution time by combining the options and symbols from multiple `GDXXRW` calls in a single option file used in a single `GDXXRW` call.

6.28.5.8 Reading Multi-dimensional Parameter from Spreadsheet

This example illustrates how to read a four dimensional parameter from spreadsheet:

The strings in the first two columns and the first two rows of the data range A1:F6 shall be used as labels for the parameter values. Therefore, we define `rDim=2` and `cDim=2` (see also rDim and cDim). The parameter will be named `data3` (within the par declaration). Run the following command to read the data with `GDXXRW`:

```
gdxxrw Test1.xlsx par=data3 rng=EX3!A1:F6 rDim=2 cDim=2
```

Note that the data range was specified using the block range notation. However, it might be more comfortable to specify only the top left corner sometimes, but empty rows or columns may affect the reading process, i.e. `GDXXRW` might stop to early when encountering empty rows or columns or it will try to read data separated by empty rows or columns not being part of the data you wish to read. When we specify the range as a block, an empty row or column will always be ignored. When we specify the top left cell only, the skipEmpty option can be used to ignore one or more empty rows or columns. However, for the data in this example, we do not need to ignore empty rows or columns within the data range, but suppose there is non-relevant data starting in column H. By default (`skipEmpty=1`), `GDXXRW` would try to read the data starting in H. When we specify `skipEmpty=0` and cells A7, B7, G1 and G2 are empty, the range can be specified with the top left cell only in this example:

```
gdxxrw Test1.xlsx skipEmpty=0 par=data3 rng=EX3!A1 rDim=2 cDim=2
```

Since `skipEmpty` is a global option, affecting every symbol that follows, we define it before declaring the parameter `data3`.

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample7]** for reference.

6.28.5.9 Reading Spreadsheet using the Index Option

The index option is used to read a number of parameters and sets based on information stored in the spreadsheet itself. By doing this, the `GDXXRW` command becomes quit short and the informations on how to read the data can be written directly within the spreadsheet to increase readability. The first row of the range is used for column headings indicating additional parameters. We will discuss the results only briefly, because all other options used in this example were demonstrated before.

Suppose we want to read the parameters and sets from the following spreadsheet:

The informations about all parameters and sets are stored within the sheet index in the same spreadsheet file exampleData.xlsx as testIndex above:

The following statement reads parameters and sets from the spreadsheets using the `index` option:

```
gdxxrw exampleData.xlsx output=gdxAll.gdx index=index!A1
```

We use the `output` option to write the data to gdxAll.gdx for demonstration here. By default, `GDXXRW` would write to testIndex.gdx. Some brief remarks on the results:

- Elements of set i1 and i1a: trains, cars and planes. There is no explanatory text, since the data range (third row) is empty. We can use dim=1 equivalently to cDim=1 in this example, because by default, cDim=1 and rDim=dim-1. See also dimensions.
- When reading the sets i2 and i3, we are defining which elements and explanatory text will be stored by specifying the values option.
- The set i4 will contain all elements of the thirteenth row. Afterwards, skipEmpty is set to zero, affecting the reading of i4a and i5. Unlike i4, the sets i4a and i5 will not contain the element houston, since skipEmpty=0 and the empty column signals GDXXRW to stop reading (note that the range is not defined using the block range specification). Since there is a data range for i5, the strings city1 - city5 will be used as explanatory text by default. After reading i4a and i5, skipEmpty is reset to one (default).
- Each of the sets i6, i6a and i6b will contain the elements cleveland, chicago and dallas. There is no explanatory text for any of the sets in the GDX file, however, one must enforce this for the set i6 with values=noData.

There are no interesting details for the remaining sets and parameters to discuss here. The complete example and the results displayed within GAMS can be found in the GAMS Data Utilities Library, see model **[GDXXRWExample10]** for reference.

Note

- The parameters and sets are read using the specifications within the myIndex sheet. They are treated as if they appeared directly on the command line.
- In the spreadsheet, the first three columns of the range have a fixed interpretation: DataType (par, set, dSet, equ, or var), Item name identifier and spreadsheet data range. The fourth and following columns can be used for additional parameters like dim, rDim, cDim, merge, clear and skipEmpty. The column header contains the keyword when necessary, and the cell content is used as the option value.
- When an entry appears in a column without a heading then it is directly copied into the GDXXRW option file. Thus in the example above the items in column G are directly copied into the file.
- Rows do not need to have entries in the first three columns if one just wants to enter persistent options such as skipEmpty or some of the special character string re-definitions (as in row seven and ten from the spreadsheet above).

6.28.5.10 Reading Data from Spreadsheet and Loading into GAMS

One can use $call to execute the GDXXRW command in the GAMS code to read from spreadsheet at compilation time (the data is taken from the previous example):

```
$call gdxxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData
```

Getting a set from the spreadsheet into a GDX file is only half the battle. One must also use commands in GAMS to load the data as discussed in the chapter Using GAMS Data Exchange or GDX Files. At compile time this is done using:

```
Set i9;
* read/load set from data at compile time
$call gdxxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData
$gdxIn testIndex.gdx
$load i9
```

where the set must be declared in a set statement then one can if needed create the GDX file using `GDXXRW`, then one uses a `$gdxIn` to identify the source file and a `$load` to bring in the data.

Some users may wish to load sets at execution time. However, this is limited to subsets that are dynamic sets and cannot be used in domains. To do this one simply uses the statements as above, but substitutes `execute` in place of `$call` as follows:

```
Set i9(i6a);
* read/load set from data at execution time
execute 'gdxxrw testIndex.xlsx set=i9 rng=Sheet1!B20:C20 cDim=1 values=noData'
execute_load 'testIndex' i9;
```

where the set `i9` must be declared as a subset in a set statement (of `i6a` in this case), then one can if needed create the GDX file using execution time `GDXXRW`, and an `execute_load` to bring in the data with an identification of the GDX source file name. Note that we used the set `i6a` as superset, fitting best to the data from the previous example. However, one could also use the statement `Set i9(*);`.

One can load the universe of labels from a GDX file into a set at run-time using the syntax:

```
execute_load 'someFile', someSet=*;
```

Note

In doing this, only labels known to the GAMS program will be loaded.

6.28.5.11 Reading empty Cells with colMerge

Suppose we want to read the four dimensional parameter from the following spreadsheet:

The cells B4, B5, C4 and D5 might be empty to avoid duplication, i.e., the non-empty content of the previous cell in the same column shall be used as content for the empty cell. In particular: the content of B3 shall be used for the content of B4 and B5, the content of C3 for C4 and the content of D4 for cell D5. Reading the above spreadsheet using the following GAMS statement:

```
gdxxrw exampleData.xlsx par=A_d rng=colMerge!B2 rDim=3 cDim=1
```

results in empty cells B4, B5, C4 and D5, causing troubles if you want to declare the parameter as `A(number,number,number,color)` certainly:

Adding the symbol attribute colMerge, we use the non-empty content of the previous cell in the same column as the content for the empty cell. Specifying `colMerge=2` will do this for the first two columns for instance.

```
gdxxrw exampleData.xlsx par=A_2 rng=colMerge!B2 rDim=3 cDim=1 colMerge=2
```

Only the two entries corresponding to the cell D5 are still empty, since we do not specify all three columns within `colMerge`:

Note

A blank field displayed in GAMS Studio indicates an empty UEL. In the GAMS IDE, there would be an <empty> entry instead.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample20**] for reference.

6.28.5.12 Reading merged Excel Ranges with cMerge

Suppose we want to read the three dimensional parameter from the following spreadsheet:

Note that the label 'red' is centered over the merged cells B1, C1 and D1 and label 'green' over the merged cells E1 and F1. Additionally, in the data range, the cells B3:C3 and B4:C4 are merged, too. The option cMerge can be used to control the way merged cells are handled. We will discuss the effect of the possible values for `cMerge` on the data presented above by running the following commands (an option file is used to increase readability):

```
$onEcho > howToRead.txt
cMerge=0 par=B_d rng=cMerge!A1 rDim=1 cDim=2
cMerge=1 par=B_1 rng=cMerge!A1 rDim=1 cDim=2
cMerge=2 par=B_2 rng=cMerge!A1 rDim=1 cDim=2
$offEcho

$call gdxxrw exampleData.xlsx output=cMerge.gdx @howToRead.txt
```

We specify `cMerge` in advance, since it is a global option affecting every symbol that follows. Executing the three statements will create three different output files, all displayed in GAMS Studio:

Note

A blank field displayed in GAMS Studio indicates an empty UEL. In the GAMS IDE, there would be an <empty> entry instead.

Some remarks on the results:

cMerge=0 (default)
Empty cells being part of a merged Excel range will remain empty. Thus, the cells C1, D1, F1, C3 and C4 from the spreadsheet above will remain empty when reading with `GDXXRW`. Since C3 and C4 are empty while being part of the data range, they won't show up in the GDX file. The values in the last column of the GDX file are useful to compare the results with the spreadsheet. For instance, the UELs of the value `5` are `a`, <empty> and `two`, since the cell F1 is empty, while A3 contains the string `a` and F2 the string `two`.

cMerge=1
The value of a merged range within a row or column header will be used for all cells being part of the merged range. Thus, the string 'red' will be used for the cells in the column header C1 and D1 and the string 'green' for the cell F1. Since C3 and C4 are part of the data range, they will remain empty and are not displayed in the GDX file. As you can see, there is no longer an empty UEL.

cMerge=2
The value of a merged range will be used for all cells being part of the merged range, i.e. `cMerge=1` is extended to the data range. Therefore, the value 1 resp. 11 will be also used for the cell C3 resp. C4, appearing in the GDX file for the first time. Since there is no change in handling merged cells within the row or column header, all values have non-empty labels.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample20**] for reference.

6.28.5.13 Skipping Empty Rows and Columns

By using the skipEmpty option, we can control the way blank rows or columns are handled and causes `GDXXRW` to either stop or skip over if a blank row or column is encountered when using the top left corner range specification instead of a block range. If the range is specified using the `TopLeft:BottomRight` cell notation (often refered as block range notation), empty rows or columns will be skipped automatically. Suppose the data is stored in the following spreadsheet:

We can read this spreadsheet and skip blank rows and columns with the following command:

```
gdxxrw exampleData.xlsx      par=A_d rng=skippingRC!A2 rDim=2 cDim=1
```

or

```
gdxxrw exampleData.xlsx se=1 par=A_1 rng=skippingRC!A2 rDim=2 cDim=1
```

Note that there will be no difference concerning the generated GDX files, since `skipEmpty` is set to one by default. On the other hand, if `skipEmpty` is set to zero

```
gdxxrw exampleData.xlsx se=0 par=A_0 rng=skippingRC!A2 rDim=2 cDim=1
```

the blanks terminate the read not reading the `rail` column and the `san francisco.chicago` row. After loading into GAMS the data become:

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample21**] for reference.

Note

> The `skipEmpty` option must appear before any `par`, `set`, `dSet` etc statements that use it and will persist for the rest of the statements in a command unless it is set to another value.

6.28.5.14 Ignoring Rows and Columns

We can use the options ignoreColumns and ignoreRows to ignore columns and rows when reading data for a symbol. Suppose we want to ignore the red colored rows and columns of the following spreadsheet, when reading the four dimensional parameter:

We can read this spreadsheet and ignore the red colored columns and rows with `GDXXRW` by running the following command:

```
gdxxrw exampleData.xlsx par=A rng=ignoringRCC1 cDim=2 rDim=2 ignoreRows=2,6 ignoreColumns=D,G
```

The options `ignoreRows` and `ignoreColumns` are symbol options and therefore must appear after the symbol specification, affecting only this particular symbol.

In the example above we ignored column D which would have been part of the index for the rows. So the range for the row index was extended with column E. The E column is no longer part of the data range. The treatment of the column index is similar. The second row would have been part of the column index, and now that the row is ignored, the next row becomes part of the column index and the third row is no longer part of the data range.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample21**] for reference.

Note

The `ignoreColumns` and `ignoreRows` options appear after any `par`, `set`, `dSet` etc `GDXXRW` command instruction and only affect reading of that item.

6.28.5.15 Reading Parameter from Spreadsheet using pre-defined Excel Named Ranges

As mentioned in section Excel Ranges, the range of a symbol to be read can be defined by using named ranges. To name a cell range within Excel, simply select the cell range and type in the name you want to assign to this particular range by using the name-box next to the top left corner of your spreadsheet. Suppose we want to read the data in the range A1:D3 taken from the example Reading Parameter from Spreadsheet:

Instead of specifying the range explicitly by `rng=A1:D3`, we use the pre-defined named range 'parRange', i.e. `rng=parRange`. `GDXXRW` uses the string specified to search for a matching pre-defined named range first. In summary, run the following command to read the parameter `data4` from the file Test1.xlsx:

```
gdxxrw Test1.xlsx par=data4 rng=parRange rDim=1 cDim=1
```

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample5**] for reference.

Note

In Excel, one can assign a single name to several separated block ranges, e.g. assign the name 'disconnected' to the block ranges A1:D3 and F4:H5 (by holding 'Strg' while selecting the second block range and using the name-box to assign the name). However, such disconnected data cannot be read using the named-range specification.

6.28.6 Writing to Spreadsheet - Examples:

6.28.6.1 Unloading Data from GAMS before Writing to Spreadsheet

When writing to spreadsheet with GDXXRW, one must use commands in GAMS to place the data into the GDX file at first (see also Using GAMS Data Exchange or GDX files). When reading data, it is often desirable to use a $call command to run GDXXRW and the statements $gdxIn and $load afterwards to load the data, allowing also domain definitions (at compile time) for instance. This is hardly ever desirable or realizable when unloading and writing to a spreadsheet, for example, if one wish to write the solution after solving a model to spreadsheet. This should generally not be done at compile time so one should only use the execute and execute_unload commands at execution time as follows:

```
execute_unload 'test.gdx' someParameter;
execute 'gdxxrw test.gdx par=someParameter rng=A1'
```

where the execute_unload tells what data to place in the GDX file and determines the GDX source file name. The matching GDXXRW execution tells the name of the GDX file, the name of the spreadsheet (optional) and identifies the data to write.

See also Writing Set to Spreadsheet and Writing Parameter to Spreadsheet for demonstration.

Note

- One must be careful when using GDXXRW as each time the command is executed the GDX file is erased and only has the current contents and thus should be written just before if reusing the name.
- One also obtains output of sets using the command execute_unloaddi where the GDX file automatically includes all sets associated with unloaded parameters, variables and equations without need to list the set element names.

6.28.6.2 Writing Set to Spreadsheet

At fist, we will create a GDX file containing a simple set using the execute and execute_unloading directives. Most of the elements have an explanatory text:

```
Set x / element1 'explanatory text'
        element2
        element3 'previous element does not have explanatory text' /;

execute_unload 'writingSet.gdx' x;
```

Of course, in this particular code section above, one could also use $gdxOut and $unload. The values option can be used to control whether explanatory text is written to the spreadsheet. We'll demonstrate all three possible values explicitly within a single GDXXRW execution:

```
$onEcho > howToWrite.txt
hText="values: noData,,,values: yn,,,values: string" rng=Sheet1!A1
set=x rng=Sheet1!A2 rDim=1 values=noData
set=x rng=Sheet1!D2 rDim=1 values=yn
set=x rng=Sheet1!G2 rDim=1 values=string
$offEcho

execute 'gdxxrw writingSet.gdx output=writingSet.xlsx @howToWrite.txt'
```

Before executing this example, check if the Excel file (writingSet.xlsx) is open. If you run `GDXXRW` for writing a file sharing conflict will arise. To avoid this problem, either close the Excel file or use the Excel Tools menu to make this a shared notebook. After writing to the spreadsheet (still opened), use the Excel "File Save" command to verify the changes made.

By adding two additional commas within the hText statement, the cells B1, C1 and E1, F1 will be skipped when writing the text to the first row of the spreadsheet. If `values=noData`, neither explanatory text nor a `Y` are written to spreadsheet for the set elements. If `values=yn`, `GDXXRW` writes a `Y` for each set element to spreadsheet. To write the explanatory text, specify `values=string`.

Since `cDim=0`, the default option is `string` (see values). Imagine a two dimensional set, one could write the set in a table format, i.e. `cDim=1` and `rDim=1`. By default, `GDXXRW` would write this set using the `values=yn` format.

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample11b]** for reference.

Note

- A workbook cannot in general be open unless you have made special provisions with an error signaled indicating a file sharing conflict will arise when the target file is open in Excel.
- To avoid the sharing conflict error the user must either close the file or indicate that the spreadsheet is a shared Excel workbook by using the Excel Tools Share Workbook dialogue.
- In an open shared workbook the contents are not updated until you have done a file save in Excel.
- Writing to a shared workbook can be painfully slow.
- In general, it is best to close the workbook.

6.28.6.3 Writing Parameter to Spreadsheet

At first, we create a GDX file, containing some random data:

```
* file makeData.gms
Set
   i / i1*i4 /
   j / j1*j4 /
   k / k1*k4 /;

Parameter V(i,j,k);
V(i,j,k)$(uniform(0,1) < 0.30) = uniform(0,1);
```

When we run this GAMS model from the command prompt using the following statement, the file writingPar.gdx will be created at the end of the run.

```
GAMS makeData gdx=writingPar
```

Using the file writingPar.gdx, we can write to a spreadsheet:

> Write parameter V to the first cell in the first sheet; because we only specify the top left corner of the sheet, the complete sheet can be used to store the data. We do not specify the row and column dimension, so they will be set to rDim=2 and cDim=1 by default. (See also dimensions)

By using the following command (remember to close an already existing file writingPar.xlsx in advance or make it a shared notebook as discussed in the previous example):

```
gdxxrw writingPar.gdx output=writingPar.xlsx par=V rng=A1
```

The steps above can be combined in a single GAMS model using the execute_unload and execute statements as follows:

```
Set
   i / i1*i4 /
   j / j1*j4 /
   k / k1*k4 /;

Parameter V(i,j,k);
V(i,j,k)$(uniform(0,1) < 0.30) = uniform(0,1);

execute_unload 'writingPar.gdx', i, j, k, V;
execute 'gdxxrw writingPar.gdx par=V rng=A1';
```

The resultant spreadsheet looks like:

Note that if we only want to write the parameter V, there is no need to unload the sets i, j and k explicitly. The labels written to the columns A and B and to the first row are stored directly together with V in the GDX file.

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample11b]** for reference.

6.28.6.4 Writing to Spreadsheet with merge Option

When writing to a spreadsheet one can control data handling and matching using the merge command line option. When `merge` is active, the only data that will be written to the spreadsheet are those data for which the element names match row and column labels that are in the spreadsheet already. Also under `merge`, spreadsheet cells for which there is no matching row/column pair will not be changed. This option might be very useful, e.g., if there is a fixed report layout/framework already in your spreadsheet file which should not be changed when writing the data from GAMS.

Suppose we want to write the parameter `A`, already stored in data.gdx, to an existing spreadsheet:

In the following spreadsheet, there are row and column labels matching most of the data in the GDX file, except for the additional column header 'horse' and the non existing row labels 'san francisco.chicago':

Use the following commands to write the data from GDX to spreadsheet twice to different ranges for comparison:

```
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!B1:G4 rDim=2 cDim=1 merge
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!B8    rDim=2 cDim=1
```

Note that `merge` is a symbol option affecting only the symbol `A`. The resultant spreadsheet looks like:

The parameter is written to the range B8-F12 without `merge` enabled, while the option is enabled when writing to the range B1:G4, respecting the data arrangement already existing. Note that the column and row orders vary and the san francisco - chicago row is missing since it is not mentioned in the labels within the spreadsheet before the merge operation, while the horse column is still present with it's data left alone, not being overwritten by the parameter.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample12**] for reference.

Note

- Using the `merge` option will force the data to be presented in the order in which the row and column labels are entered already.
- GDX file contents that do not have matching row/column pair of named elements in the spreadsheet will be overlooked.
- A write under a `merge` option addressing a blank area of a spreadsheet will always be blank as there will not be matching set elements at all.
- The matching of labels is not case-sensitive.
- Warning: Enabling the `merge` option will clear the Excel formulas in the rectangle used, even if the cells do not have matching row/column headings in the GDX file. Cells containing strings or numbers are not affected.

6.28.6.5 Writing to Spreadsheet with clear Option

When writing to a spreadsheet one can also use the clear option to control data handling and matching. When `clear` is enabled, the only data that will be written to the spreadsheet are those data for which the element names match row and column labels that are in the spreadsheet already but all data and formulas in the target range will be removed.

Suppose the parameter `A` from the previous example is stored in data.gdx and there are row and column labels matching most of the data in the GDX file, except for the additional column header 'horse' and the non existing row labels 'san francisco.chicago':

Use the following command to write to exampleData.xlsx with `clear` enabled:

```
gdxxrw data.gdx output=exampleData.xlsx par=A rng=merge_clear!I1 rDim=2 cDim=1 clear
```

then the result is

The results are similar to those under merge but the old data in the column labeled 'horse' has been removed.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample12**] for reference.

Note

- Using the `clear` option will force the data to be presented in the order in which the row and column labels are entered already.
- GDX file contents that do not have matching row/column pair of named elements in the spreadsheet will be overlooked.
- A write under a `clear` option addressing a blank area of a spreadsheet will always be blank as there will not be matching set elements.
- The matching of labels is not case-sensitive.
- Warning: The `clear` option will clear all Excel formulas and values in the rectangle used, even if the cells do not have matching row/column labels in the GDX file.

6.28.6.6 Writing to Spreadsheet using a Filter

In Excel, you can filter the data by some specified conditions so that only the data matching the conditions is displayed. This might be useful in some cases, e.g. it helps you to focus on the most relevant data within a large table of data. With `GDXXRW` you can add some basic filter to your spreadsheet when writing data from a GDX file.
The following example creates a small GDX file with some random data, which is used to write the symbol `A` to a spreadsheet later on with the filter option enabled.

```
Set
   i / i1*i2 /
   j / j1*j2 /
   k / k1*k2 /;

Parameter A(i,j,k);
A(i,j,k) = uniform(0,1);

execute_unload 'test.gdx', A;
execute 'gdxxrw test.gdx filter=1 par=A rDim=1 cDim=2 rng=Sheet1!A1';
```

Since `filter` is a global option, it must be specified in advance of the symbols for which you want to add a filter. The default value is zero, i.e. no filter will be added. If there are multiple rows in the column header, i.e. `cDim` is greater than zero, the valid range for the filter option is `1..cDim`.

The screenshot above shows the filter in Excel. When we specify `filter=2` in this example with two dimensions for the column header, the row with the filter moves away from the data range as illustrated below:

One could now filter the data, e.g. displaying only the values where the label of the first dimension is `i1` by selecting this value exclusively within the drop down menu of column A.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample15**] for reference.

6.28.6.7 Writing to Spreadsheet adding Text and Hyperlinks

The following example illustrates the use of the text directive. Adding text and hyperlinks to your spreadsheet is useful to customize the output and to navigate more quickly through the data.

First, we write some data to a GDX file and we use the text directive to write text to various cells in the spreadsheet; some of the cells are hyperlinks to other locations. To increase readability, we'll use a parameter file howToWrite.txt to shorten the `GDXXRW` statement.

```
Set
   i / i1*i9 /
   j / j1*j9 /;

Parameter A(i,j);
A(i,j) = 10*ord(i) + ord(j);

execute_unload 'pv.gdx' A;

$onEcho > howToWrite.txt
text="Link to data" rng=Index!A2 linkID=A
text="Below the data for symbol A" rng=data!C2
par=A rng=data!C4
text="Back to index" rng=data!A1 link=Index!A1
text="For more information visit GAMS" rng=data!C1 link=http://www.gams.com
$offEcho

execute 'gdxxrw pv.gdx output=pv.xlsx @howToWrite.txt'
```

We will write the text "Link to data" to the cell A2 of sheet Index. The option `linkID` is used to add a hyperlink to the range of the symbol `A`. In addition, we create a hyperlink "Back to index" in the cell A1 of sheet data using the `link` option pointing to the cell A1 of sheet Index. One can also specify links to external sources. For demonstration, we add a link to the GAMS homepage.

Below a screen shot showing both sheets data and Index created by the commands above:

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample16**] for reference.

6.28.7 Reading and Writing, Extended Use - Examples:

6.28.7.1 Reads a Table from Spreadsheet, manipulates the Data and writes back to Spreadsheet

In the following example, we read data from a spreadsheet and save the data in a GDX file. Using the `$gdxIn` and `$load` GAMS directives, we load the data from the GDX file into GAMS afterwards. The GAMS program modifies the data and at the end of the run the data is saved in a new GDX file (tmp.gdx). The last step updates the spreadsheet with the modified parameter.

We read the spreadsheet and load the data from the resultant GDX file at compile time. The data modification and the unloading and writing process are done at execution time, using the `execute_unload` and `execute` directives:

```
$call gdxxrw test1.xlsx dSet=I rng=A2:A3 rDim=1 dSet=A rng=B1:D1 cDim=1 par=X rng=A1:D3 rDim=1 cDim=
$gdxIn test1.gdx

Set I(*), A(*);
$load I A

Parameter X(I,A);
$load X

display I, A, X;
$gdxIn
X(I,A) = - X(I,A);

execute_unload 'tmp.gdx', I, A, X;
execute 'gdxxrw tmp.gdx output=test1.xlsx par=X rng=EX6!A1:D3 rDim=1 cDim=1';
```

The parameter is written to the sheet EX6. However, since we only write the parameter `X`, we do not necessarily have to unload the set `I` and `A`.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample13**] for reference.

6.28.7.2 Reading Data from Spreadsheet and writing Data to Spreadsheet after Solve

In this example, we use a modified version of the [**trnsport**] model from the GAMS model library to demonstrate the process of reading data, defining and solving the model and writing a solution report to spreadsheet altogether. This example illustrates in particular:

- Compilation phase
- Read data from a spreadsheet and create a GDX file
- Loading sets from the GDX file
- Using the sets as a domain for additional declarations
- Reading additional data elements

- Execution phase
- Solve the model
- Write solution to a GDX file
- Use GDX file to update spreadsheet

A parameter file howToRead.txt is used to increase the readability of the GDXXRW call. Note that the dimension of the scalar we want to read is set to zero. The range of each parameter can be specified by using the top left corner only, since there are two empty rows separating the data blocks from each other and the default value of skipEmpty signals to stop reading if two empty rows occur.

```
$onEcho > howToRead.txt
dSet=i rng=A3:A4 rDim=1
dSet=j rng=B2:D2        cDim=1
par =d rng=A2    rDim=1 cDim=1
par =a rng=A8    rDim=1
par =b rng=A13   rDim=1
par =f rng=A19    dim=0
$offEcho
$call gdxxrw TrnsportData.xlsx @howToRead.txt
$gdxIn TrnsportData.gdx

Set
   i(*) 'canning plants'
   j(*) 'markets';

$load i j

display i, j;

Parameter
   a(i)   'capacity of plant i in cases'
   b(j)   'demand at market j in cases'
   d(i,j) 'distance in thousands of miles';

Scalar f 'freight in dollars per case per thousand miles';
$load d a b f
$gdxIn

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

Variable
   x(i,j) 'shipment quantities in cases'
   z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
   cost      'define objective function'
   supply(i) 'observe supply limit at plant i'
   demand(j) 'satisfy demand at market j';

cost..      z =e= sum((i,j), c(i,j)*x(i,j));
supply(i).. sum(j, x(i,j)) =l= a(i);
demand(j).. sum(i, x(i,j)) =g= b(j);
```

```
Model transport / all /;

solve transport using lp minimizing z;

display x.l, x.m;

execute_unload 'TrnsportData.gdx', x;
execute 'gdxxrw TrnsportData.gdx output=TrnsportData.xlsx squeeze=n var=x.l rng=Sheet2!A1';
```

The solution is written to Sheet2 of the input file TrnsportData.xlsx by executing GDXXRW at execution time. The var statement is used in the symbol specification to write out the level of variable x. In order to write zero values, the squeeze option is disabled. Otherwise, the cells C4 and D3 remain blank.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample14**] for reference.

6.28.7.3 Reading Special Values from Spreadsheet and writing to Spreadsheet

This example demonstrates the reading and writing of special values with GDXXRW.

Reading Special Values
Assuming we want to read the special values stored in the spreadsheet below:

The following statement reads parameter vIN from the spreadsheet above:

```
gdxxrw exampleData.xlsx output=specialValues.gdx NaIn=N/A squeeze=n par=vIN cDim=1 rng=specialValues
```

To affect the parameter vIN, the global options NaIn and squeeze (to display zero values in the GDX file) must be specified in advance.

The cells containing the strings Eps, +Inf, -Inf, Inf, NA and Undf are read in correctly. The division by zero error in the spreadsheet will be written as Undf. We defined a new string - 'N/A' - within the option NaIn to be recognized as NA additionally. Within GAMS, there is no directly comparable data type for 'None' and 'Null', so these strings will be interpreted as Undf. Note the importance of the cell format specified within Excel. For instance, the values for v13 and v14 are different, although both fields contain a Dollar sign, since the cell format of M2 is 'General', while the cell format of N2 is 'Currency'.
The boolean 'False' turns into zero after reading, while 'True' turns into -1. Note here, that this is independent whether the booleans are written as plain text within Excel or by using the formulas '=True()' resp. '=False()'. We disabled squeeze in order to display the zero values for the elements v16, v18 and v19 in the GDX file.

Writing Special Values
Initially, we declare a parameter with special values and generate a GDX file from it:

```
$onUndf
Set v / v1*v7 /;
Parameter vOUT(v) / v1 Eps, v2 +Inf, v3 -Inf, v4 Inf, v5 Undf, v6 0.0, v7 NA /;
Scalar S / 0 /;

execute_unload 'specialValues.gdx', v, vOUT, S;
```

We will write the parameter vOUT to spreadsheet twice. At first to demonstrate the usage of EpsOut, pInfOut, mInfOut, UndfOut, zeroOut and NaOut, the second time to show the default settings when writing. Those output string options may be useful, if the GAMS default settings are not appropriate for your Excel calculations later on or to customize the representation of the values in Excel in general. Note that there is also a scalar declaration in order to demonstrate the different behavior when writing scalars and parameter with zero values to Excel while using the zeroOut option.

```
$onEcho > howToWrite.txt
* defining new strings to be used when writing special values:
EpsOut=0 pInfOut=+1E+100 mInfOut=-1E+100 UndfOut=undefined zeroOut=zero NaOut=notAvailible

* now write parameter vOUT with merge to force the column F containing set element "v6" and vOUT("v6"
set=v    rng=specialValues!A6:G6 cDim=1
par=vOUT rng=specialValues!A6:G7 cDim=1 merge
text="Special values of Parameter vOUT written with user defined output strings:" rng=specialValues!A
text="Scalar S / 0 /:" rng=specialValues!I6
par=S rng=specialValues!I7

* reset the strings for special values back to default und write vOUT again
resetOut
set=v    rng=specialValues!A10:G10 cDim=1
par=vOUT rng=specialValues!A10:G11 cDim=1 merge
text="Special values of Parameter vOUT written with default output strings:" rng=specialValues!A9
text="Scalar S / 0 /:" rng=specialValues!I10
par=S rng=specialValues!I11
$offEcho

execute 'gdxxrw specialValues.gdx output=exampleData.xlsx @howToWrite.txt';
```

In order to increase readability when executing GDXXRW, we use a parameter file named howToWrite.txt and additionally, we write some text out to structure the Excel file. The range for the non-default values is A6:G7. We define the new strings to be used for the special values first, affecting the following symbols. To write the default values to the range A10:G11, we use the resetOut option to reset the output strings to default, otherwise, the new strings remain in effect, since they are global options.

As mentioned briefly above, the zeroOut option affects the scalar S and the parameter vOUT differently. While we get the expected result - 'zero' - for the scalar S, the cell F7 for the zero value of vOUT remains empty, since zero values of parameters are not even part of the GDX file in general (and cannot be added from your GAMS model; Note here, that the squeeze options only affects the writing of sub-fields of variables and equations). Therefore, cell F11 has no value, too. Though scalars with zero values are stored in the GDX file. We'll present a workaround for writing zero values of a parameter to spreadsheet in the next example Writing Parameter to Spreadsheet including Zero Values.

The complete example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample8]** for reference.

Note

- When writing to a spreadsheet, special values such as Eps, NA, Undf and Inf will be written but this can be changed. When reading data from a spreadsheet, the ASCII strings for these special character strings will be used to write corresponding special values to the GDX file.
- Cells that are empty or zero will not be written to the GDX file.

6.28.7.4 Writing Parameter to Spreadsheet including Zero Values

There is no straight way to write zero values of a GAMS parameter to spreadsheet from your model using GDXXRW, since zero values of a parameter are not stored in the GDX file (see also the previous example Reading Special Values from Spreadsheet and writing to Spreadsheet). However, instead of the zero values one can store an EPS in the GDX file and instruct GDXXRW to use a zero when writing the value for EPS using the epsOut option afterwards as demonstrated below:

```
Set i / i1*i9 /;
Parameter A(i), Amod(i);
A(i) = uniformInt(0,1);

* Amod(i) = A(i) if A(i) <> 0 and Amod(i) = EPS if A(i) = 0
Amod(i) = EPS$(not A(i)) + A(i);

* Unload the unmodified and modified parameter and write to spreadsheet using an option file
execute_unload 'zeroPar1.gdx' A Amod;

$onEcho > howToWrite1.txt
text="Parameter A" rng=A1
par=A rng=A2
text="Modified Parameter Amod written with epsOut: 0" rng=A5
epsOut=0 par=Amod rng=A6
$offEcho
execute 'gdxxrw zeroPar1.gdx output=writingZeros.xlsx @howToWrite1.txt';
```

This approach is impracticable in the unlikely event that there are already EPS values in your parameter and you want to write these as EPS to your spreadsheet.

An alternate approach regarding variables while exploiting the squeeze option is demonstrated below (parameter A and set i refer to the data above):

```
Variable dummyPar(i);
dummyPar.l(i) = A(i);

* In order to write every entry of dummyPar in the spreadsheet, one must allocate
* a non-zero value to one of the variable attributes .m, .lo or .up
dummyPar.up(i) = 1;

* Unload the dummy variable and write the .l subfield to spreadsheet while disabling squeeze
execute_unload 'zeroPar2.gdx' dummyPar;

$onEcho > howToWrite2.txt
text="Variable dummyPar written with squeeze: n" rng=A9
squeeze=n var=dummyPar.l rng=A10
$offEcho
execute 'gdxxrw zeroPar2.gdx output=writingZeros.xlsx @howToWrite2.txt';
```

Creating the additional variable and the allocation of a non-zero value to one of the other variable attributes are the drawbacks of this approach. Note that we only declared the parameter Amod to keep the original data of A untouched in order to run the code in a single model and to write A to spreadsheet, too.

This example is also part of the GAMS Data Utilities Library, see model [**GDXXRWExample19**] for reference.

6.28.7.5 Reading several Scalars from Spreadsheet

Suppose you want to read a large number of scalars (i.e. scalar names and their associated values) for your model from a spreadsheet file as shown below. Originally, the data was taken from the model **[indus89]** of the GAMS Model Library.

Natively, one could declare all scalars directly inside the model and read every single one of them with GDXXRW afterwards (the following code is shortened for representation):

```
Scalar baseyear, repco;

$call gdxxrw exampleData.xlsx output=indus89Scalars.gdx par=baseyear rng=indus89Scalars!B2 dim=0 par
$gdxIn indus89Scalars.gdx
$load baseyear repco
```

However, the code would become quit long and circumstantially for this example, especially the GDXXRW statement, because every scalar must be specified individually. Therefore, we present a more sneaky way to tackle the problem. At first, the file mkScalar.gms is written. We declare a set and a parameter to hold the scalar names and their values. A simple GDXXRW call is used to read the names and values from the file exampleData.xlsx. Afterwards, we load the data using $gdxIn and $load. The result of this is a single parameter containing the scalar values defined over a set containing the scalar names. However, this is not exactly what we are looking for. To transform this representation into several single scalar definitions, we use a simple loop over the set scalarNames and the Put Writing Facility to generate a file scalars.gms, containing those single scalar statements.

```
$onEchoV > mkScalar.gms
Set scalarNames;
Parameter scalarValues(scalarNames);

$call gdxxrw exampleData.xlsx output=indus89Scalars.gdx set=scalarNames rng=indus89Scalars!A2:A22 rD
$ifE errorLevel<>0 $abort Problems reading sheet indus89Scalars with GDXXRW
$gdxIn indus89Scalars.gdx
$load scalarNames scalarValues

File fs / 'scalars.gms' /;
put  fs;
loop(scalarNames, put / 'Scalar ' scalarNames.tl ' / ' scalarValues(scalarNames):>20:10 ' /;';);
$offEcho

$call gams mkScalar
$include scalars.gms
```

Calling the file mkScalar.gms will create the file scalars.gms, containing all those scalar statements (shortened for representation):

```
Scalar baseyear     /      1988.0000000000 /;
Scalar repco        /         2.5000000000 /;
```

Finally, the file scalars.gms is included to your model.

This example is also part of the GAMS Data Utilities Library, see model **[GDXXRWExample17]** for reference.

6.28.8 Changes in the Set Values Parameter

The following documents some changes that were made when reading a set using the `values=string` option. Reading a domain or a parameter was not affected by these changes.

To illustrate the various behaviors in different versions of `GDXXRW`, we are using the spreadsheet data as shown below, using the following call:

```
$call gdxxrw test.xlsx set=one rng=B2 rDim=1 values=string set=two rng=B1 rDim=1 cDim=1 values=string
```

We read the one dimensional set in column B by specifying the top-left corner of the data (cell B2) or the full range (B2..B5). A two dimensional set is read using the top-left corner of the data (cell B1) or the full range (B1..E5). Variations are introduced by specifying options for `values` to be `string` or `yn` or `noData`. The option value `all` is only available in later versions of `GDXXRW` and was used to introduce the same behavior as `strings` in earlier versions.

6.28.8.1 Chronological description of the changes made to the Values option:

- **GAMS versions prior to version 24.3:**

  `values=string` results in reading the data dense. The contents of a cell is used for the set associated text and an element is included even if the data cell is empty.

- **GAMS version 24.3.1**

  We changed the interpretation of `string` to mean that the set element was only to be included when the string data was not empty. Note below that element a4 is missing from the one-dimensional set and so are a1.b1, a2.b2 etc from the two-dimensional set. For the two-dimensional case this looked more or less how the GAMS compiler interprets a table statement. Unfortunately, the interpretation of empty data cells was also applied to one-dimensional sets leading to undesired results.

- **GAMS version 24.4.1**

  Recognizing that reading a set dense was no longer available, we introduced a new option `values=all`. This allowed us to read Excel data the same way as was possible before version 24.3.1 using the `values=string` i.e. reading the data dense and including all cells whether the data cell is empty or not and use the content of the data cell for the set associated texts.

- **GAMS version 24.4.6 (Current status)**

  We decided to remove some confusion with the interpretation of the `values` option by introducing the options `dense` and `sparse` and flagging the options `strings` and `all` as deprecated. Both `strings` and `all` are replaced with `dense`.

Backward compatibility issues.

With these changes we broke our in house rule not to introduce changes that break backward compatibility. Because of this, the user needs to change the parameters for the `GDXXRW` call or change the workbook data. The parameters for the call should be changed from `values=string` to `values=dense` for one-dimensional sets where we specify the top-left corner only. In case the data in the workbook needs to be changed, inserting a string to in the data cell will address the issue. In the example on the top of the page, inserting a 'Y' in cell C5.

6.29 Invert

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This calculates the inverse of a square matrix A: `A*AInv = I`. The matrices `A` and `AInv` are indexed over `(i,i)`.

6.29.1 Usage

Command line:

```
gamstool [linalg.]Invert i A AInv gdxIn=fileIn.gdx gdxOut=fileOut.gdx
```

Compile time:

```
$callTool [linalg.]Invert i A AInv [gdxIn=fileIn.gdx] [gdxOut=fileOut.gdx]
```

Execution time:

```
executeTool '[linalg.]Invert i A AInv [gdxIn=fileIn.gdx] [gdxOut=fileOut.gdx]';
```

Where:

Argument	Description
`i`	Name of set used in matrix `i(*)`.
`A`	Name of two-dimensional matrix parameter `A(i,i)`.
`AInv`	Name of two-dimensional parameter containing the inverse `AInv(i,i)`.

The following parameters are available:

Parameter	Description
`gdxIn=fileIn.gdx`	Name of GDX file that contains symbols `i` and `A`. Mandatory if called from the command line, otherwise optional.
`gdxOut=fileOut.gdx`	Name of GDX file that contains symbol `AInv` after execution. Mandatory if called from the command line, otherwise optional.

6.29.2 Example

```
Set i /i1*i3 /;

Alias (i,j);

Table a(i,j) 'original matrix'
        i1   i2   i3
   i1    1    2    3
   i2    1    3    4
   i3    1    4    3;

Parameter inva(i,j) 'inverse of a';

execute_unload 'a.gdx', i, a;
executeTool.checkErrorLevel 'linalg.invert i a inva -gdxIn=a.gdx -gdxOut=b.gdx';
execute_load 'b.gdx', inva;

display a, inva;
```

6.30 MDB2GMS

Author

Erwin Kalvelagen

Attention

`MDB2GMS` is deprecated (see GAMS 40 MDB2GMS release notes). Please use Connect agent SQLReader instead.

6.30.1 Overview

`MDB2GMS` is a tool to convert data from an Microsoft Access database into GAMS readable format. The source is an MS Access database file (.mdb or .accdb) and the target is a GAMS Include File (.inc) or a GAMS GDX File (.gdx).

When running the executable mdb2gms.exe without command-line arguments, the tool will run interactively with a built-in GUI interface. Alternatively `MDB2GMS` can be run in batch mode, which is useful when running it directly from a GAMS model without user intervention using the `$call` command at compile time or the `execute` command at execution time.

Database tables can be considered as a generalization of a GAMS parameter. GAMS parameters have multiple index columns but just one value column. If the table is organized as multi-valued table, a UNION operation in the SQL statement can be used to generate the correct GAMS file.

There are no special requirements on the data types used in the database. The data is converted to strings, which is almost always possible. Data types like LONG BINARY may not be convertible to a string, in which case an exception will be raised. In general NULL's should not be allowed to get into a GAMS data structure. The handling of NULL's can be specified in an option.

Besides parameters it is also possible to generate set data.

6.30.2 Requirements

MDB2GMS runs only on Windows PC's and with MS Access installed. MS Access comes with certain versions of MS Office, but some MS Office versions will not include Access. The actual retrieval of the database records is performed by [DAO] (https://en.wikipedia.org/wiki/Data_access_object) or Data Access Objects, an object layer for accessing the database. The actual database is the Jet engine, which performs the queries and retrieves the data. Also consider to use SQL2GMS instead of MDB2GMS, if MS Access is not installed on your system.

To use this tool effectively you will need to have a working knowledge of [SQL] (https://en.wikipedia.org/wiki/SQL) in order to formulate proper database queries.

6.30.3 Batch Usage

MDB2GMS can be run in batch mode without user intervention from within the GAMS model by using the $call resp. execute statements or directly from command prompt while specifying all arguments in the command-line. A MDB2GMS batch call is of the following form:

```
mdb2gms inputFile outputFile queryString
```

A proper batch call will at least contain the following three command-line arguments:

1. The name of the MS Access database inputFile must be specified (.mdb or .accbd format). Use the I argument to enter the file name, i.e. I=inputFile.
2. The name of outputFile, either an include file (.inc) or GDX file (.gdx), must be specified. Using an include file to store the results of the query is indicated by the option O, i.e. O=outputFile.inc, while the use of a GDX file is indicated by the option X, i.e. X=outputFile.gdx.
3. The SQL queryString, containing the SQL statement to be executed on the database, must be specified within the option Q, i.e. Q=queryString.

See also Command-line Arguments below for a complete list of all possible command-line arguments. Consider that the $call or execute usage is rather error prone and you will need to spend a little bit of time to get the call correct and reliable. Alternatively, use the interactive built-in GUI interface or enter the command-line arguments in an external text file in order to write a more structured and readable command. The use of an external parameter file is indicated by preceding the file name with a @ (At sign).

Also consider to take a look at the section Strategies, mentioning some of the drawbacks of the batch usage and how to overcome them.

If you only specify I=inputFile then the interactive user interface is started with an initial setting of the input file name edit box equal to the name given in the command-line argument. Only if an input file, an output file and a query string is provided, the call will be considered as batch invocation.

6.30.3.1 Command-line Arguments

The following table summarizes the command-line arguments that can be specified when using MDB2GMS directly from the GAMS model or command prompt.

Argument	Interpretation	Default	Description
I	inputFile	none	Specify the name of the input file (required).
O	outputIncludeFile	none	Specify the name of the output file (.inc). Either `O=` or `X=` must be specified (or both).
On	n-th outputIncludeFile	none	Match the nth query with the nth output file (.inc format) if multiple queries are used.
X	outputGDXFile	none	Specify the name of the output file (.gdx). Either `O=` or `X=` must be specified (or both).
Q	Query	none	This option can be used to specify a SQL query (required).
Qn	n-th Query	none	Match the nth query with the nth output file (.inc) format or with the nth set- or parameter name when writing to GDX if multiple queries are used.
S	setName	none	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file.
Sn	n-th setName	none	Match the nth query with the nth set in the GDX file if multiple queries are used.
Y	setName (with expl. text)	none	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. Use this argument to store a set with explanatory text.
Yn	n-th setName (with expl. text)	none	Match the nth query with the nth set (with explanatory text) in the GDX file if multiple queries are used.
P	parameterName	none	If we write to a GDX file, use this option to specify the name of a parameter to be used inside the GDX file.
Pn	n-th parameterName	none	Match the nth query with the nth parameter in the GDX file if multiple queries are used.
D	Debug	disabled	Generate debug information.
B	Quote Blanks	disabled	Quote strings if they contain blanks or embedded quotes.
M	Mute	disabled	Controls if additional information is written to the log and include file.
L	Listing	disabled	Controls if the data is embedded in the listing file.
@fileName	ext. options file	none	Causes the program to read options from an external text file.
N	iniFileName	mdb2gms.ini	Indicates the usage of a different INI file.
F	formatString	none	Specify a format string.

Argument	Interpretation	Default	Description
W	Wiring	none	Maps database columns to GAMS index positions.
R	rowBatchSize	100	Row batch size; the default is 100 records.

Some more detailed remarks on the command-line arguments:

I = *string* (inputFile, default = none)

This option is required and specifies the name of the .mdb or .accbd file containing the Access database. If the file contains blanks the name should be surrounded by double quotes. It is advised to use absolute paths, so Access has no confusion what file to open. On a network UNC Names can be used, and files from another computer can be accessed, e.g.

```
"\\hostname\c\my documents\a.mdb."
```

This option is required for batch processing. To specify a path equal to the location where the .gms file is located, you can use:

```
I=system.fpmydb.mdb
```

This option is demonstrated in most examples, see Example 1 - Reading a single valued Table for instance.

O = *string* (outputIncludeFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter or set statement. Make sure the directory is writable. UNC Names can be used. An output file must be specified for batch operation: i.e. either `O=` or `X=` needs to be specified (or both). The include file will be an ASCII file that can be read by GAMS using the $include command within the data definition of a set, parameter or scalar. If the include file already exists, it will be overwritten. This option is demonstrated in most examples, see Example 1 - Reading a single valued Table for instance.

On = *string* (outputIncludeFile, default = none)

When using multiple queries in a single `MDB2GMS` call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition:

```
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

X = *string* (outputGDXFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS GDX file. Make sure the directory is writable. UNC names can be used. If the GDX file already exists it will be overwritten - it is not possible to append to a GDX file. An output file must be specified for batch operation: i.e. either `O=` or `X=` needs to be specified (or both). This option is demonstrated in Example 5 - Reading Set with Explanatory Text or Example 7 - Multi-Query Batch Example for instance.

Q = *string* (Query, default = none)

This option can be used to specify an SQL query. Queries can contain spaces and thus have to be surrounded by double quotes. For the exact syntax of the queries that is accepted by Access we refer to the documentation that comes with MS Access. The query is passed on directly to the Jet database engine, so the complete power and expressiveness of Access SQL is available. For an exact description of allowed expressions consult a text on MS Access.

One notable syntax feature is that when field names or table names contain blanks, they can be specified in square brackets. Examples:

```
Q="SELECT * FROM mytable"
Q="SELECT year, production FROM [production table]"
Q="SELECT [GAMS City], value FROM [example table], CityMapper
WHERE [Access City]=city"
```

This option is demonstrated in Example 1 - Reading a single valued Table for instance

Qn = *string* (Query, default = none)

When using multiple queries in a single `MDB2GMS` call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition. In addition, you can match the results of a query with a specific set- or parameter name when writing to GDX.

```
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

or (GDX output file format - where several sets and parameters can be stored in a single file):

```
Q1="SELECT a, b FROM table"
P1=abParameter
Q2="SELECT c FROM table"
S2=cSet
```

Note the usage of the arguments Pn resp. Sn in order to store the results as parameter resp. set and to specify the name of the symbols. See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

S = *string* (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be stored in the GDX file (containing the results of the query). This option is demonstrated in Example 4 - Reading a multi dimensional Set.

Sn = *string* (setName, default = none)

If multiple queries are used in a single MDB2GMS call while writing to a GDX file, use this option to specify the name of the nth set to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i FROM table"
S1=iSet
Q2="SELECT j FROM table"
S2=jSet
```

See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

Y = *string* (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. The last column specified within the select clause in the SQL statement will be used as explanatory text. This option is demonstrated in Example 5 - Reading Sets with Explanatory Text for instance.

Yn = *string* (setName, default = none)

If multiple queries are used in a single MDB2GMS call while writing to a GDX file, use this option to specify the name of the nth set (with explanatory text) to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i, explTextForSeti FROM table"
Y1=iSet
Q2="SELECT j, explTextForSetj FROM table"
Y2=jSet
```

The last column specified within the select clause in the SQL statements will be used as explanatory text.

P = *string* (parameterName, default = none)

If we write to a GDX file, use this option to specify the name of a parameter to be stored the GDX file (containing the results of the query).

Pn = *string* (parameterName, default = none)

If multiple queries are used in a single MDB2GMS call while writing to a GDX file, use this option to specify the name of the nth parameter to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i, j, value FROM table"
A1=ijValue
Q2="SELECT n, m, value FROM table"
A2=nmValue
```

See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

D (Debug, default = disabled)

This option can be used for debugging purposes. If specified the import filter will not run minimized but "restored", i.e. as a normal window. In addition the program will not terminate until the user clicks the Close button. This allows you to monitor possible errors during execution of MDB2GMS.

B (Quote Blanks, default = disabled)

If this parameter is specified, strings that have blanks in them will be quoted. If the string is already quoted this step is not performed. If the name contains an embedded single quote, the surrounding quotes will be double quotes. If the name already contains a double quote, the surrounding quotes will be single quotes. If both single and double quotes are present in the string, then all double quotes are replaced by single quotes and the surrounding quotes will be double quotes. By default this option is turned off. For more information see subsection Quotes. This option only applies to an output include file.

M (Mute, default = disabled)

Run in modest or mute mode: no additional information, such as version number, number of rows in the data, elapsed time, used query etc. is written to the log and include file.

L (Listing, default = disabled)

Embed the data between the `$offListing` and `$onListing` dollar control options, so the data will not be listed in the listing file. This is a quick way to reduce the size of the listing file when including very large data files into the model. Otherwise the listing file would become too large to be handled comfortably.

@fileName = *string* (fileName, default = none)

Causes the program to read options from an external text file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if they were specified directly on the command-line. See also Command Files for some further details.

N = *string* (fileName, default = mdb2gms.ini)

Use a different INI file than the standard mdb2gms.ini located in the same directory as the executable mdb2gms.exe.

F = *string* (formatString, default = none)

In special cases we can apply a format string on the include file output (not for GDX output). Each column in the result set is a string and can be represented by a s in the format string.

W = *string* (wiring, default = none)

By using the `W` option, one can map database columns to GAMS index positions. See model **[Wiring]** for reference.

R = *integer* (rowBatchSize, default = 100)

Row batch size; the default is 100 records. This option must be specified in an INI file when using the interactive mode of `MDB2GMS`.

6.30.3.2 Example 1 - Reading a single valued Table

Suppose we want to read the distances parameter of the [**trnsport**] model from the GAMS Model Library. The data is stored in the Microsoft Access Database format (file Sample.mdb).

Distances

City1	City2	Distance
SEATTLE	CHICAGO	1,7
SAN-DIEGO	CHICAGO	1,8
SEATTLE	NEW-YORK	2,5
SAN-DIEGO	NEW-YORK	2,5
SEATTLE	TOPEKA	1,8
SAN-DIEGO	TOPEKA	1,4

Figure 6.55 Table: distances

The data can be queried with a simple SQL statement:

```
SELECT city1, city2, distance
FROM   distances
```

By running the following `MDB2GMS` statement, the connection to the database Sample.mdb is established. In addition, the data will be queried and the results are written to a GAMS include file afterwards (.inc).

```
mdb2gms I=Sample.mdb Q="SELECT city1, city2, distance FROM distances" O=distances.inc
```

The MS Access database file name is specified using the argument I. Note that the string is enclosed by quotes, as the string contains blanks. The arguments Q and O are used to specify the query and the output file name (and format).

The generated include file distances.inc looks like:

```
* -----------------------------------------------------
* MDB2GMS          24.8.5 r61358 Released May 10, 2017 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* DAO version: 14.0
* Jet version: 4.0
* Database:    F:\datalib\Sample.mdb
* Query:       SELECT city1, city2, distance FROM distances
* -----------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----------------------------------------------------
```

The commented header section summarizes some information about the `MDB2GMS` resp. GAMS version and about the executed database query. The standard export format is to consider the last column as the value column (containing the distances) and the previous columns as the indices (containing the city names). The indices are separated by a dot, allowing the generated include file to be used as part of a parameter declaration statement in your GAMS model.

Retrieving the data using `MDB2GMS` from the database and including the queried data in your GAMS model within the parameter declaration statement (at compile time) can be combined in the following way:

```
Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

$call mdb2gms I=Sample.mdb Q="SELECT city1, city2, distance FROM distances" O=distances.inc
Parameter d(i,j) 'distance in thousands of miles' /
$include distances.inc
/;

display d;
```

Finally, the values of the parameter `d` are displayed:

```
             new-york      chicago       topeka

seattle         2.500        1.700        1.800
san-diego       2.500        1.800        1.400
```

This example is also part of the GAMS Data Utilities Library, see model [**Distances1**] for reference. Note that the query results are written to a GDX file in addition.

6.30.3.3 Example 2 - Reading a multi valued Table

In this scenario, we want two read the three index columns `year`, `loc`, `prod` and the value columns `sales` and `profit` from the database file Sample.mdb. Therefore, we use two separate parameters and queries or alternatively a parameter with an extra index position (for `sales` resp. `profit`) and a UNION select.

Consider the table with two value columns `sales` and `profit`:

data

year	loc	prod	sales	profit
1997	la	hardware	80	8
1997	la	software	60	16
1997	nyc	hardware	110	5
1997	nyc	software	100	10
1997	sfo	hardware	80	9
1997	sfo	software	50	10
1997	was	hardware	120	7
1997	was	software	70	20
1998	la	hardware	70	6
1998	la	software	70	10

Figure 6.56 Table: data (shortened for presentation)

Two separate Parameters

A simple way to import this into GAMS is to use two parameters and two SQL queries. The SQL queries can look like:

```
SELECT year, loc, prod, sales
FROM   data
SELECT year, loc, prod, profit
FROM   data
```

We can generate a include file sales.inc by running the following command:

```
mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, sales FROM data" O=sales.inc
```

Note that we specify the first query in order to select the sales and the relevant index columns within the Q argument. The query results are written to sales.inc using the O argument. Analogously we generate a include file profit.inc by running the following command while specifying the second query in order to obtain the profits and the relevant index columns:

```
mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, profit FROM data" O=profit.inc
```

Retrieving the data using MDB2GMS from the database Sample.mdb and including the queried data in your GAMS model within the parameter declaration statements of sales and profit (at compile time) can be combined in the following way:

```
Set
   year 'years'     / 1997*1998 /
   loc  'locations' / nyc, was, la, sfo  /
   prd  'products'  / hardware, software /;

$call mdb2gms I=Sample.mdb Q="SELECT year, loc, prod, sales FROM data" O=sales.inc
Parameter sales(year,loc,prd) /
$include sales.inc
/;

$call mdb2gms I=Sample.mdb" Q="SELECT year, loc, prod, profit FROM data" O=profit.inc
Parameter profit(year,loc,prd) /
$include profit.inc
/;
```

This example is also part of the GAMS Data Utilities Library, see model [**SalesProfitDB1**] for reference.

Single Parameter with extra Index Position

The operation can also be performed in one big swoop by using a different GAMS datastructure, i.e. a single parameter is defined with an extra index type to indicate the data type (sales or profit). The index and value columns will be selected by the following SQL statement. Note the UNION operation in order to combine the results and the strings 'sales' resp. 'profit' to identify the data type later on.

```
SELECT year, loc, prod, 'sales', sales
FROM   data
UNION
SELECT year, loc, prod, 'profit', profit
FROM   data
```

The data is accessed, queried and written to data.inc by running the following command:

```
mdb2gms @howToRead.txt
```

Note that usage of the external parameter file howToRead.txt shown below in order to increase the readability of the command (one argument per line, quotes can be omitted):

```
I=Sample.mdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FR
O=data.inc
```

The generated include file data.inc looks like (shortened for presentation):

```
* ------------------------------------------------------
* MDB2GMS          24.8.5 r61358 Released May 10, 2017 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* ------------------------------------------------------
* DAO version: 14.0
* Jet version: 4.0
* Database:    F:\datalib\Sample.mdb
* Query:       SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit
* ------------------------------------------------------
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
* ------------------------------------------------------
```

Retrieving the data using `MDB2GMS` from the database and including the queried data in your GAMS model within the parameter declaration (at compile time) can be combined in the following way (note that the parameter has a fourth index `type` in order to access the data type `sales` resp. `profit`):

```
$onEcho > howToRead.txt
I=Sample.mdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FR
O=data.inc
$offEcho

Set
   year 'years'     / 1997*1998 /
   loc  'locations' / nyc, was, la, sfo  /
   prd  'products'  / hardware, software /
   type 'data type' / sales, profit      /;

$call mdb2gms @howToRead.txt
Parameter data(year,loc,prd,type) /
$include data.inc
/;
```

This example is also part of the GAMS Data Utilities Library, see model [**SalesProfitDB2c**] for reference.

6.30.3.4 Example 3 - Reading a one dimensional Set

This example demonstrates how to read set elements of a one dimensional set from a single column of a MS Access database file. Suppose we want to read the column City1 (see table distances) in order to define the set `i` in the GAMS model. Make sure elements are unique by using the distinct operation within the SQL statement (otherwise there will be an error when including the file within the set definition in the GAMS model, as some set elements will be redefined):

```
SELECT distinct(City1)
FROM   distances
```

The include file city1.inc looks like (header informations are removed in order to shorten the representation):

```
* ----------------------------------------------------
 SAN-DIEGO
 SEATTLE
* ----------------------------------------------------
```

All steps (data access via `MDB2GMS`, set definition) can be combined:

```
$call mdb2gms I=Sample.mdb Q="SELECT distinct(city1) FROM distances" O=city_i.inc
Set i 'canning plants' /
$include city_i.inc
/;

display i;
```

The display statement generates the following output in the listing file:

```
----     56 SET i

seattle  ,    san-diego
```

6.30.3.5 Example 4 - Reading a multi dimensional Set

When reading a multi dimensional set from database and writing the results to an include file by using the O argument, one has to observe that the elements in the include file must have the correct format in order to be interpreted as element of a multi dimensional set. For instance, a line containing `a b c` is not recognized as a proper set element of a three dimensional set. In particular, one has to add periods between the single elements, i.e. `a.b.c` will be interpreted correctly.

There are different ways to add these periods explicitly within the SQL statement. E.g. add a dummy value field by adding a quoted blank to the select clause (index1, index2, index3 and dataTable are some placeholders):

```
SELECT index1, index2, index3, " " FROM dataTable
```

or by adding the periods explicitly within the select clause (|| or & depending on DBMS):

```
SELECT index1&'.'&index2&'.'&index3 FROM dataTable
SELECT index1||'.'||index2||'.'||index3 FROM dataTable
```

For instance, suppose we want to define a two dimensional set

```
Set ij(i,j) 'canning plants - markets';
```

based on the data of the table distances stored in Sample.mdb. The following MDB2GMS statement connects you to the database, queries the columns with the city names and adds an empty value field in order to create periods between the set elements:

```
mdb2gms I=Sample.mdb Q="SELECT city1, city2, ' ' FROM distances" O=city_ij.inc
```

The include file city_ij.inc looks like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
SEATTLE.NEW-YORK ' '
SAN-DIEGO.NEW-YORK ' '
SEATTLE.CHICAGO ' '
SAN-DIEGO.CHICAGO ' '
SEATTLE.TOPEKA ' '
SAN-DIEGO.TOPEKA ' '
* -----------------------------------------------------
```

Without adding the empty value field, the resulting include file would look like (shortened):

```
* -----------------------------------------------------
SEATTLE NEW-YORK
SAN-DIEGO NEW-YORK
* -----------------------------------------------------
```

Since the periods are missing, the lines are not recognized as valid elements of a two dimensional set. All steps can be combined in the following way:

```
Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

$call mdb2gms I=Sample.mdb" Q="SELECT city1, city2, ' ' FROM distances" O=city_ij.inc

Set ij(i,j) 'two dimensional set' /
$include city_ij.inc
/;

display ij;
```

The display statement generates the following output in the listing file:

```
----     75 SET ij  two dimensional set

             new-york     chicago      topeka

SAN-DIEGO         YES         YES         YES
SEATTLE           YES         YES         YES
```

Note that there is no need to add periods explicitly when reading multi dimensional sets, if the results are written only to a GDX file by using the X and S resp. Y arguments, i.e. there is no need to modify the query:

```
SELECT index1, index2, index3 FROM datatable
```

when using MDB2GMS in the following way:

```
mdb2gms I=Sample.mdb Q="SELECT index1, index2, index3 FROM datatable" X=setData.gdx S=setName
```

which will generate the file setData.gdx with a three dimensional set named setName containing the results of the query.

6.30.3.6 Example 5 - Reading Sets with Explanatory Text

In this example, we will demonstrate how to read set elements with explanatory text from a MS Access database file using MDB2GMS. In the first place, we are going to write the query results to an include file, afterwards we use the Y argument in order to store the query results as a set with explanatory text in a GDX file.

setData

setElement	explText
firstSetElement	Explanatory text for the first 'set element'
secondSetElement	Explanatory text for the second "set element"
thirdSetElement	Explanatory text for the third 'set element"
fourthSetElement	Explanatory text for the fourth set element

Figure 6.57 Table: setData

Note the blanks and the mixed quotes in the column containing the explanatory text. The data can be accessed by the following query:

```
SELECT setElement, explText
FROM   setData
```

Writing the Query Results in an include File

The last column in the select clause will be used as explanatory text. Take in mind to add the argument B in order to handle text strings with embedded blanks or quotes. The following GAMS code accesses the data and writes the results to an include file setData.inc:

```
$call mdb2gms I=Sample.mdb B Q="SELECT setElement, explText FROM setData" O=setData.inc

Set a /
$include setData.inc
/;
```

The resulting include file will look like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
firstSetElement "Explanatory text for the first 'set element'"
secondSetElement 'Explanatory text for the second "set element"'
thirdSetElement "Explanatory text for the third 'set element'"
fourthSetElement 'Explanatory text for the fourth set element'
* -----------------------------------------------------
```

Note the handling of the quotes according to the description in B.

Writing the Query Results in a GDX File
When storing the results of the query as a set with explanatory text in a GDX file, there is no need to observe embedded blanks or quotes manually, instead one can use the Y argument. The last column specified in the select clause of the SQL statement will be interpreted as explanatory text. The following GAMS code accesses the data and writes the results to a GDX file setData.inc:

```
$call mdb2gms I=Sample.mdb Q="SELECT setElement, explText FROM setData" X=setData.gdx Y=set_b

Set b;
$gdxIn setData.gdx
$load b = set_b
$gdxIn
```

Note that the name of the set in the GDX file is `set_b` (specified within the `Y` argument), while the name of the GDX file was specified within the X argument.

6.30.3.7 Example 6 - Index Mapping

In some cases the index elements used in the database are not the same as in the GAMS model. E.g. consider the case where the GAMS model has defined a set as:

```
Set i / NY, DC, LA, SF /;
```

Now assume a data table looks like:

example table

city	value
new york	100
los angeles	120
san francisco	105
washington dc	102

Figure 6.58 Table: example table

This means we have to map 'new york' to 'NY' etc. This mapping can be done in two places: either in GAMS or in the database.

Index Mapping done in GAMS

When we export the table directly, we get the following include file (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
'new york' 100
'los angeles' 120
'san francisco' 105
'washington dc' 102
* -----------------------------------------------------
```

Note that the single quotes are added by activating the option B (quote blanks), as the index elements contain blanks. Accessing the data, importing the resulting include file and converting it to a different index space can be done by the following GAMS code:

```
Set i / NY, DC, LA, SF /;

Set idb 'from database' / 'new york', 'washington dc', 'los angeles', 'san francisco' /;

$call mdb2gms I=Sample.mdb B O="city1.inc" Q="SELECT city, value FROM [example table]"
Parameter dbdata(idb) /
$include city1.inc
/;

Set mapindx(i,idb) / NY.'new york', DC.'washington dc', LA.'los angeles', SF.'san francisco' /;

Parameter data(i);
data(i) = sum(mapindx(i,idb), dbdata(idb));
display data;
```

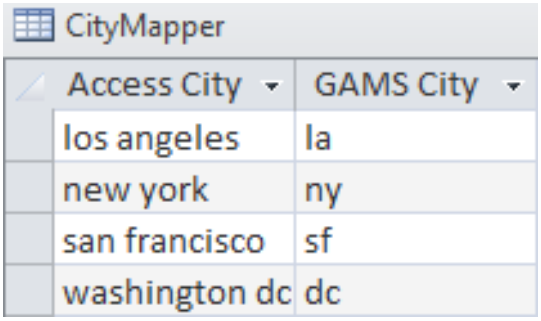
The display statement generates the following output in the listing file:

```
----     47 PARAMETER data

NY 100.000,    DC 102.000,    LA 120.000,    SF 105.000
```

This example is also part of the GAMS Data Utilities Library, see model [**IndexMapping1**] for reference.

Index mapping done in Database

The second approach is to handle the mapping inside the database. We can introduce a mapping table that looks like:



CityMapper

Access City	GAMS City
los angeles	la
new york	ny
san francisco	sf
washington dc	dc

Figure 6.59 Table: CityMapper

This table can be used in a join to export the data in a format we can use by executing the query:

```
SELECT [GAMS City], value
FROM    example_table, CityMapper
WHERE   CityMapper.[Access City]=example_table.city
```

The resulting include file looks like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
la 120
ny 100
sf 105
dc 102
* -----------------------------------------------------
```

All steps can be combined in the GAMS model:

```
Set i / NY, DC, LA, SF /;

$onEcho > howToRead.txt
I=Sample.mdb
Q=SELECT [GAMS City], [value] FROM example_table, CityMapper WHERE CityMapper.[Access City]=example_
O=city2.inc
$offEcho

$call mdb2gms @howToRead.txt
Parameter data(i) /
$include city2.inc
/;
display data;
```

The display statement generates the following output in the listing file:

```
----     38 PARAMETER data

NY 100.000,    DC 102.000,    LA 120.000,    SF 105.000
```

Note: MS Access allows table names with embedded blanks. In that case the table name can be surrounded by square brackets. Other databases may not allow this.

This example is also part of the GAMS Data Utilities Library, see model [**IndexMapping2**] for reference.

6.30.4 Multi-Query Batch Usage

In some cases a number of small queries need to be performed on the same database. However, several individual MDB2GMS execution can become expensive, since there is significant overhead in starting Access and opening the database. For these cases, we have added the option to do multiple queries in one call. To execute several queries in a single MDB2GMS call and write several GAMS include files containing the results of the queries, we can use the command-line arguments Qn and On. The structure of a multi-query call looks like:

```
I=sample.mdb

Q1=firstQuery
O1=outputFileName.inc

Q2=secondQuery
O2=outputFileName.inc

Q3=thirdQuery
O3=outputFileName.inc
```

The terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. We see that the argument `Qn` is matched by an argument `On`. That means that the results of the n-th query are written to the n-th output file.

In case we want to store the results of a multi-query call to a single GDX file, we can use the command-line arguments Qn, Sn, Pn and Yn. The structure of a multi-query call when writing to a GDX file looks like:

```
I=sample.mdb
X=sample.gdx

Q1=firstQuery
S1=setName

Q2=secondQuery
S2=setName

Q3=thirdQuery
A3=parameterName

Q4=fourthQuery
A4=setName
```

Again, the terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. Here we see that a query `Qn` is matched by either a set name `Sn` or a parameter name `Pn`, i.e. the results of the first query will be stored as a set whose name is specified within the `S1` argument, the results of the third query will be stored as a parameter whose name is specified within the `P3` argument etc. The `X` argument is used to specify the name of the GDX file.

For a complete example see section Example 7 - Multi-Query Batch Example.

6.30.4.1 Example 7 - Multi-Query Batch Example

As an example database we use the following Access table (file Sample.mdb):

data

year	loc	prod	sales	profit
1997	la	hardware	80	8
1997	la	software	60	16
1997	nyc	hardware	110	5
1997	nyc	software	100	10
1997	sfo	hardware	80	9
1997	sfo	software	50	10
1997	was	hardware	120	7
1997	was	software	70	20
1998	la	hardware	70	6
1998	la	software	70	10

Figure 6.60 Table: data (shortened for presentation)

We want to extract the following information:

- The set **year**
- The set **loc**
- The set **prd**
- The parameter **sales**
- The parameter **profit**

Output: Several include Files

This can be accomplished using the following GAMS code with multiple queries in a single `MDB2GMS` call (note the usage of the distinct operator in the select clauses of the queries whose results will be used as sets in order to keep the set elements unique):

```
$onEcho > howToRead.txt
I=Sample.mdb

Q1=SELECT distinct(year) FROM data
O1=year.inc

Q2=SELECT distinct(loc) FROM data
O2=loc.inc

Q3=SELECT distinct(prod) FROM data
O3=prod.inc
```

```
Q4=SELECT prod, loc, year, sales FROM data
O4=sales.inc

Q5=SELECT prod, loc, year, profit FROM data
O5=profit.inc
$offEcho

$call =mdb2gms @howToRead.txt

Set y 'years' /
$include year.inc
/;
Set loc 'locations' /
$include loc.inc
/;
Set prd 'products' /
$include prod.inc
/;

Parameter sales(prd,loc,y) /
$include sales.inc
/;
display sales;

Parameter profit(prd,loc,y) /
$include profit.inc
/;
display profit;
```

This example is also part of the GAMS Data Utilities Library, see model [**SalesProfitDB3**] for reference.

Output: A single GDX File

The same example imported through a GDX file can look like:

```
$onEcho > howToRead.txt
I=Sample.mdb
X=Sample.gdx

Q1=SELECT distinct(year) FROM data
S1=year

Q2=SELECT distinct(loc) FROM data
S2=loc

Q3=SELECT distinct(prod) FROM data
S3=prd

Q4=SELECT prod, loc, year, sales FROM data
P4=sales

Q5=SELECT prod, loc, year, profit FROM data
P5=profit
$offEcho
```

```
$call =mdb2gms @howToRead.txt

Set
   y   'years'
   loc 'locations'
   prd 'products';

Parameter
   sales(prd,loc,y)
   profit(prd,loc,y);

$gdxIn Sample.gdx
$load y=year prd loc sales profit
$gdxIn

display sales, profit;
```

The call of the GDXViewer will display the GDX file in the stand-alone GDX viewer. This example is also part of the GAMS Data Utilities Library, see model **[SalesProfitDB4]** for reference.

6.30.5 Interactive Usage

When the tool is called without command-line parameters, it will startup interactively. Using it this way, one can specify the database file (.mdb or .accbd file), the query and the final destination file (a GAMS include file or a GDX file) using the built-in interactive environment. The main screen (see figure below) contains a number of buttons and edit boxes, which are explained below.

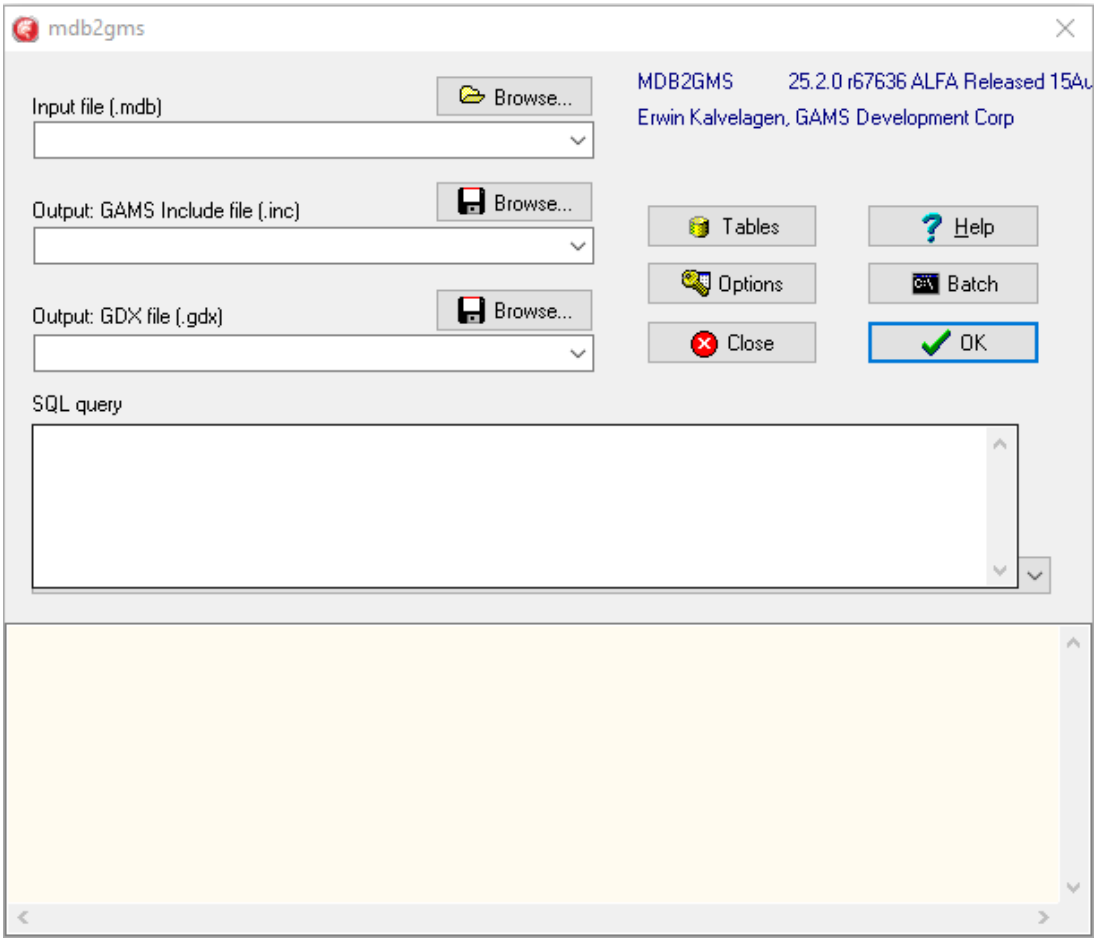
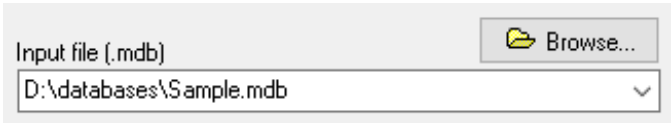


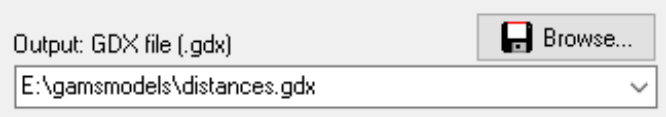
Figure 6.61 MDB2GMS - Graphical User Interface

- **Input file (.mdb or .accbd)**. This is the combo box to specify the input file. See also inputFile for some more detailed remarks. The browse button can be used to launch a file open dialog which makes it easier to specify a file. The file may be located on a remote machine using the notation `\\machine\directory\file.mdb`.

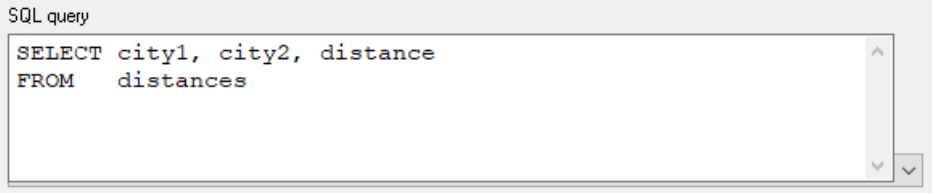


- **Output GAMS Include file (∗.inc)**. If you want to create a GAMS include file, then specify here the destination file. See also outputIncludeFile for some more detailed remarks.

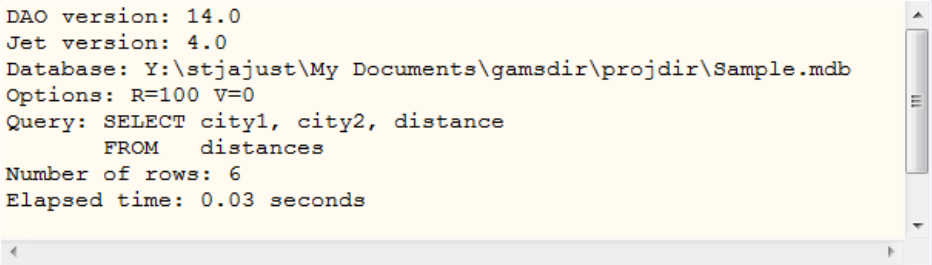
- **Output GDX file (∗.gdx)**. As an alternative to a GAMS include file, the tool can also generate a GDX file. One or both of the output files need to be specified. See also outputGDXFile for some more detailed notes.



- **SQL Query**. The SQL Query box is the place to provide the query. Note that the actual area for text can be larger than is displayed: use the cursor-keys to scroll. See also Q for some more detailed remarks. For an exact description of allowed expressions consult a text on MS Access.



- **Progress Memo**. This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.



- The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).

- The **Tables button** will pop up a new window with the contents of the database file selected in the input file edit line. This allows you to see all table names and field names needed to specify a correct SQL query. An exception will be generated if no database file name is specified in the input edit line.

- The **Options button** will pop up a window where you can specify a number of options.

- Pressing the **Help button** will show this documentation.

- Pressing the **OK button** will execute the query and an include file or GDX file will be generated.

- Pressing the **Batch button** will give information on how the current query can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied to the clipboard. In the IDE press `Ctrl-C` or choose `Edit|Paste` to copy the contents of the clipboard to a GAMS text file.

- Pressing **Close button** will exit the application. The current settings will be saved in an INI file so when you run `MDB2GMS` again all current settings will be restored.

6.30.5.1 Options

The **Options** window can be created by pressing the options button:

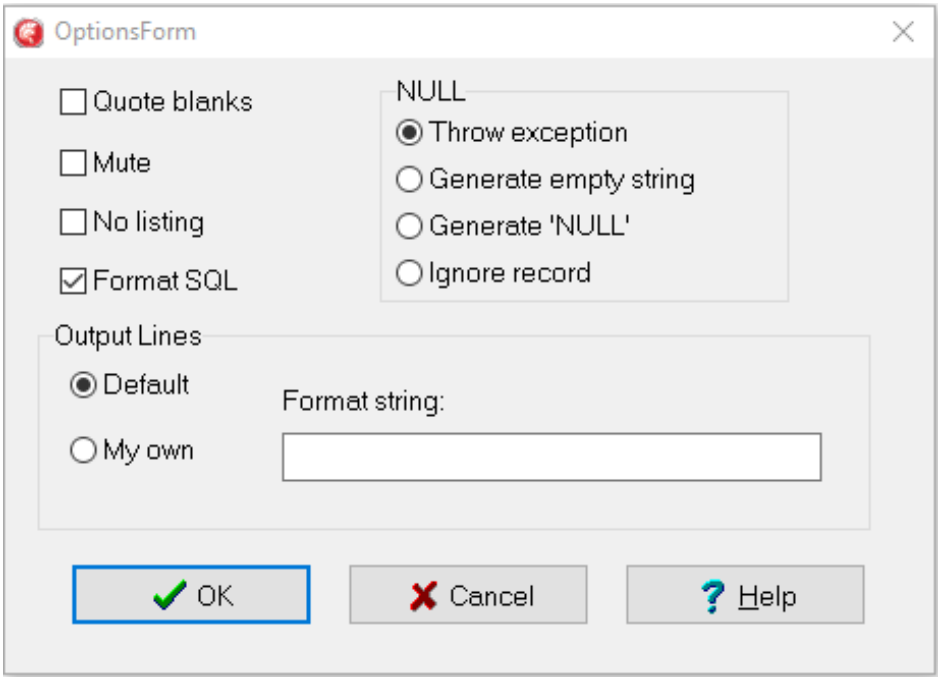
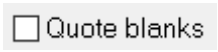


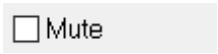
Figure 6.62 Options Menu of the Graphical User Interface

The following options are available in the options window:

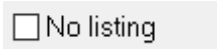
- **Quote blanks:** Quote strings if they contain blanks or embedded quotes. See also B for some more detailed notes.



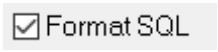
- **Mute:** Don't include the extra informational text (such as used query etc.) in the include file.



- **No listing:** Surround the include file by `$offListing` and `$onListing` so that the data will not be echoed to the listing file. The equivalent command-line argument is L.

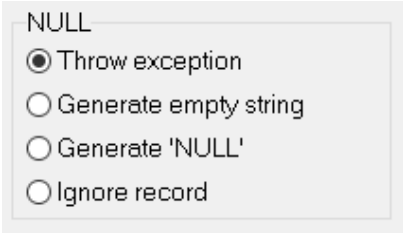


- **Format SQL:** If an SQL text is reloaded in the SQL Edit Box, it will be formatted: keywords will be printed in CAPS and the FROM and WHERE clause will be printed on their own line. If this check box is unchecked this formatting will not take place and SQL queries will be shown as is.

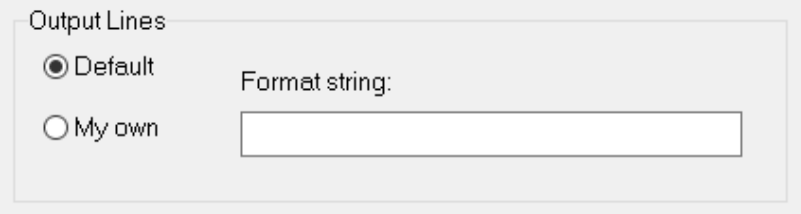


The following options are only needed in special cases:

- **NULL:** This radio box determines how NULL's are handled. A NULL in an index position or a value column will usually make the result of the query useless: the GAMS record will be invalid. To alert you on NULL's the default to throw an exception is a safe choice. In special cases you may want to map NULL's to an empty string or a 'NULL' string.



- **Output Lines:** By default output lines are created as follows: all first n-1 fields are considered indices and the last n-th column is the value. The format corresponding to this situation is '%s.%s.%s %s'` (for a three dimensional parameter). In special cases you may want to tinker with the format string being used. The fields are all considered strings, so only use s as format placeholder. Make sure you specify exactly the same number of s's as there are columns in the result set.



The buttons have an obvious functionality:

- **OK button** will accept the changes made.

- **Cancel button** wil ignore the changes made, and all option settings will revert to their previous values.

- **Help button** will show this help text.

6.30.6 Strategies

Including SQL statements to extract data from a database inside your model can lead to a number of difficulties:

- The database can change between runs, leading to results that are not reproducible. A possible scenario is a user calling you with a complaint: "the model is giving strange results". You run the model to verify and now the results are ok. The reason may be because the data in the database has changed.
- There is significant overhead in extracting data from a database. If there is no need to get new data from the database it is better to use a snapshot stored locally in a format directly accessible by GAMS.
- It is often beneficial to look at the extracted data. A first reason, is just to make sure the data arrived correctly. Another argument is that viewing data in a different way may lead to a better understanding of the data. A complete "under-the-hood" approach may cause difficulties in understanding certain model behavior.

Often it is a good strategy to separate the data extraction step from the rest of the model logic.

If the sub-models form a chain or a tree, like in:

```
Data Extraction --> Data Manipulation --> Model Definition --> Model Solution --> Report Writing
```

we can conveniently use the save/restart facility. The individual submodel are coded as:

- **Step 0: sr0.gms**

```
$onText
step 0: data extraction from database
execute as: > gams sr0 save=s0
$offText

Set
   i 'suppliers'
   j 'demand centers';

Parameter
   demand(j)
   supply(i)
   dist(i,j) 'distances';
```

```
$onEcho > howtoRead.txt
I=transportation.mdb

Q1=select name from suppliers
O1=i.inc

Q2=select name from demandcenters
O2=j.inc

Q3=select name,demand from demandcenters
O3=demand.inc

Q4=select name,supply from suppliers
O4=supply.inc

Q5=select supplier,demandcenter,distance from distances
O5=dist.inc
$offEcho

$call =mdb2gms.exe @howtoRead.txt

Set i /
$include i.inc
/;

Set j /
$include j.inc
/;

Parameter demand /
$include demand.inc
/;

Parameter supply /
$include supply.inc
/;

Parameter dist /
$include dist.inc
/;

display i, j, demand, supply, dist;
```

- **Step 1: sr1.gms**

```
$onText
step 1: data manipulation step
execute as: > gams sr1 restart=s0 save=s1
$offText

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*dist(i,j)/1000;
```

- **Step 2: sr2.gms**

```
$onText
step 2: model definition
```

```
execute as: > gams sr2 restart=s1 save=s2
$offText

Variable
   x(i,j) 'shipment quantities in cases'
   z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
   ecost      'define objective function'
   esupply(i) 'observe supply limit at plant i'
   edemand(j) 'satisfy demand at market j';

ecost..      z =e= sum((i,j), c(i,j)*x(i,j));

esupply(i).. sum(j, x(i,j)) =l= supply(i);

edemand(j).. sum(i, x(i,j)) =g= demand(j);
```

- **Step 3: sr3.gms**

```
$onText
step 3: model solution
execute as: > gams sr3 restart=s2 save=s3
$offText

option lp = cplex;

Model transport / all /;
solve transport using lp minimizing z;
```

- **Step 4: sr4.gms**

```
$onText
step 4: report writing
execute as: > gams sr4 restart=s3
$offText

abort$(transport.modelStat <> 1) "model not solved to optimality";

display x.l, z.l;
```

A model that executes all steps can be written as:

```
execute '=gams.exe sr0 lo=3 save=s0';
abort$errorLevel "step 0 failed";

execute '=gams.exe sr1 lo=3 restart=s0 save=s1';
abort$errorLevel "step 1 failed";

execute '=gams.exe sr2 lo=3 restart=s1 save=s2';
abort$errorLevel "step 2 failed";

execute '=gams.exe sr3 lo=3 restart=s2 save=s3';
abort$errorLevel "step 3 failed";

execute '=gams.exe sr4 lo=3 restart=s3';
abort$errorLevel "step 4 failed";
```

If you only change the reporting step, i.e. generating some output using PUT statements, then you only need to change and re-execute step 4. If you change solver or solver options, then only steps 3 and 4 need to be redone. For a small model like this, this exercise may not be very useful, but when the model is large and every step is complex and expensive, this is a convenient way to achieve quicker turn-around times in many cases.

The model [**MDBSr5**] is also part of the GAMS Data Utilities Library.

In some cases the save/restart facility is not appropriate. A more general approach is to save the data from the database in a GDX file, which can then be used by other models. We can use the model from step 0 to store the data in a GDX file:

MDB2GDX1.gms

```
execute '=gams.exe sr0 lo=3 gdx=trnsport.gdx';
abort$errorLevel "step 0 failed";

execute '=gdxviewer.exe trnsport.gdx';
```

The model [**MDB2GDX1**] is also part of the GAMS Data Utilities Library.

We can also let MDB2GMS create the GDX file:

MDB2GDX2.gms

This model demonstrates how to store data from Access database (file Transportation.mdb) into a GDX file.

```
$onEcho > howToRead.txt
I=Transportation.mdb
X=Transportation.gdx

Q1=SELECT name FROM suppliers
S1=i

Q2=SELECT name FROM demandcenters
S2=j

Q3=SELECT name, demand FROM demandcenters
P3=demand

Q4=SELECT name, supply FROM suppliers
P4=supply

Q5=SELECT supplier, demandcenter, distance FROM distances
P5=dist
$offEcho

$call =mdb2gms.exe @howToRead.txt
```

The first approach has the advantage that a complete audit record is available from the data moved from the database to the GDX file in the sr0.lst listing file. If someone ever wonders what came out of the database and how this was stored in the GDX file, that file gives the answer.

The model [**MDB2GDX2**] is also part of the GAMS Data Utilities Library.

To load the GDX data the following fragment can be used:

GDXTRNSPORT.gms

This model demonstrates how to load the transportation data from GDX file at compile time.

```
Set
   i 'suppliers'
   j 'demand centers';

Parameter
   demand(j)
   supply(i)
   dist(i,j) 'distances';

$gdxIn transportation.gdx
$load i j demand supply dist

display i, j, demand, supply, dist;
```

DBTimestamp1.gms

In one application I had to retrieve data from the database each morning, at the first run of the model. The rest of the day, the data extracted that morning could be used. The following logic can implement this:

```
$onText
Retrieve data from data base first run each morning.
$offText

$onEcho > getdate.txt
I=%system.fp%transportation.mdb
Q=select day(now())
O=dbtimestamp.inc
$offEcho

$if not exist dbtimestamp.inc $call "echo 0 > dbtimestamp.inc"

Scalar dbtimestamp 'day of month when data was retrieved' /
$include dbtimestamp.inc
/;

Scalar currentday 'day of this run';
currentday = gday(jnow);

display "compare", dbtimestamp, currentday;

if(dbtimestamp <> currentday,
   execute '=gams.exe sr0 lo=3 gdx=transportation.gdx';
   abort$errorLevel "step 0 (database access) failed";
   execute '=mdb2gms.exe @getdate.txt'
);
```

The include file dbtimestamp.inc contains the day of the month (1,..,31) on which the data was extracted from the database. If this file does not exist, we initialize it with 0. We then compare this number with the current day of the month. If the numbers do not agree, we execute the database extraction step and rewrite the dbtimestamp.inc file. This last operation could be done using a `PUT` statement, but in this case we used an SQL statement.

The model [**DBTimestamp1**] is also part of the GAMS Data Utilities Library.

6.30.7 Command Files

Parameters can be specified in a command file. This is important if the length of the command-line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command-lines. Instead of specifying a long command-line as in:

```
$call =mdb2gms I="c:\My Documents\test.mdb" O="c:\My Documents\data.inc" Q="SELECT * FROM mytable"
```

we can use a command-line like:

```
$call =mdb2gms @"c:\My Documents\options.txt"
```

The command file

```
c:\My Documents\options.txt
```

can look like:

```
I=c:\My Documents\test.mdb
O=c:\My Documents\data.inc
Q=SELECT * FROM mytable
```

It is possible to write the command file from inside a GAMS model using the `$echo` command. The following example will illustrate this:

```
$set cmdfile "c:\windows\temp\commands.txt"
$echo "I=E:\models\labordata.mdb" >  "%cmdfile%"
$echo "O=E:\models\labor.INC"     >> "%cmdfile%"
$echo "Q=SELECT * FROM labor"     >> "%cmdfile%"
$call =mdb2gms @"%cmdfile%"

Parameter p /
$include "E:\models\labor.INC"
/;

display p;
```

Newer versions of GAMS allow the usage of the `$onEcho` and `$offEcho` commands:

```
$set cmdfile "c:\windows\temp\commands.txt"
$onEcho > "%cmdfile%"
I=E:\models\labordata.mdb
O=E:\models\labor.INC
Q=SELECT * FROM labor
$offEcho
$call =mdb2gms @"%cmdfile%"

Parameter p /
$include "E:\models\labor.INC"
/;

display p;
```

Note that the quotes enclosing strings with blanks like `Q=SELECT * FROM labor` can be omitted when using an external parameter file.

If a query becomes very long, it is possible to spread it out over several lines. To signal a setting will continue on the next line insert the character \ as the last character. E.g.:

```
Q=SELECT prod, loc, year, 'sales', sales FROM data \
UNION \
SELECT prod, loc, year, 'profit', profit FROM data
```

6.30.8 Notes

6.30.8.1 GDX Files

A GDX file contains GAMS data in binary format. The following GAMS commands will operate on GDX files: $gdxIn, $load, execute_load, execute_unload. The GDX=filename command-line argument will save all data to a GDX file. A GDX file can be viewed in the GAMS IDE using `File|Open`.

6.30.8.2 UNC Names

UNC means Unified Naming Convention. UNC names are a Microsoft convention to name files across a network. The general format is:

```
\\<server>\<share>\<path>\<file>
```

Examples:

```
\\athlon\c\My Documents\MDB2GMS.rtf
```

6.30.8.3 Quotes

Examples of handling of indices when the option B for quoting strings containing blanks is used:

6.30.8.4 $CALL Command

The `$call` command in GAMS will execute an external program at compile time. There are two forms:

The `$call` command in GAMS will execute an external program at compile time. There are two forms:

```
$call externalProgram
```

```
$call =externalProgram
```

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=' bypasses the command processor and directly executes the external program. We mention some of the differences:

1. Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE. If you want to execute these commands you will need to use the form `$call externalProgram` which uses the command processor.
2. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form `$call externalProgram`.
3. If it is important to stop with an appropriate error message if the external program does not exist, only use the form `$call =externalProgram`. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: `$call =externalProgram`.
4. When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the `START` command, as in: `$call start /w externalWindowsProgram`. Otherwise, it is preferred to use: `$call =externalWindowsProgram`.

Attention

In general it is recommended to use the `$call =externalProgram` version for its better error-handling.

When command line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

```
$call externalProgram parameter1 parameter2
$call =externalProgram parameter1 parameter2
```

The total length of the command line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

```
$call '"external program" "parameter 1" "parameter 2"'
$call ="external program" "parameter 1" "parameter 2"
```

It is noted that the first form needs additional quotes around the whole command line due to bugs in the parsing of the $call in GAMS. The second form work without additional quotes *only if* the = appears outside the double quotes.

6.30.8.5 Compile Time Commands

All $ commands in GAMS are performed at compile time. All other statements are executed at execution time. This means that a compile time command will be executed **before** an execution time command, even if it is below. As an example consider:

```
File batchfile / x.bat /;
putClose batchfile "dir"/;
$call x.bat
```

This fragment does not work correctly as already during compilation, the `$call` is executed, while the put statements are only executed after the compilation phase has ended and GAMS has started the execution phase. The above code can be fixed by moving the writing of the batch file to compilation time as in

```
$echo "dir" > x.bat
$call x.bat
```

or by moving the external program invocation to execution time:

```
File batchfile / x.bat /;
putClose batchfile "dir"/;
execute x.bat;
```

Notice that all $ commands do not include a semi-colon but are terminated by the end-of-line.

6.31 MessageReceiverWindow

6.31.1 Introduction

`MessageReceiverWindow` is a graphical tool that receives and displays Windows messages. The tool can either be used with the put_utility feature or with an arbitrary client (e.g. a Python program) that sends messages to a previously spawned instance of `MessageReceiverWindow.exe`.

6.31.2 Usage

An instance of `MessageReceiverWindow.exe` can be started as follows:

```
MessageReceiverWindow <title>
```

The optional argument `title` specifies the title of the `MessageReceiverWindow` instance. If the argument is omitted, the instance gets the default title `Form1`. Windows messages of type `WM_COPYDATA` that are send to the spawned instance, appear in its message log. The `Copy To Clipboard` button allows to copy the content of the message log to the clipboard. The `Save As ...` button can be used to save the content to a file.

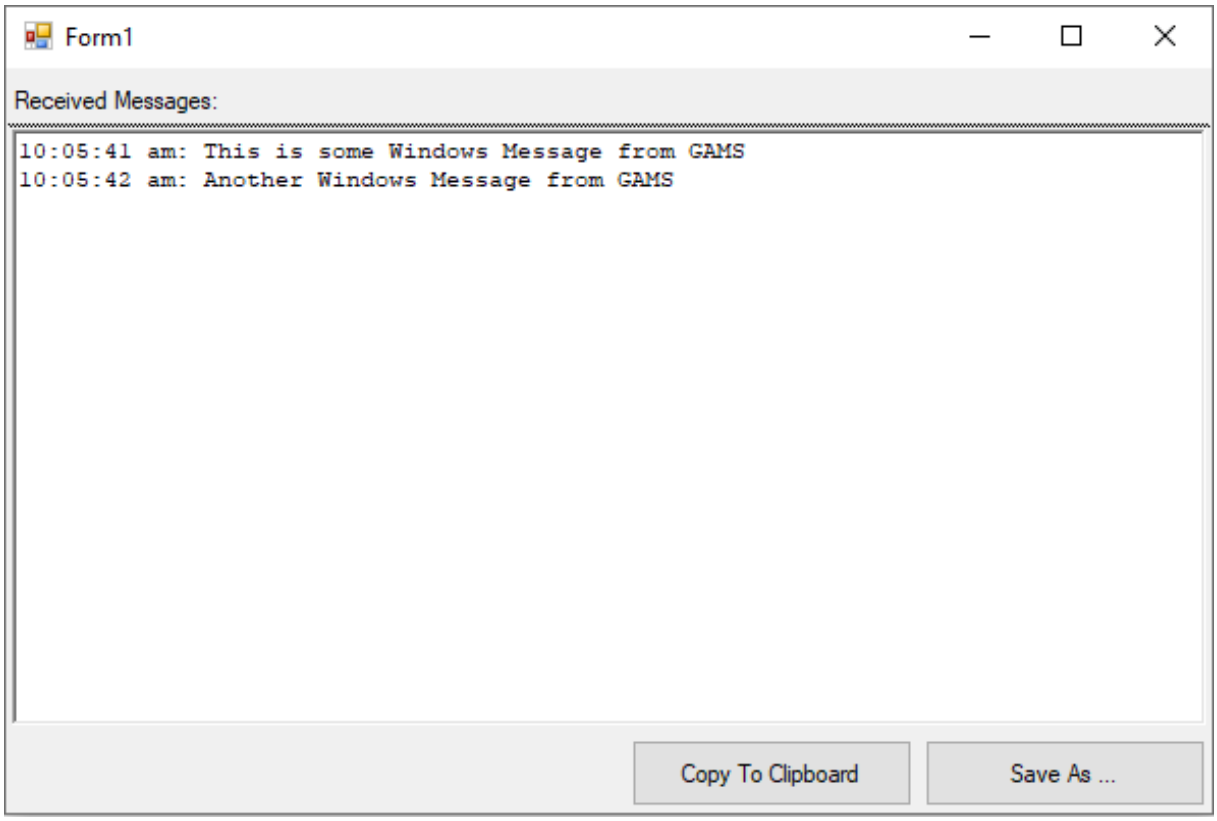


Figure 6.63 MessageReceiverWindow during the execution of Test Library model `MRW01`.

6.31.3 Special Commands

While arbitrary text messages are diplayed in the message log, there are special commands for controlling special behavior of an instance of `MessageReceiverWindow`:

Message	Action
@CopyToClipboard	Copies the content of the message log to the clipboard
@SaveAs [file]	Writes the content of the message log to [file]
@Terminate	Terminates the MessageReceiverWindow executable

6.31.4 Usage With put_utility

The put_utility feature offers a convenient integration of `MessageReceiverWindow`. The example **MRW01** shows how this works. First of all, an instance of MessagereceiverWindow needs to be spawned from within GAMS:

```
$set title Form1
Execute.Async "MessageReceiverWindow.exe %title%";
```

As soon as the window is ready, text messages or special commands can be sent by using put_utility with the command winMsg:

```
file f;
put_utility f 'WinMsg' / '%title%' / 'This is some Windows Message from GAMS';
```

By sending @Terminate, the instance can be terminated:

```
put_utility f 'WinMsg' / '%title%' / '@Terminate';
```

6.31.5 Usage With Python

The example **MRW02** from the GAMS Test Library demonstrates the use of MessageReceiverWindow from within Python. While it uses the Python programming language from within an Embedded Code section, other programming languages can be used as long as they provide access to Windows messages.

First of all, a ctypes.Structure derived class is defined which is used for sending Windows message to a MessageReceiverWindow instance:

```
class _CopyDataStruct(ctypes.Structure):
    _fields_ = [('dwData', ctypes.c_char_p),
                ('cbData', ctypes.c_ulong),
                ('lpData', ctypes.c_char_p)]
```

The function sendMessage() sends a message to the window with the title windowName. It makes use of functions FindWindowA and SendMessageA from ctypes.windll.user32:

```
def sendMessage(windowName, message):
    cs = _CopyDataStruct()
    receiver = ctypes.windll.user32.FindWindowA(None, bytes(windowName, 'utf-8'))
    cmd = bytes(message, 'utf-8')
    cs.dwData = 1
    cs.cbData = len(cmd)+1
    cs.lpData = cmd
    WM_COPYDATA = 0x4A
    ctypes.windll.user32.SendMessageA(receiver, WM_COPYDATA, 0, ctypes.byref(cs))
```

Popen from the subprocess module is used in order to start an instance of MessageReceiverWindow. The system_directory property of a GamsWorkspace instance from the GAMS Python API is used for finding a GAMS system directory:

```
windowName = 'MRW Python'
p = Popen([os.path.join(GamsWorkspace().system_directory, 'MessageReceiverWindow.exe'), windowName])
```

As soon as the window is ready, text messages or special commands can be sent by using the previously defined sendMessage function:

```
sendMessage(windowName, "This is some Windows Message from Python")
```

Sending @Terminate will terminate the running MessageReceiverWindow instance:

```
sendMessage(windowName, "@Terminate")
```

6.32 MODEL2TEX

A Tool to generate a documentation from GAMS source code in LaTeX format.

Author

Clemens Westphal

Special thanks to Ingmar Schlecht for supporting us and sharing his gamsToLatex implementation.

6.32.1 Introduction

MODEL2TEX is a tool to generate a documentation from GAMS source code in LaTeX format. This LaTeX output can then be further processed in order to generate pretty output files like PDF. The tool can be found in the root directory of GAMS. The tool allows to document one specific model symbol inside of a GAMS program. The resulting documentation contains two parts. The first part shows a list of symbols that are used by the model. The second part shows the actual algebra of the used equations and information about the types of used variables. An optional third part can contain additional notes.

6.32.2 Usage

MODEL2TEX is a command line tool. The general command line usage is as follows:

```
model2tex baseName [-h] [-m MODEL] [-f] [-o OUTPUT]
```

Since it operates on output files generated by GAMS, the first step is to generate the required files using the docfile option. In order to to that, execute the following command line:

```
gams myModel.gms docfile=myModel
```

The second step is to call MODEL2TEX: model2tex is distributed as Python source code (model2tex.py) and is executed via model2tex.cmd on Windows and model2tex.sh on all other platforms.

Windows:

```
model2tex myModel [-m MODEL] [-f] [-o OUTPUT]
```

Linux/macOS:

```
model2tex.sh myModel [-m MODEL] [-f] [-o OUTPUT]
```

The output file myModel.tex can be further processed for example by calling pdflatex in order to generate a PDF file.

```
pdflatex myModel.tex
```

6.32.3 Options

The following parameters can be used when calling MODEL2TEX:

Parameter	Description
-m model	Since MODEL2TEX generates documentation for one model symbol, the model name needs to be specified explicitly, if the GAMS file contains more than one model symbol
-f	MODEL2TEX does not allow the occurrence of suffixes in equations. This option can be used in order to force the creation of the tex file by skipping the checks for suffixes.
-o output	Use this name for the generated TeX file instead of the base name.

6.32.4 Using a JSON style file

MODEL2TEX automatically creates a JSON file that can be modified in order to customize the output. If you want to get the default settings back, just delete the generated JSON file and let MODEL2TEX create it again. The following list shows the available options in the JSON file:

Parameter	Default	Description
fontSize	11	The size of the used font
hrules	true	Horizontal rules are added between equations.
colors	black	Specifies the used colors in equations for variables, parameters, and sets.
landscape	false	Allows to change the page format to landscape.
noPowerFunc	false	Beside a power operator, GAMS offers several power functions. Setting this option to true will replace all power functions with the power operator.
reduceFrac	false	Try to resolve unbalanced fractions. This means that fractions with an unbalanced length of denominator and numerator will be changed in order to shrink the fraction.
reduceFracRatio	5.0	This number has only effect when reduceFrac is set to true. It specifies the ratio between denominator and numerator that is used as a threshold for reduceFrac.
latexDescription	false	Explanatory text for symbols is expected to be arbitrary text. In order to display the text correctly in LaTeX, some automatic adjustments are applied. If all explanatory text already contains valid LaTeX strings, this option can be enabled.
nameMap	the original names	This map allows to specify alternative names for symbols.
extraSymbols	empty list	On default, only the required symbols in a GAMS model symbol are contained in the generated LaTeX file. This list can be used in order to specify further symbols that should be included as well.
notes	empty list	Allows to specify additional notes for the documentation. Each list element will result in a new line.

6.32.5 Example

This example creates a PDF file for the pump model from the GAMS Model Library.

1. Retrieve model pump from GAMS Model Library

   ```
   gamslib pump
   ```

2. Generate required doc files

   ```
   gams pump.gms docfile=pump
   ```

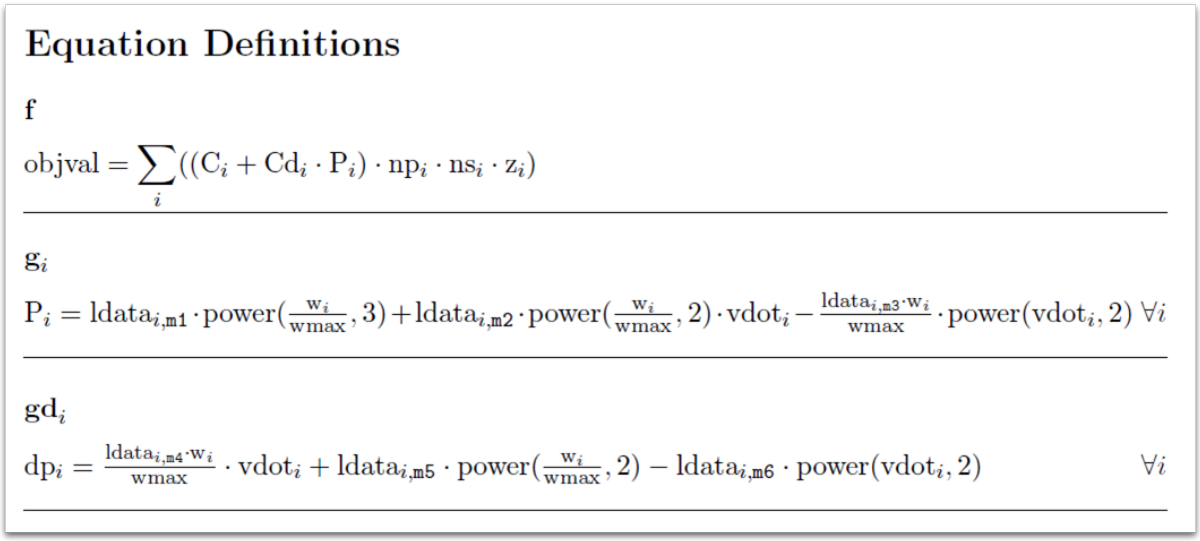
3. Generate the LaTex files

   ```
   model2tex pump -m=pump
   ```

4. Generate PDF documentation

   ```
   pdflatex pump.tex
   ```

The following pictures show snippets from the resulting PDF file:



Equation Definitions

f

$$\text{objval} = \sum_i ((\text{C}_i + \text{Cd}_i \cdot \text{P}_i) \cdot \text{np}_i \cdot \text{ns}_i \cdot \text{z}_i)$$

---

g_i

$$\text{P}_i = \text{ldata}_{i,\texttt{m1}} \cdot \text{power}(\tfrac{\text{w}_i}{\text{wmax}}, 3) + \text{ldata}_{i,\texttt{m2}} \cdot \text{power}(\tfrac{\text{w}_i}{\text{wmax}}, 2) \cdot \text{vdot}_i - \tfrac{\text{ldata}_{i,\texttt{m3}} \cdot \text{w}_i}{\text{wmax}} \cdot \text{power}(\text{vdot}_i, 2) \;\forall i$$

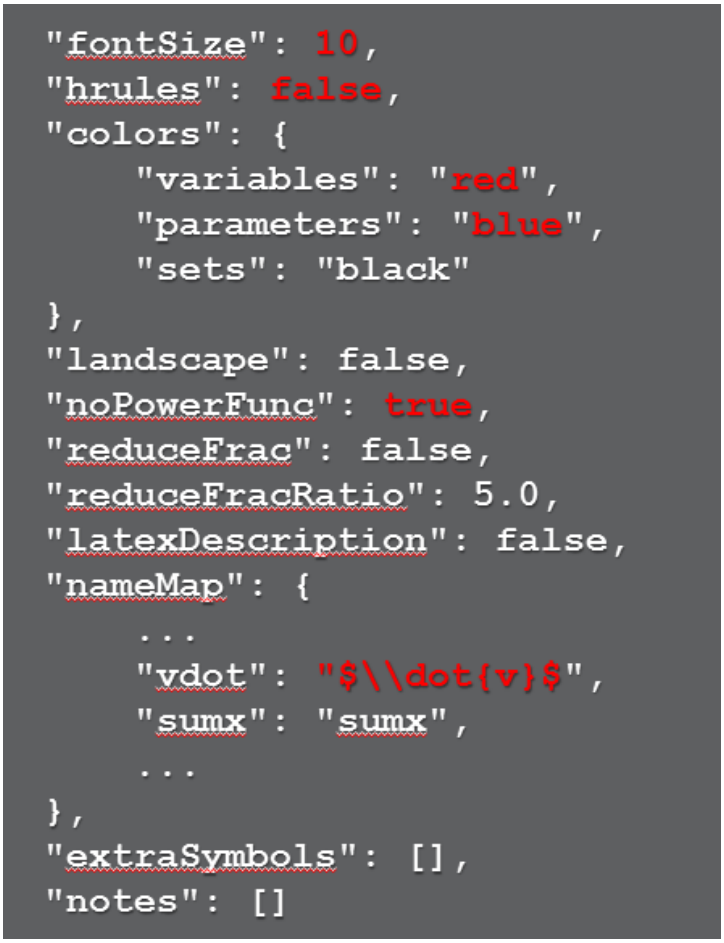
---

gd_i

$$\text{dp}_i = \tfrac{\text{ldata}_{i,\texttt{m4}} \cdot \text{w}_i}{\text{wmax}} \cdot \text{vdot}_i + \text{ldata}_{i,\texttt{m5}} \cdot \text{power}(\tfrac{\text{w}_i}{\text{wmax}}, 2) - \text{ldata}_{i,\texttt{m6}} \cdot \text{power}(\text{vdot}_i, 2) \qquad \forall i$$

---

5. Customizing the JSON style file In order to change the appearance of the PDF file, the generated JSON file `pump.json` can be modified. Changing the options in `pump.json` according to the picture and executing the commands from step 3 and 4 again will result in a customized PDF output.



```
"fontSize": 10,
"hrules": false,
"colors": {
    "variables": "red",
    "parameters": "blue",
    "sets": "black"
},
"landscape": false,
"noPowerFunc": true,
"reduceFrac": false,
"reduceFracRatio": 5.0,
"latexDescription": false,
"nameMap": {
    ...
    "vdot": "$\\dot{v}$",
    "sumx": "sumx",
    ...
},
"extraSymbols": [],
"notes": []
```

Variables

Name	Domains	Description
P	i	power output of pumps on level i
w	i	rotation speed for pumps on level i
dp	i	pressure rise on level i
$\dot{v}$	i	flow through pumps on level i

Equation Definitions

f

$$\text{objval} = \sum_i ((\text{C}_i + \text{Cd}_i \cdot \text{P}_i) \cdot \text{np}_i \cdot \text{ns}_i \cdot \text{z}_i)$$

g_i

$$\text{P}_i = \text{ldata}_{i,\text{m1}} \cdot \frac{\text{w}_\text{f}}{\text{wmax}}^3 + \text{ldata}_{i,\text{m2}} \cdot \frac{\text{w}_\text{f}}{\text{wmax}}^2 \cdot \dot{v}_i - \frac{\text{ldata}_{\text{f},\text{m3}} \cdot \text{w}_\text{f}}{\text{wmax}} \cdot \dot{v}_i^2 \qquad \forall i$$

gd_i

$$\text{dp}_i = \frac{\text{ldata}_{\text{f},\text{m4}} \cdot \text{w}_\text{f}}{\text{wmax}} \cdot \dot{v}_i + \text{ldata}_{i,\text{m5}} \cdot \frac{\text{w}_\text{f}}{\text{wmax}}^2 - \text{ldata}_{i,\text{m6}} \cdot \dot{v}_i^2 \qquad \forall i$$

6.33 MPS2GMS

Translates an MPS or LP file into an equivalent short generic GAMS program using a GDX file to store data. A number of solver specific extensions are recognized.

The MPS file is first attempted to be read in free format (no spaces in names). If it seems that row or column names have spaces but have at most 8 characters, parsing in fixed format is attempted. The MPS and LP readers of the HiGHS solver are utilized. GAMS Data Exchange (GDX) files and matching GAMS source is written.

Notes:

- Row, column, SOS (special ordered sets), and cone names can have up to 63 characters and are tested for case sensitivity and uniqueness because these names become GAMS set members. If the test fails, the tool attempts to modify the names.
- MPS files may contain multiple N rows. The first one will be used as objective function and remaining ones will be ignored.
- If no optimization direction is given in a MPS file, minimization will be assumed.
- Variables in SOS must be continuous.
- SOS must not overlap. A variable cannot appear multiple times in the same SOS.

The MPS and LP formats have been extended in different forms by several solver vendors. Not all extensions are supported by MPS2GMS.

The following MPS sections are recognized by the free format parser:

- NAME
- OBJSENSE, MAX, MIN
- ROWS
- COLUMNS
- RHS
- BOUNDS
- RANGES
- QUADOBJ, QMATRIX
- QSECTION, QCMATRIX
- CSECTION with cone types QUAD and RQUAD
- SETS, SOS
- ENDATA

The fixed format parser also recognizes these sections except for QMATRIX, QSECTION, QCMATRIX, CSECTION, SETS, and SOS.

For a description of the MPS format, see also

- the CPLEX manual,
- the FICO Xpress manual,
- the GUROBI manual,
- the MOSEK manual.

The following LP format features are recognized:

- Comments
- Single objective function: linear or quadratic only
- Constraints (`Subject To` section): linear or quadratic only
- Variable bounds and types (`Bounds`, `Binaries`, `Generals`, `Semis` section); to specify semi-integer variables, they need to appear in sections `Generals` and `Semis`
- Special ordered sets (`SOS` section)

For a description of the LP format, see also

- the CPLEX manual
- the FICO Xpress manual
- the GUROBI manual
- the MOSEK manual

6.33.1 Usage

```
mps2gms mpsfile | lpfile [ gdxfile [ gmsfile ] ] { key=value }
```

If the first parameter is not of the form key=value and ends with `.lp` or `.lp.gz`, it is assumed to be the name of an LP file, otherwise it is assumed to be the name of an MPS file. If no GDX filename is given, the name of the MPS or LP file is used, with `.mps`/`.lp` (or `.mps.gz`/`.lp.gz`) extension replaced by `.gdx`, or `.gdx` appended. If no GMS filename is given, the name of the GDX file is used, with `.gdx` extension replaced by `.gms`, or `.gms` appended.

Guide to parameters:

Key	Description
MPS	Name of MPS input file, can be compressed with GZIP
LP	Name of LP input file, can be compressed with GZIP
GDX	Name of GDX output file
GMS	Name of GAMS program output file
CEQUATIONS	Whether to write quadratic cones from CSECTION sections into deprecated =C= equations. Possible values: 0, N, 1, Y. Default: 0
COLUMNINTVARSAREBINARY	Whether integer variables that appear first in COLUMNS section should be assumed to be binary variables. Possible values: 0, N, 1, Y. Default: 0. Note that modern solvers assume binary variables, but for backward compatibility the default is set to the original IBM interpretation.
DUPLICATES	Whether to check and how to handle multiple coefficients for the same variable or product in LP files. If set to NOCHECK, then no checks for multiple coefficients are done. In this case, writing the GDX file may fail. For performance reasons, this is the default. If set to ADD, then coefficient are added up. If set to IGNORE, then additional coefficients are ignored and a warning is printed to the log. If set to ERROR, then warnings are printed to the log and MPS2GMS terminates with a nonzero return code.

6.34 MSAppAvail

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

Checks if a MS Office Application is available. The return code (errorLevel) signals if the application is available (0) or not (not 0).

6.34.1 Usage

Command line:

```
gamstool [win32.]MSAppAvail Application
```

Compile time:

```
$callTool [win32.]MSAppAvail Application
```

Execution time:

```
executeTool '[win32.]MSAppAvail Application';
```

Argument	Description
Application	MS Office application to check.

6.34.2 Example: Checking whether MS Access is available

This example checks if Microsoft Access is available on the system and quit without error if this is not the case.

```
$callTool win32.msappavail Access
$if errorlevel 1 $abort.noError "No MS Access available"
```

6.35 Ordinary Least Squares (OLS)

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This estimates the unknown parameters in a linear regression model. The set `i` are the observations, the set `p` are the estimates. The matrix `A(i,p)` contains the explanatory variable and `y(i)` the dependent variable. On return the symbol `estimate(p)` will contain the estimated statistical coefficients. Other statistical information parameters are available.

6.35.1 Usage

Command line:

```
gamstool [linalg.]OLS i p A y est [covar=id] [df=id] [fitted=id] [r2=id] [resid=id] [resvar=id] [rss=
```

Compile time:

```
$callTool [linalg.]OLS i p A y est [covar=id] [df=id] [fitted=id] [r2=id] [resid=id] [resvar=id] [rs
```

Execution time:

```
executeTool '[linalg.]OLS i p A y est [covar=id] [df=id] [fitted=id] [r2=id] [resid=id] [resvar=id]
```

Where:

Argument	Description
i	Name of set of observations i(*).
p	Name of set of estimates p(*).
A	Name of two-dimensional explanatory variable matrix A(i,p).
y	Name of one-dimensional dependent variable y(i).
est	Name of one-dimensional estimated statistical coefficients est(p).

The following names parameters are available:

Parameter	Description
covar=id	Statistical info: variance-covariance matrix CoVar(p,p)
df=id	Statistical info: degrees of freedom (scalar)
fitted=id	Statistical info: fitted values for dependent variable fitted(i)
r2=id	Statistical info: R squared (scalar)
resid=id	Statistical info: residuals resid(i)
resvar=id	Statistical info: residual variance (scalar)
rss=id	Statistical info: residual sum of squares (scalar)
se=id	Statistical info: standard errors se(p)
sigma=id	Statistical info: standard error (scalar)
tval=id	Statistical info: standard errors se(p)
gdxIn=fileIn.gdx	Name of GDX file that contains symbols i, p, A, and y. Mandatory if called from the command line, otherwise optional.
gdxOut=fileOut.gdx	Name of GDX file that contains symbol est and statistical information parameters after execution. Mandatory if called from the command line, otherwise optional.

6.35.2 Example

For an example see model [**LeastSquares**] in the GAMS Data Utilities Library.

6.36 GAMS Posix Utilities

Starting with distribution 20.6 the GAMS system for Windows includes a collection of Posix utilities which are usually available for the different Unix systems and therefore help to write platform independent scripts. The following utilities are available:

- awk - Pattern scanning and processing language (1)
- cat - Concatenate and print files
- cksum - Write file checksums and sizes
- cmp - Compare two files

- comm - Select or reject lines common to two files
- cp - Copy files
- cut - Cut out selected fields of each line of a file
- diff - Compare two files
- expr - Evaluate arguments as an expression
- fold - Fold lines
- gsort - Sort, merge, or sequence check text files (1)
- grep - File pattern searcher
- gdate - Write the date and time (1)
- gunzip - File decompression
- gzip - File compression
- head - Copy the first part of files
- join - Relational database operator
- make - Build automation tool
- mkdir - Create a new directory
- mv - Move files
- od - Dump files in various formats
- paste - Merge corresponding or subsequent lines of files
- printf - Write formatted output
- rm - Remove directory entries
- sed - Stream editor
- sleep - Suspend execution for an interval
- tail - Copy the last part of a file
- tar - Collect files into one archive file
- tee - Duplicate standard input
- test - Evaluate expression
- touch - Update access date and/or modification date of a file or directory
- tr - Translate characters
- uniq - Report or filter out repeated lines in a file
- wc - Word, line, and byte count
- xargs - Construct argument list(s) and invoke utility

(1) Please note that the utilities "date" and "sort" have been renamed to "gdate" and "gsort" to avoid conflicts with the Windows commands "date" and "sort". For compatibility reasons the GNU implementation of awk called "gawk" has been renamed to "awk".

The collection consists of native Windows ports of GNU implementation of these utilities taken from the Sourceforge. Detailed descriptions of the utilities can be found at the GNU website.

The Posix tools and in particular the awk utility can be used to transform a variety of different text inputs into GAMS readable input files. Examples can be found in the **GAMS Model Library** (Subject "GAMS Tools").

6.37 Rank

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This sorts one dimensional symbol `sym` and stores sorted indices in one dimensional parameter `symIdx`.

6.37.1 Usage

Command line:

```
gamstool [alg.]Rank sym symIdx gdxIn=fileIn.gdx gdxOut=fileOut.gdx
```

Compile time:

```
$callTool [alg.]Rank sym symIdx [gdxIn=fileIn.gdx] [gdxOut=fileOut.gdx]
```

Execution time:

```
executeTool '[alg.]Rank sym symIdx [gdxIn=fileIn.gdx] [gdxOut=fileOut.gdx]';
```

Where:

Argument	Description
`sym`	Name of parameter or set to be sorted `sym(*)`
`symIdx`	Name of parameter containing sort indexes `symIdx(*)`

The following named parameters are available:

Parameter	Description
`gdxIn=fileIn.gdx`	Name of GDX file that contains symbol sym. Mandatory if called from the command line, otherwise optional.
`gdxOut=fileOut.gdx`	Name of GDX file that contains symbol symIdx after execution. Mandatory if called from the command line, otherwise optional.

6.37.2 Example

```
set I /i1 * i6/;
```

```
parameter A(I) /i1=+Inf, i2=-Inf, i3=Eps, i4= 10, i5=30, i6=20/;
parameter AIndex(i) 'permutation index';

* sort symbol; result in parameter AIndex
$callTool.checkErrorLevel 'alg.Rank A AIndex';
display AIndex;
```

The complete example is also part of the GAMS Data Utilities Library, see model [**GDXRANKExample18**] for reference. The libInclude rank.gms uses this tool internally.

6.38 SCENRED

6.38.1 Release Notes

- May, 2002: Level 001 (GAMS Distribution 20.6)
  - GAMS/SCENRED introduced.

6.38.2 Introduction

Stochastic programs with recourse employing a discrete distribution of the random parameters become a deterministic programming problem. They can be solved by an appropriate optimization algorithm, ignoring the stochastic nature of (some or all) parameters.

SCENRED is a tool for the reduction of scenarios modeling the random data processes. The scenario reduction algorithms provided by SCENRED determine a scenario subset (of prescribed cardinality or accuracy) and assign optimal probabilities to the preserved scenarios. The reduced problem is then solved by a deterministic optimization algorithm provided by GAMS.

6.38.3 Scenario Reduction Algorithms

Many solution methods for stochastic programs employ discrete approximations of the uncertain data processes by a set of scenarios (i.e., possible outcomes of the uncertain parameters) with corresponding probabilities.

For most practical problems the optimization problem that contains all possible scenarios (the so-called deterministic equivalent program) is too large. Due to computational complexity and to time limitations this program is often approximated by a model involving a (much) smaller number of scenarios.

The reduction algorithms developed in [1,2] determine a subset of the initial scenario set and assign new probabilities to the preserved scenarios. All deleted scenarios have probability zero.

SCENRED contains three reduction algorithms: The Fast Backward method, a mix of Fast Backward/Forward methods and a mix of Fast Backward/Backward methods. In general, the computational performance (accuracy, running time) of the methods differ. For huge scenario trees the Fast Backward method has the best expected performance with respect to running time. The results of the Forward and Backward methods are more accurate, but at the expense of higher computing time. The Forward method is the best algorithm when comparing accuracy, but it can only be recommended if the number of preserved scenarios is small (strong reduction). The combined methods improve the result of the Fast Backward method if the Forward or Backward method, respectively, can be completed within the running

time limit. If no reduction method is selected, the method with the best expected performance with respect to running time is chosen.

The reduction algorithms exploit a certain probability distance of the original and the reduced probability measure. The probability distance trades off scenario probabilities and distances of scenario values. Therefore, deletion will occur if scenarios are close or have small probabilities.

The reduction concept is general and universal. No requirements on the stochastic data processes (e.g. the dependency or correlation structure of the scenarios, the scenario probabilities or the dimension of the process) or on the structure of the scenarios (e.g. tree-structured or not) are imposed. The reduction algorithms can be tailored to the stochastic model if the user provides additional information (How many decision stages are involved? Where do the random parameters enter the model – in objective and/or right hand sides and/or technology matrices?) The information is used to choose the probability distances (cf. Remark 1 in [1]).

References:

1. J. , N. Gröwe-Kuska, W. Römisch: Scenario reduction in stochastic programming: An approach using probability metrics. Revised version to appear in Mathematical Programming.
2. H. Heitsch, W. Römisch: Scenario reduction algorithms in stochastic programming. Preprint 01-8, Institut für Mathematik, Humboldt-Universität zu Berlin, 2001.

6.38.4 Using GAMS/SCENRED

The reduction algorithms require additional data preparation and reformulation of the GAMS program for the stochastic programming model.

GAMS offers great flexibility with respect to the organization of data specification, model definition and solve statements. The most common way to organize GAMS/SCENRED programs is shown below. Since the initial scenarios and a number of input parameters have to be passed to SCENRED, the corresponding components of the GAMS program have to be defined before the SCENRED call. The reduced scenarios have to be defined before the equations of the (reduced) stochastic programming model are used in a solve statement. Therefore the SCENRED call can be placed anywhere between the definitions of the GAMS parameters and the solve statement of the reduced stochastic programming model.

When building or modifying a model for use with GAMS/SCENRED the following steps should be taken:

- Analyse the GAMS program of the stochastic programming model.
  Since the initial scenarios and a number of input parameters have to be passed to SCENRED (see Section The SCENRED Input File), one must identify the corresponding components of the GAMS model and create or calculate them if they do not already exist.
- Reformulate the GAMS program.
  Check if the model can handle varying scenario or node probabilities, and whether the equations are defined in terms of a (possibly reduced) tree. If the model doesn't already contain a scenario tree, one should be added. If it does, it is a simple task to rewrite the equation definitions (and possibly other statements too) in terms of a subset of the original nodes or tree.
- Add the statements for passing the initial set of scenarios to SCENRED, for the execution of SCENRED and for the import of the reduced scenarios from SCENRED.

A reduction of the initial scenarios makes sense only if we are able to generate that part of the model that corresponds to the preserved scenarios (i.e. the reduced subtree). This is done by declaring a subset of the nodes in the original tree. The parameters and equations are declared over the original node set, but are defined over only the subtree. This will be illustrated by an example later in the section.

Further, one should verify that the model can handle changing probabilities. Many practical models involve scenarios with equal probabilities. This property will not be maintained by the probabilities in the reduced subtree.

ORGANIZATION OF GAMS/SCENRED PROGRAMS

Component	Contents
1. DATA	* set & parameter declarations and definitions * $libinclude scenred.gms * assignments displays
2. SCENRED CALL	* export the initial scenarios from GAMS to SCENRED * execute SCENRED * import the reduced scenarios from SCENRED to GAMS
3. MODEL	* variable declaration * equation declarations * equation definitions (using sets from reduced tree) * model definition & solution

Prior to calling SCENRED, you should include the declaration of the SCENRED input and output parameters and the definition of the sets they are indexed by from the GAMS include library:

```
$libinclude scenred.gms
```

Once you have created all the inputs to SCENRED and assigned values to them, you are ready to write the SCENRED GDX data input file, write the SCENRED options file, call SCENRED, and read the reduced tree data from the SCENRED GDX data output file (see Section The SCENRED Input File, Section SCENRED Options and the Option File, and Section The SCENRED Output File). Assuming your model is formulated to use a subtree of the original, you can now continue with the solve and any subsequent reporting.

SCENRED is executed by issuing the statement

```
execute 'scenred optfilename';
```

where `optfilename` is the name of the SCENRED option file.

As an example, consider the `srkandw` model in the GAMS model library, and the `kand` model upon which it is based (get these from the modlib now!). To produce `srkandw` from `kand`, we first reformulate the original to allow for solution over a reduced tree. To do this, we introduce a subset of the node set: `set sn(n) 'nodes in reduced tree';` For convenience and clarity, we introduce a second subset at the same time, the set of leaf nodes: `set leaf(n) 'leaf nodes in original tree';` as well as some code to compute this set based on the existing time-node mapping. We also declare a new parameter, the probabilities for the reduced tree: `parameter sprob(n) 'node probability in reduced tree';` Once these are declared, we can quickly edit the equation *definitions* so that they run only over the reduced subtree: we simply substitute the reduced probabilities `sprob` for the original `prob`, and the reduced node set `sn` for the original node set `n`. Note that the *declaration* of the equations does not change.

This example illustrates one other change that may be required: the stochastic data must be in parameters having the node set as their last index. This is not the case in the `kand` model, so we simply reversed the indices in the `dem` parameter to meet the requirement in `srkandw`. It is also possible to create a transposed copy of the original data and pass that the SCENRED if the original data cannot be changed conveniently.

6.38.5 The SCENRED Input File

The SCENRED input file contains the initial scenarios and their stochastic parameter data, as well as statistics describing this input and (possibly) options to control the SCENRED run. This input file has a special binary format; it is a GDX (GAMS Data Exchange) file. The name of the SCENRED input file is assigned in the option file (see Section SCENRED Options and the Option File).

The scalar inputs to SCENRED are collected in the one-dimensional parameter `ScenRedParms`, the first parameter stored in the SCENRED input file. Some of the elements of `ScenRedParms` are required (e.g. statistics for the input tree) while others are optional (e.g. the run time limit). SCENRED will stop if a required element is missing or out of range.

Table 1 Required `ScenRedParms` elements

Element	Description
num_leaves	the number of initial scenarios or leaves of the scenario tree (i.e., before the reduction)
num_nodes	number of nodes in the initial tree (the number of scenarios if not tree-structured)
num_random	Number of random variables assigned to a scenario or node, i.e., the dimension of the random data process
num_time_steps	Length of a path from the root node to a leaf of the scenario tree, i.e., the number of time steps involved

Table 2 Optional ScenRedParms elements

Element	Description	Default
red_num_leaves	specifies the desired number of preserved scenarios or leaves	none
red_percentage	specifies the desired reduction in terms of the relative distance between the initial and reduced scenario trees (a real between 0.0 and 1.0)	none
num_stages	Set the number of branching levels of the scenario tree, i.e., the number of stages of the model -1. Hence num_stages=1 if no branching occurs, i.e., the values of the scenarios differ for all time steps	1
where_random	An integer indicating where the randomness enters the model. The value is interpreted as a digit map computed using the formula 100∗inObj + 10∗inRHS + inMatrix, where inObj is 1 if the objective contains random parameters and 0 otherwise, inRHS is 1 if the right-hand side contains random parameters and 0 otherwise, and inMatrix is 1 if the constraint matrix contains random coefficients and 0 otherwise.	10 (random right-hand side)
reduction_method	Select a reduction method: 0: automatic (best expected performance with respect to running time) 1: Fast Backward method 2: Mix of Fast Backward/Forward methods 3: Mix of Fast Backward/Backward methods	0
run_time_limit	Defines a limit on the running time in seconds	none
report_level	Control the content of the SCENRED log file: 0: Standard SCENRED log file 1: Additional information about the tree	0

A few comments on the parameters red_percentage and red_num_leaves are in order. At least one of these values must be set. The value of red_percentage will be ignored if the parameter red_num_leaves is non-zero. Otherwise, the tree will not be reduced if red_percentage=0, while the reduction of the tree will be maximal (i.e. only one scenario will be kept) if red_percentage=1. A numeric value of 0.5 means that the reduced tree maintains 50% of the information contained in the original tree. The reduction algorithms are skipped if red_num_leaves=num_leaves or if red_num_leaves=0 and red_percentage=0. These values can be assigned if the user wishes to run the scenario tree diagnostic.

The second data element in the input file is the set of nodes making up the scenario tree. Note that the cardinality of this set is part of ScenRedParms.

The third data element is the ancestor mapping between the nodes. This mapping determines the scenario tree. Note that the mapping can be either an ancestor mapping (i.e. child-parent) or a successor mapping

(parent-child). By default, SCENRED expects an ancestor mapping. If the check for this fails, it looks for a successor mapping.

The fourth data element is the parameter of probabilities for the nodes in the original tree. It is only required that probabilities for the scenarios (i.e. the leaf nodes) be provided, but the parameter can contain probabilities for the non-leaf nodes as well.

The remaining elements in the input data file specify the parameter(s) that comprise the random values assigned to the initial scenarios, or to the nodes of the scenario tree. There can be more than one such parameter, included in any order. The only requirement is that the node set be the final index in each of these parameters.

Table 3 summarizes the content of the SCENRED input file. Please keep in mind that the order of the entries must not be altered!

Table 3 Content of the SCENRED Input File

No.	Symbol	Type	Dimension	Content
1	`ScenRedParms`	Parameter	1	scalar SCENRED input
2	(any name)	Set	1	nodes in the scenario tree
3	(any name)	Set	2	the ancestor set
4	(any name)	Parameter	1	node probabilities; at least for the leaves
≥ 5	(any name)	Parameter	≥ 1	random values assigned to the nodes

To create the SCENRED data input file, the GAMS `execute_unload` statement is used. This statement is used to transfer GAMS data to a GDX file at execution time. As an example, to create a GDX file with the 4 required input parameters and one parameter `demand` containing the stochastic data, you might have the following statement:

```
execute_unload 'sr_input.gdx', ScenRedParms, node, ancestor, prob, demand;
```

6.38.6 SCENRED Options and the Option File

When the SCENRED executable is run, it takes only one argument on the command line: the name of the SCENRED option file. The option file is a plain text file. Typically, it is used to specify at least the names of the SCENRED data input and output files. The option file must be created by the SCENRED user (typically via the GAMS put facility during the GAMS run). The syntax for the SCENRED option file is

`optname value` or `optname = value`

with one option on each line. Comment lines start with an asterix and are ignored.

Some of the SCENRED options may be specified in two places: as elements of the `ScenRedParms` parameter of the SCENRED input file, or as entries in the options file. These parameters have been summarized in Table 2. If an option is set in both these places, the value in the option file takes precedence of over the value from `ScenRedParms`. In addition, the parameters in Table 4 can only be specified in the option file.

Table 4 Options - optfile only

Option	Description	Default
`input_gdx`	Name of the SCENRED data input file	`xllink.gdx`
`output_gdx`	Name of the SCENRED data output file	`scenred.gdx`
`log_file`	Name of the SCENRED log file	`scenred.log`

6.38.7 The SCENRED Output File

The SCENRED output file contains the reduced scenario tree and the `ScenRedReport` parameter. Like the input file, the output file has a special binary format; it is a GDX (GAMS Data Exchange) file.

The first data element in the output file is the `ScenRedReport` parameter containing the scalar outputs and statistics from the SCENRED run. The elements of this parameter are summarized in Table 5. The second data element is the parameter containing the probabilities of the nodes in the reduced scenario tree. These node probabilities are required to construct the reduced tree. The third and final data element is the ancestor map for the reduced scenario tree. This map can be read from the GDX file, or the reduced tree can be built from the original one by using the reduced probabilities. The content of the data output file is summarized in Table 6.

Table 5 `ScenRedReport` elements

Element	Description
`ScenRedWarnings`	number of SCENRED warnings
`ScenRedErrors`	number of SCENRED errors
`run_time`	running time of SCENRED in sec.
`orig_nodes`	number of nodes in the initial scenario tree
`orig_leaves`	number of leaves (scenarios) in the initial scenario tree
`red_nodes`	number of nodes in the reduced scenario tree
`red_leaves`	number of leaves(scenarios) in the reduced tree
`red_percentage`	relative distance of initial and reduced scenario tree
`red_absolute`	absolute distance between initial and reduced scenario tree
`reduction_method`	reduction method used: 0: the program stopped before it could select a method 1: Fast Backward method 2: Mix of Fast Backward/Forward methods 3: Mix of Fast Backward/Backward methods

Table 6 Content of the SCENRED Output File

No.	Symbol	Type	Dimension	Content
1	`ScenRedReport`	Parameter	1	report of the SCENRED run
2	`red_prob`	Parameter	1	node probabilities for the reduced scenarios
3	`red_ancestor`	Set	2	the ancestor map for the reduced scenarios

To read the SCENRED data output file, the GAMS `execute_load` statement is used. This statement is used to transfer GDX data to GAMS at execution time. As an example, to read a GDX file named `sr_output.gdx` created by SCENRED, you might have the following statement:

```
execute_load 'sr_output.gdx', ScenRedReport, sprob=red_prob, sanc=red_ancestor;
```

In the statement above, the equal sign `=` is used to indicate that the data in the GDX parameter `red_prob` should be read into the GAMS parameter `sprob`, and the data in the GDX set `red_ancestor` should be read into the GAMS set `sanc`.

6.38.8 Diagnostic Check of Scenario Trees

When SCENRED reads its input data, it performs a number of checks to verify that the data is correct. The diagnostic checks of the input parameters include:

- consistency of the desired input parameters with the contents of the SCENRED input file (number of nodes, number of leaves, number of time steps, number of random values assigned to a node)
- range check of desired input parameters and options
- check of scenario and node probabilities
- check of the ancestor matrix (check the orientation of the graph, check if the graph contains a cycle, check if the graph contains incomplete forests or scenarios, check the consistency of the parameter `num_time_steps` with the ancestor matrix)

The following errors in the specification of the scenario tree cause SCENRED to skip the reduction algorithms:

- The input files cannot be opened.
- Not all required input parameters are given.
- The required input parameters are not consistent with the contents of the SCENRED input file.
- The required input parameters are out of range.
- Missing or negative scenario probabilities (probabilities of leaves).
- The ancestor set contains too many entries (more than `2*num_nodes`).
- SCENRED detects a cycle in the ancestor set.
- SCENRED detects incomplete scenarios in the ancestor set.
- Run time limit reached

6.38.9 SCENRED Errors and Error Numbers

When SCENRED encounters a serious error in the input files or in the scenario tree, it sends an error message to the screen and to the log file. These messages always start with

```
**** SCENRED run-time error ...
```

The number of SCENRED errors are contained in the parameter `ScenRedReport` of the SCENRED output file (if it could be created). The occurrence of an error can also be detected from the last line that SCENRED sends to the screen:

```
**** SCENRED ErrCode=...
```

The numerical values of `ErrCode` and their meaning are given below.

ErrCode	Meaning
1	(for internal use)
2	fatal error while reading from SCENRED input file
3	fatal error while writing to SCENRED output file
4	fatal error while reading from SCENRED option file

ErrCode	Meaning
5	log file cannot be opened
6	a memory allocation error occurred
7	there are missing input parameters
8	could not access the GAMS names for the nodes
9	(for internal use)
10	ancestor set not given or contains too many entries
11	node probabilities cannot be not read or are wrong
12	random values for the nodes cannot be read
13	input parameters are out of range
14	ancestor set contains a cycle
15	incomplete scenarios or forests detected
16	fatal error in reduction algorithm (not enough memory)
17	running time limit reached

6.38.10 SCENRED Warnings

SCENRED warnings are caused by misspecification of the initial scenarios that can be possibly fixed. When SCENRED encounters such an error in the input files or in the scenario tree, it sends a message to the screen and to the log file. These messages always start with

```
**** SCENRED Warning ...
```

The following list gives an overview of the cases that produce warnings, and the action taken by SCENRED in these cases.

- The user assigned an option value that is out of range.
  Action: Assign the default value.

- Both parameters `red_num_leaves` and `red_percentage` are assigned nontrivial values.
  Action: The value of `red_percentage` will be ignored.

- The scenario probabilities (probabilities of leaves) do not sum up to 1.
  Action: The scenario probabilities are rescaled. Assign new probabilities to the remaining (inner) nodes that are consistent with the scenario probabilities.

- Missing probabilities of inner nodes.
  Action: Assign node probabilities that are consistent with the scenario probabilities.

- The ancestor set contains more than one ancestor for a node.
  Action: SCENRED assumes to be given a successor set instead of an ancestor set (i.e., the transpose of an ancestor matrix. This means that the graph corresponding to the ancestor set has the wrong orientation). SCENRED starts the tree diagnostic for the successor set. The reduced tree will be defined in terms of a successor set as well (if the successor set passes the tree diagnostic and if SCENRED locates no fatal error during the run).

- The fast backward method delivered a result, but the result cannot be improved by the forward or backward method (running time limit reached).
  Action: Use the result of the fast backward method.

6.39 SCENRED2

6.39.1 Introduction

Scenred2 is a fundamental update of the well-known scenario reduction software Scenred. A lot of new features come along with the latest release version. Beside updates and extensions concerning the control of the scenario reduction action an all new device for scenario tree construction has been implemented in Scenred2. Moreover, a lot of visualization functions to plot scenario trees and scenario processes linked to the free Gnuplot plotting software are available with Scenred2 now.

Table: Summary of basic new functions in Scenred2

Description	Section
Additional options for controlling the scenario reduction	Scenario Reduction
New device of scenario tree construction	Scenario Tree Construction
Visualization of scenario trees and processes	Visualization
Command line interface and data export	Command Line Interface

6.39.2 Using Gams/Scenred2

Successful applying Scenred or Scenred2 requires a special formulation of the stochastic programming model within the Gams program. Probabilistic information must be given by a set of nodes implying a certain ancestor structure including a well-defined root node. Note that the usage of Gams/Scenred2 is basically the same as the usage of Gams/Scenred. Hence, it is recommended for new users to look at the Scenred documentation first. All details about how to organize your Gams program, how to run Scenred from the Gams program by using the gdx interface, and, of course, examples can be found in that documentation.

The Gams/Scenred2 link provides the same gdx interface. But, due to new features some small changes in controlling the options are needed. Scenred2 supports now two types of option files. The first one is the SR-Command-File which must be passed to Scenred2 together with the Scenred2 call. The second one, the SR-Option-File includes more specific options to control the selected scenario reduction or scenario construction methods and can be declared in the SR-Command-File.

6.39.2.1 SR-Command-File

The command file includes the basic specifications. These are input/output gdx file names, the log file name, all other file names which are needed for diverse visualization and output options. It also includes the name of the SR-Option-File.

Option	Description	Required
`log_file`	specify a log file name	yes
`input_gdx`	specify the gdx input file for Scenred	yes
`output_gdx`	specify the gdx output file of Scenred	yes
`sr_option`	specify a SR-Option-File	no
`visual_init`	specify a name for visualization of input tree	no
`visual_red`	specify a name for visualization of reduced/constructed tree	no
`plot_scen`	specify a name for visualization of scenario processes	no
`out_scen`	specify a file for scenario data output in fan format	no
`out_tree`	specify a file for scenario data output in tree format	no

Example:

Scenred2 must be called together with a command file, which contains at least all required options. The data exchange via the gdx interface and the Scenred2 call from the Gams program is of the form (be careful with the meanings and right order of gdx symbols):

```
execute_unload 'srin.gdx', ScenRedParms, n, ancestor, prob, random;
execute 'scenred2 scenred.cmd';
execute_load 'srout.gdx', ScenRedReport, ancestor=red_ancestor, prob=red_prob;
```

For example, the command file could be the following (note the compatible gdx file names):

```
* scenred command file 'scenred.cmd'

log_file    sr.log
input_gdx   srin.gdx
output_gdx  srout.gdx
sr_option   scenred.opt
visual_red  tree
out_scen    raw.dat
```

6.39.2.2 ScenRedParms

With the symbol list of the parameter ScenRedParms and the SR-Option-File all necessary information regarding the Scenred2 run can be assigned. The Gams parameter ScenRedParms can easily included to the Gams program by the statement:

```
$libInclude scenred2
```

Of course, the include must be stated before calling Scenred2. After that statement all supported parameters can be assigned, but at least all required parameters regarding the input scenarios. By the symbols of the parameter ScenRedParms you make also the decision of what features you exactly want to use with Scenred2. Moreover, some other usefull parameters for the Scenred2 run are included in the symbol list of the parameter ScenRedParms.

Table: Supported Scenred2 parameters in ScenRedParms

Symbol	Description	Required
num_time_steps	path length from root to leaf	yes
num_leaves	leaves/scenarios in the initial tree	yes
num_nodes	nodes in the initial tree	yes
num_random	random variables assigned to a scenario or node	yes
red_num_leaves	desired number of preserved scenarios or leaves	no
red_percentage	desired relative distance (accuracy)	no
reduction_method	desired reduction method	no
construction_method	desired tree construction method	no
num_stages	number stages	no
run_time_limit	time limit in seconds	no
report_level	report level: more messages by higher values	no
scen_red	scenario reduction command	no
tree_con	tree construction command	no
visual_init	visualization initial tree	no
visual_red	visualization reduced (constructed) tree	no
plot_scen	visualization scenario processes	no
out_scen	output of scenario raw data	no

To enable some options assign a value to the parameter. A parameter value of zero (default) disables an option. Note that when running Scenred2 either scenario reduction or scenario tree construction can be performed. Hence, only `scen_red` or `tree_con` should be used at once.

Example:

The following statements describe a possible example setup for proceeding the scenario tree construction with visualization of the scenario tree and output of the scenarios to a raw data file afterwards. Note that for the visualization and the scenario output the name of output files must be specified in the SR-Command-File. Otherwise a warning will inform you about not selected file names.

```
* general parameters
ScenRedParms('num_leaves') = 100;
ScenRedParms('num_nodes') = 200;
ScenRedParms('num_time_steps') = 5;
ScenRedParms('num_random') = 2;
ScenRedParms('report_level') = 2;
ScenRedParms('run_time_limit') = 30;

* execution commands
ScenRedParms('tree_con') = 1;
ScenRedParms('visual_red') = 1;
ScenRedParms('out_scen') = 1;
```

Scenred2 can also be used for plotting tasks only. Disable both the `scen_red` and `tree_con` option and use one ore more visualization options only (see also Section Visualization for more details regarding visualizations).

6.39.2.3 SR-Option-File

The SR-Option-File is the more specific option file and will be passed to Scenred2 by the`sr_option` statement specified in the SR-Command-File. It serves as control unit for available methods provided by Scenred2. The supported options depend on what kind of method is called with Scenred2. A detailed list of all options together with examples are given below for both the scenario reduction and the scenario construction devices (see Sections Scenario Reduction and Scenario Tree Construction, respectively). Note that certain parameters can be assigned by using both ScenRedParms and the SR-Option-File. In case of having parameters defined twice a warning by Scenred2 will generated to inform you.

6.39.3 Scenario Reduction

The scenario reduction device consists of approved methods for reducing the model size by reducing the number of scenarios in an optimal way. Here it doesn't make any difference whether the input data is structured as scenario tree or not. But note, the classical scenario reduction approach is actually developed for two-stage models. Extensions for the multistage case are planed in the near future. To learn more about the mathematical theory see, for example,

- H. Heitsch, W. Römisch, and C. Strugarek. Stability of multistage stochastic programs. SIAM Journal on Optimization, 17, 2006.
- H. Heitsch and W. Römisch. A note on scenario reduction for two-stage stochastic programs. Operations Research Letters, 35:731-738, 2007.
- H. Heitsch and W. Römisch. Scenario tree reduction for multistage stochastic programs. Computational Management Science, 6:117-133, 2009.

With Scenred2 the most popular and accurate reduction algorithms of forward and backward type are maintained further on. New options make it possible to proceed with the scenario reduction more individual. The most important new parameter is given by the option `metric_type` which allows to control the reduction process by different type of probability distances. Altogether three distances can be selected (see Table below). All probability distances are associated with a special order specification which can be set by the new option `order`. Both options replace the old option `where_random` which is not used any longer.

Table: SR Options - Scenario Reduction

Option	Description
`red_num_leaves`	desired number of scenarios (integer)
`red_percentage`	relative accuracy (number from 0.0 to 1.0)
`reduction_method`	1 - Forward, 2 - Backward, 0 - Default
`metric_type`	1 - Transport (default), 2 - Fortet-Mourier, 3 - Wasserstein
`p_norm`	choice of norm (example: 0 - max, 1 - sum, 2 - Euclidian)
`scaling`	0 - scaling off, 1 - scaling on (default)
`order`	metric order (integer, default is 1)

Example:

For example, a valid SR-Option-File is the following:

```
* scenred option file

reduction_method  1
red_percentage    0.3
metric_type       2
order             2
p_norm            1
scaling           0
```

Lines starting with the star symbol (route symbol can be used too) provide comment lines. The star symbol can also be used to out comment and disable certain options.

6.39.4 Scenario Tree Construction

Scenario tree construction is the outstanding all new device of Scenred2. It allows to construct scenario trees as accurate input for multistage stochastic programs (cf. H. Heitsch and W. Römisch, Scenario tree modeling for multistage stochastic programs, Mathematical Programming, 118:371–406, 2009). The input are individual scenarios in form of a fan which must be allocated before calling Scenred2. A lot of options are offered to control the tree construction process. Note that in some cases due to sensibility of certain parameters some tuning is indispensable for producing good results.

Table: SR Options (basic) - Scenario Tree Construction

Option	Description
`construction_method`	1 - forward, 2 - backward
`reduction_method`	1 - forward, 2 - backward, used within the iteration
`first_branch`	time period of first branch (integer)
`red_percentage`	relative accuracy (level from 0.0 to 1.0)
`eps_growth`	1 - linear, 2 - exponential
`eps_evolution`	tree structure parameter (from 0.0 to 1.0)
`scaling`	0 - scaling off, 1 - scaling on (default)
`order`	order of metric

The Table above displays the main options to control the tree construction process. They are very similar to the reduction options. The role of the option `red_percentage` is here to prescribe a total epsilon accuracy (level) for the approximation scheme. But the approximation scheme is based on stagewise approximations which requires a splitting of the total level to the stages. Two strategies are offered by Scenred2 a linear and an exponential mapping of the total level to the intermediate levels. Use option `eps_growth` to select one of them. Both strategies allow a second tuning parameter `eps_evolution` which effects the slope of the epsilon splitting.

Even though this kind of control may generate good results for many applications, sometimes a more individual control can be needed. For example, some applications require a localization of branching stages. Moreover, to setup approximation bounds directly to stages can be very useful. To this end the standard options are extended by a new section environment.

6.39.4.1 Additional options - The section environment

An alternative control for the accurate constructions is provided by using the section environment. The section environment aims to establish a better monitoring of the construction process. There are overall three section types supported by Scenred2 with the same syntax.

Branching control:

This section environment allows to specify branching points, i.e., an explicit selection of stages serving for branching. For example, use

```
section branching
  2
  4
  6
end
```

to allow branching only at time period 2, 4, and 6. Note that each stage statement must be placed in one line. But stages can be merged. A shorter formulation of the same contents can be written in closed form

```
section branching
  2*6  2
end
```

This statement reads branching within time periods from period 2 to period 6 with increment 2 steps. Both assignments can be combined and should be used together with the `red_percentage` option.

Epsilon control:

In the similar way by the epsilon section it is possible to assign epsilon tolerances for the stage approximations explicitly. This environment overcomes difficulties at times coming across with the automatic epsilon control. Note that this environment disables the option `red_percentage`. For example, use

```
section epsilon
  2    0.04
  3*4  0.03
  5    0.02
  6    0.01
end
```

to control the approximation scheme by assigning different epsilon values per stage. Note that the value 0.03 is assigned to time period 3 and 4 in the example.

Node control:

The node control is the most specific control you have over the tree construction. With this environment the number of nodes of the tree which will generated can be assigned for each time stage explicitly. For example, use

```
section num_nodes
  1    1
  2*3  5
  4*5  10
  6    15
end
```

The syntax is the same as before. Note that only one section environment can be use at once. In particular, only the first section environment detected in the option file is used. The section environment can be out commented like a standard option too.

Experimental option:

There is one other useful option to speed up computations when building different scenario trees from exactly the same input data. In this case the scenario distances needed to compute the trees could be saved to a external file at the first run and reloaded at later runs. Hence, the distances must be calculated only once. For example, use the option

```
write_distance  dist.sr2
```

to save the computed distances to the file 'dist.sr2'. To reload them at next run use the option

```
read_distance  dist.sr2
```

The option is classified as experimental since no validation of the input file takes place. Before using this option, please ensure that the distances loaded with the `read_distance` option are the right ones.

Example:

Finally, look at the following example to see a valid SR-Option-File which can be passed to the scenario tree construction:

```
* tree construction option file

construction_method  2
reduction_method     1
order                1
scaling              0

section epsilon
  2*4    0.1
  5      0.2
  6      0.1
end
```

6.39.4.2 Example problems 'srtree.gms' and 'srpchase.gms'

Small example problems has been included to the GAMS Model Library.

The implementation can be found in the Gams programs 'srtree.gms' and 'srpchase.gms'. It might help you to practice in building scenario trees using Gams/Scenred2. The problem 'srtree.gms' converts a fan format scenario representation into a tree format representation and then reduces the tree size. The problem 'srpchase.gms' is to solve a simple stochastic purchase problem involving three stages. Sample scenarios which are generated from a fixed distribution using a random value generator serve as input for the tree construction.

6.39.5 Visualization

Visualization is another all new feature of Scenred2. In this section an easy way for making plots of scenario processes and tree structures is described. To this end you need the free Gnuplot software or any other plotting software which allows plotting directly from simple data files.

The concept of plotting tasks is the following. For each plot two files are generated, a Gnuplot access file (name.plt) and a raw data file (name.dat). The access file contains basic Gnuplot options and it can be adjusted for individual liking afterwards. The default output is the display. The supported plotting commands are

```
visual_init, visual_red, plot_scen
```

for plotting the initial tree structure, the reduced/constructed tree structure, and the scenario process(es), respectively.

Example:

For example, to visualize the constructed tree use the option

```
visual_red tree
```

within the SR-Command-File to specify the name for the output and activate the ScenRedParms parameter

```
ScenRedParms('visual_red') = 1;
```

in the Gams program. The result are the output files 'tree.plt' and 'tree.dat'. To compute the picture now you simply open the file 'tree.plt' with Gnuplot from the directory, where both output files are located (that should be the working directory). Alternatively, from the command line prompt call

```
>gnuplot tree.plt
```

Gnuplot will automatically generate the picture. Feel free to change any option in the Gnuplot access file for individual requirements. See also the Gnuplot manual for more details. In particular, to compute a well-scaled encapsulated postscript picture (eps), you simply have to uncomment a few lines in the Gnuplot option file above and to open it with Gnuplot once again.

With the command `plot_scen` the scenario process(es) can be visualized. Note that Scenred2 generates Gnuplot access and data files according to the number of random values.

6.39.6 Command Line Interface

The command line interface allows to run Scenred2 stand alone without using Gams. In this case the input for scenario reduction and scenario tree construction is handled by special input data files. The command file will be extended by the parameters having with the ScenRedParms otherwise.

To execute Scenred2 from the command line prompt together with a specified command file (which is required again), for example, call

```
>scenred2 command.file -nogams
```

To avoid diverse error messages do not forget the '-nogams' option to switch off the Gams interface. The command file can include some of the following options.

```
report_level  <integer>
runtime_limit <integer>
read_scen     <input file>
scen_red      <option file>
tree_con      <option file>
visual_init   <name>
visual_red    <name>
plot_scen     <name>
out_scen      <file name>
out_tree      <file name>
```

The denotation is not accidental the same as in case of using the Gams interface. The meaning of a certain option is maintained for the command line interface. To compute any scenario reduction or scenario tree construction the same SR-Option-Files are supported. It remains to clarify the data input format which comes across with the new `read_scen` command.

6.39.6.1 Data input format

To feed Scenred2 with data the scenario parameters must be passed by the `read_scen` command. Two types of input file formats are accepted.

a) The tree format:

This file is a formated data file including all information of the input scenarios tree. It must have a header with dimension information and the scenario data separated for each node. The header includes the type declaration, the number of nodes, and the number of random values.

The data part starts with the key word `DATA` (do not forget). The tree data has to be ordered node by node. For every node the following information is expected separated by white spaces: The unique predecessor node (root node points to itself) followed by the node probability and followed by the assigned number of random data values. All information to one node should be written to one line (only for clearness reasons). Comment lines are allowed.

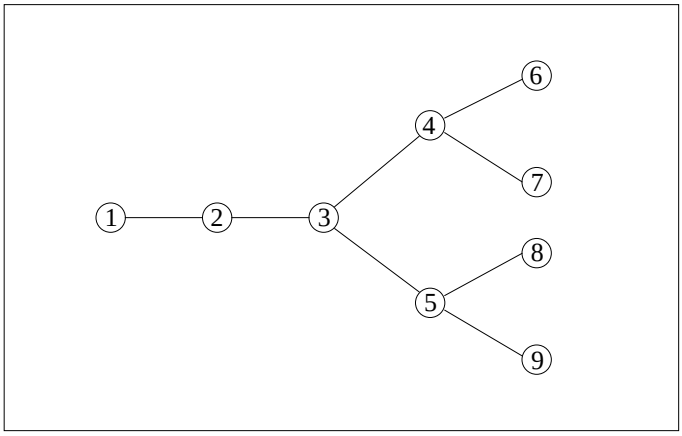
Match the following conventions:

- Nodes are identified by a sequence of integer numbers.
- The root node is expected to be the node '1'.
- The predecessor of root is '1' too, i.e., the root points to itself.

- All nodes numbers require a canonical order by stages and scenarios (see example).

Example:

```
# input tree example for scenred

TYPE TREE

NODES   9
RANDOM  4

DATA
* PRED PROB  RAND-1 RAND-2 RAND-3 RAND-4
   1    1.0    42.5    9.1    7.5  120.0
   1    1.0    39.8   11.2    8.4   90.0
   2    1.0    37.6   14.0    6.3  110.0
   3    0.5    38.9   12.4    8.1  130.0
   3    0.5    35.7   13.8    7.5  120.0
   4   0.25    40.3   14.9    7.2  120.0
   4   0.25    38.4   15.2    8.9  100.0
   5    0.3    37.6   14.9    9.3   80.0
   5    0.2    36.3   12.8   10.3   90.0
END
```



b) The fan format:

A scenario fan serves as input for the scenario tree construction but it can be used also for the scenario reduction. The scenario fan represents a special form of a scenario tree, where we consider individual scenarios merged to a collective root node (the root node can also be viewed here as some kind of artificial node).

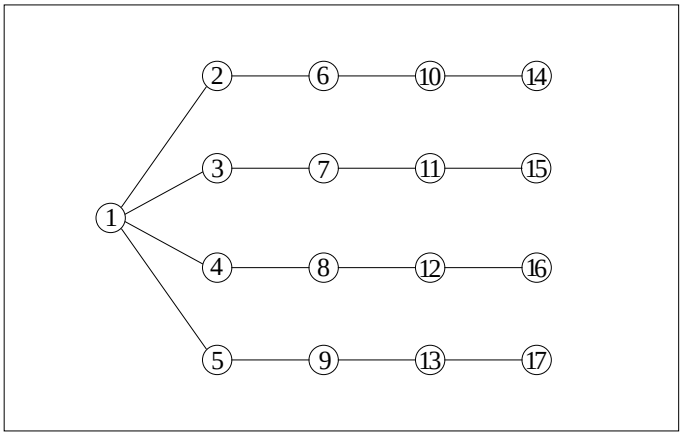
Accordingly, the fan input file is a formated data file including all information of the scenarios in individual form now. It must have a similar header with dimension information and the scenario data separated now for each scenario. The header gets the type declaration `FAN` instead of `TREE` and includes the number of scenarios, the number of time periods, and the number of random values. The data part is opened again with the `DATA` key word.

Every scenario is specified by a dataset including the scenario probability first followed by the different random values in ascending order w.r.t. time periods. All entries must be separated by a white space. Comment lines can be placed by the star and route symbols again. Note that in case of having an undetermined root node the mean of random values will taken for the first time period to appoint a unique root node. The example tree represented as input in scenario fan format is displayed in the next example.

Example:

```
# input fan example for scenred

TYPE  FAN

TIME    5
SCEN    4
RANDOM  4

DATA
0.2500
42.5      9.1      7.5     120.0
39.8     11.2      8.4      90.0
37.6     14.0      6.3     110.0
38.9     12.4      8.1     130.0
40.3     14.9      7.2     120.0

0.2500
42.5      9.1      7.5     120.0
39.8     11.2      8.4      90.0
37.6     14.0      6.3     110.0
38.9     12.4      8.1     130.0
38.4     15.2      8.9     100.0

0.3000
42.5      9.1      7.5     120.0
39.8     11.2      8.4      90.0
37.6     14.0      6.3     110.0
35.7     13.8      7.5     120.0
37.6     14.9      9.3      80.0

0.2000
42.5      9.1      7.5     120.0
39.8     11.2      8.4      90.0
37.6     14.0      6.3     110.0
35.7     13.8      7.5     120.0
36.3     12.8     10.3      90.0
END
```

Note that even though all scenarios coincide at the first three time periods, in this example, the scenarios will represented by one node each for every time period by the fan input format. The exception is the first time period, where a unique root node is expected in general and, therefore, only one node is assigned. The following picture shows the structure of the scenario fan which is generated by the example input.



6.39.6.2 Data Export

Scenred2 allows to export scenario data after computing the scenario reduction or scenario tree construction to external data files. Data export is available for both the Gams and the command line interface. To export data from Scenred2 two output options `out_tree` and `out_scen` can be use. These options generate data files according to the tree and fan format, respectively. The name of the data files will be specified in the SR-Command-File. When using the Gams interface the options must be connected by activating the corresponding ScenRedParms parameter, additionally.

6.39.7 A Simplified Interface to Scenred2: $libinclude runscenred2

While the previously described interface between GAMS and Scenred2 provides a maximum of flexibility, it also is rather complex and error-prone. The GAMS utility `runscenred2` tries to hide most of the mechanics of the GAMS/Scenred2 interface. The call to `runscenred2` looks as follows:

```
$libInclude runscenred2 myprefix tree_con n tree p rtree rp rv1 rv2
```

Table: runscenred2 Arguments:

	Argument	Description
1	`myprefix`	base name for files used with Scenred2
2	`tree_con` or `scen_red`	select Scenred2 action: tree construction or scenario reduction
3	`n`	the set of nodes in the tree
4	`tree`	the set of ancestor relations describing the tree
5	`p`	the parameter containing the node probabilities
6	`rtree`	the set of ancestor relations of the reduced tree (output)
7	`rp`	the parameter containing the node probabilities for the reduced tree (output)
8	`rv1`, `rv2`, ...	parameters containing random values of the nodes

The table above describes the arguments of the `runscenred2` call. Arguments 3, 4, 5, 8 and following correspond to the symbols that need to be exported to the Scenred2 data input file (done with the `execute_unload` call in the complex interface). The output arguments 6 and 7 correspond to the symbols imported from te Scenred2 data output file (done with the `execute_load` call in the complex interface). The parameters `ScenRedParms` and `ScenRedReport` are invisibly communicated with Scenred2.

The second argument instructs Scenred2 either to construct a tree (`tree_con`) or to reduce a tree (`scen_red`).

Instead of providing an explicit name for all the different files in the Scenred2 command file, the first argument determines the name of all files using a simple naming scheme. The following name scheme is observed:

Filename	Command option	Description
`sr2`*myprefix*`.log`	`log_file`	log file name
`sr2`*myprefix*`_in.gdx`	`input_gdx`	gdx input file name
`sr2`*myprefix*`_out.gdx`	`output_gdx`	gdx output file name
`sr2`*myprefix*`.opt`	`sr_option`	option file name
`sr2`*myprefix*`_vi.plt`	`visual_init`	file name for visualization of input tree

Filename	Command option	Description
sr2*myprefix*_vr.plt	visual_red	file name for visualization of reduced/constructed tree
sr2*myprefix*_plot.plt	plot_scen	file name for visualization of scenario process
sr2*myprefix*_raw.dat	out_scen	file name for scenario data output in fan format
sr2*myprefix*_tree.dat	out_tree	file name for scenario data output in tree format

The first three files (log_file, input_gdx and output_gdx) are always used. The only optional input file sr_option is read by Scenred2 if ScenRedParms('sroption')=1. When you create this file, make sure to use the proper file name. The output files are created by Scenred2 if the corresponding option is set to 1 in ScenRedParms, e.g. ScenRedParms('out_tree')=1.

In addition to a simpler communication of data between GAMS and Scenred2, the newer versions of GAMS/Scenred2 (starting with GAMS distribution 23.1) release the user of setting required fields in the ScenRedParms parameter: num_time_steps, num_leaves, num_nodes, and num_random. GAMS/Scenred2 calculates these numbers from its input data. In case the user still sets these fields, Scenred2 will ensure that the internally calculated numbers and the user provided numbers match.

6.40 ShellExecute

Note

This tool is part of the GAMS Tools Library. Please inspect the general information about GAMS Tools.

This allows you to spawn an external program based on the file type of the document to open.

6.40.1 Usage

Command line:

```
gamstool [win32.]ShellExecute filename [arguments] [dir=workdir] [verb=open|edit|find|print|...] [sh
```

Compile time:

```
$callTool [win32.]ShellExecute filename [arguments] [dir=workdir] [verb=open|edit|find|print|...] [s
```

Execution time:

```
executeTool '[win32.]ShellExecute filename [arguments] [dir=workdir] [verb=open|edit|find|print|...]
```

Where

Argument	Description
filename	Name of the file to open.
arguments	Additional arguments, if needed.

Argument	Description
dir=workdir	This specifies the directory where the file to be opened is located. Default: current directory.
verb=open or ...	This specifies the action to be performed (default open). The allowed actions are application dependent. Some commonly available verbs include: • edit: Launches an editor and opens the document for editing. • find: Initiates a search starting from the specified directory. • open: Launches an application. If this file is not an executable file, its associated application is launched. • print: Prints the document file. • properties: Displays the object's properties.
-showcmd	The optional argument showcmd specifies how an application is to be displayed when it is opened. The map between numerical values 0 to 11 and symbolic names can be found here: https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-showwi

6.40.2 Example: Opening MS Access database

This example uses the [**trnsport**] model from the model library. After solving the model, a couple of symbols is exported to a Microsoft Access database using GAMS Connect. MS Access is then launched to inspect the results. This is a small example that should run very quickly.

```
$call.checkErrorLevel gamslib -q trnsport
$call.checkErrorLevel gams trnsport gdx=trnsport.gdx
$onEmbeddedCode Connect:
- GDXReader:
    file: trnsport.gdx
    readAll: True
- SQLWriter:
    connection: {'DRIVER':'Microsoft Access Driver (*.mdb, *.accdb)', 'DBQ': '%system.fp%trnsport.ac
    connectionType: access
    symbols:
      - name: a
        tableName: capacity
        ifExists: replace
      - name: b
        tableName: demand
        ifExists: replace
      - name: d
```

```
        tableName: distance
        ifExists: replace
$offEmbeddedCode
$callTool win32.ShellExecute trnsport.accdb
```

The command `$callTool win32.ShellExecute trnsport.accdb` will launch Access to view the ACCDB file.

6.41 SQL2GMS

Author

Erwin Kalvelagen

Attention

`SQL2GMS` is deprecated (see GAMS 40 SQL2GMS release notes). Please use Connect agent SQLReader instead.

6.41.1 Overview

`SQL2GMS` is a tool to convert data from an SQL database into GAMS readable format. The source is any data source accessible through Microsoft's Data Access components including ADO, ODBC and OLEDB. The target is a GAMS Include File (.inc) or a GAMS GDX File (.gdx).

When running the executable sql2gms.exe without command line arguments, the tool will run interactively with a built-in GUI interface. Alternatively `SQL2GMS` can be run in batch mode, which is useful when running it directly from a GAMS model without user intervention using the `$call` command at compile time or the `execute` command at execution time.

Database tables can be considered as a generalization of a GAMS parameter. GAMS parameters have multiple index columns but just one value column. If the table is organized as multi-valued table, a UNION operation in the SQL statement can be used to generate the correct GAMS file.

There are no special requirements on the data types used in the database. The data is converted to strings, which is almost always possible. Data types like LONG BINARY may not be convertible to a string, in which case an exception will be raised. In general NULL's should not be allowed to get into a GAMS data structure. The handling of NULL's can be specified in an option.

Besides parameters it is also possible to generate set data.

If you need to read data from an MS Access database or MS Excel spreadsheet, the accompanying tools MDB2GMS or XLS2GMS resp. GDXXRW may be more appropriate. In some cases it may be more useful to use `SQL2GMS` to read from Access and Excel data sources.

6.41.2 Requirements

SQL2GMS runs only on Windows PC's with [MDAC] (https://en.wikipedia.org/wiki/Microsoft_Data_Access_Compon
(Microsoft Data Objects) installed. In many cases this will be installed on your computer. Otherwise it can
be downloaded from the [Microsoft Download Center] (https://www.microsoft.com/en-in/download/details.aspx?i
(direct link to MDAC).

In order to use SQL2GMS effectively, you will need to have a working knowledge of [SQL] (https://en.wikipedia.org/wiki/SQL) in order to formulate proper database queries. In addition you will need some knowledge on how to connect to your database using ODBC or ADO.

6.41.3 Batch Usage

SQL2GMS can be run in batch mode without user intervention from within the GAMS model by using the $call resp. execute statements or directly from command prompt while specifying all arguments in the command-line. A SQL2GMS batch call is of the following form:

```
sql2gms connectionString outputFile queryString
```

A proper call will at least contain the following three command-line arguments:

1. The name of the database and the instruction for connecting to the database must be specified within the connectionString, which is indicated by the option C. i.e. C=connectionString.
2. The name of outputFile, either an include file (.inc) or GDX file (.gdx), must be specified. Using an include file to store the results of the query is indicated by the option O, i.e. O=outputFile.inc, while the use of a GDX file is indicated by the option X, i.e. X=outputFile.gdx.
3. The SQL queryString, containing the SQL statement to be executed on the database, must be specified within the option Q, i.e. Q=queryString.

See also Command-line Arguments below for a complete list of all possible command-line arguments. Consider that the $call or execute usage is rather error prone and you will need to spend a little bit of time to get the call correct and reliable. Alternatively, use the interactive built-in GUI interface or enter the command-line arguments in an external text file in order to write a more structured and readable command. The use of an external parameter file is indicated by preceding the file name with a @ (At sign).

Also consider to take a look at the section Strategies, mentioning some of the drawbacks of the batch usage and how to overcome them.

6.41.3.1 Command-line Arguments

The following table summarizes the command-line arguments that can be specified when using SQL2GMS directly from the GAMS model or command prompt.

Argument	Interpretation	Default	Description
C	connectionString	none	Specify the connectionString (required).
O	outputIncludeFile	none	Specify the name of the output file (.inc). Either `O=` or `X=` must be specified (or both).
On	nth outputIncludeFile	none	Match the nth query with the nth output file (.inc format) if multiple queries are used.
X	outputGDXFile	none	Specify the name of the output file (.gdx). Either `O=` or `X=` must be specified (or both).
Q	Query	none	This option can be used to specify a SQL query (required).
Qn	nth query	none	Match the nth query with the nth output file (.inc) format or with the nth set- or parameter name when writing to GDX if multiple queries are used.
S	setName	none	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file.
Sn	nth setName	none	Match the nth query with the nth set in the GDX file if multiple queries are used.
Y	setName (with expl. text)	none	If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. Use this argument to store a set with explanatory text.
Yn	nth setName (with expl. text)	none	Match the nth query with the nth set (with explanatory text) in the GDX file if multiple queries are used.
A	parameterName	none	If we write to a GDX file, use this option to specify the name of a parameter to be used inside the GDX file.
An	nth parameterName	none	Match the nth query with the nth parameter in the GDX file if multiple queries are used.
L	Listing	disabled	Controls if the data is embedded in the listing file.
M	Mute	disabled	Controls if additional information is written to the log and include file.
B	Quote Blanks	disabled	Quote strings if they contain blanks or embedded quotes.
@fileName	ext. options file	none	Causes the program to read options from an external text file.
N	iniFileName	sql2gms.ini	Indicates the usage of a different INI file.
T1	connectionTimeOut	15	Indicates how long to wait while establishing a connection before terminating the attempt and generating an error.
T2	commandTimeOut	30	Indicates how long to wait while executing a command before terminating the attempt and generating an error.
T	timeOut	none	Sets both `T1` and `T2`.
D	Debug	disabled	Generate debug information.

Argument	Interpretation	Default	Description
E	Empty sets	disabled	Allow an empty result set; without this option an empty result set will generate an error.
R	rowBatchSize	100	Row batch size; the default is 100 records.
P	Password	none	Specify a password for the database.
W	Wiring	none	Maps database columns to GAMS index positions.

Some more detailed remarks on the command-line arguments:

C = *string* (connectionString, default = none)

This option is required and specifies which database is to be used and how `SQl2GMS` talks to the database. A collection of valid connection strings can be found in section Connection Strings. Often the connection string will need to be surrounded by quotes, as in: `C="DSN=sample"`. This option is demonstrated in all examples, see Example 1 - Reading a single valued Table for instance.

O = *string* (outputIncludeFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter or set statement. Make sure the directory is writable. UNC names can be used. An output file must be specified for batch operation: i.e. either `O=` or `X=` needs to be specified (or both). The include file will be an ASCII file that can be read by GAMS using the $include command within the data definition of a set, parameter or scalar. If the include file already exists, it will be overwritten. This option is demonstrated in Example 1 - Reading a single valued Table for instance.

On = *string* (outputIncludeFile, default = none)

When using multiple queries in a single `SQL2GMS` call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition:

```
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

X = *string* (outputGDXFile, default = none)

This option specifies the name of the output file. The format of the output file will be a GAMS GDX file. Make sure the directory is writable. UNC names can be used. If the GDX file already exists it will be overwritten - it is not possible to append to a GDX file. An output file must be specified for batch operation: i.e. either `O=` or `X=` needs to be specified (or both). This option is demonstrated in Example 5 - Reading Set with Explanatory Text or Example 7 - Multi-Query Batch Example for instance.

Q = *string* (query, default = none)

This option can be used to specify an SQL query. Queries can contain spaces and thus have to be surrounded by double quotes. For the exact syntax of the queries that is accepted by the database we refer to the documentation that comes with your RDBMS (Relational DataBase Management System). The query is passed on directly to the RDBMS so the complete power and expressiveness of SQL is available including stored procedures etc. For an exact description of allowed expressions consult a text on your database system. This option is demonstrated in Example 1 - Reading a single valued Table for instance.

Qn = *string* (query, default = none)

When using multiple queries in a single `SQL2GMS` call, you can append a number to match a query with an output file, as an include file storing the results for multiples queries cannot be interpreted later on in your GAMS model when using the include file in a set or parameter definition. In addition, you can match the results of a query with a specific set- or parameter name when writing to GDX.

```
Q1="SELECT a, b FROM table"
O1=ab.inc
Q2="SELECT c, d FROM table"
O2=cd.inc
```

or (GDX output file format - where several sets and parameters can be stored in a single file):

```
Q1="SELECT a, b FROM table"
A1=abParameter
Q2="SELECT c FROM table"
S2=cSet
```

Note the usage of the arguments An resp. Sn in order to store the results as parameter resp. set and to specify the name of the symbols. See also section Multi-Query Batch Usage or Example 7 - Multi-Query Batch Example for instance.

S = *string* (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be stored in the GDX file (containing the results of the query). This option is demonstrated in Example 4 - Reading a multi dimensional Set.

Sn = *string* (setName, default = none)

If multiple queries are used in a single `SQL2GMS` call while writing to a GDX file, use this option to specify the name of the nth set to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i FROM table"
S1=iSet
Q2="SELECT j FROM table"
S2=jSet
```

See section Multi-Query Batch Usage and Example 7 - Multi-Query Batch Example for instance.

Y = *string* (setName, default = none)

If we write to a GDX file, use this option to specify the name of a set to be used inside the GDX file. The last column specified within the select clause in the SQL statement will be used as explanatory text. This option is demonstrated in Example 5 - Reading Sets with Explanatory Text for instance.

Yn = *string* (setName, default = none)

If multiple queries are used in a single `SQL2GMS` call while writing to a GDX file, use this option to specify the name of the nth set (with explanatory text) to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i, explTextForSeti FROM table"
Y1=iSet
Q2="SELECT j, explTextForSetj FROM table"
Y2=jSet
```

The last column specified within the select clause in the SQL statements will be used as explanatory text.

A = *string* (parameterName, default = none)

If we write to a GDX file, use this option to specify the name of a parameter to be stored the GDX file (containing the results of the query).
Note: `MDB2GMS` uses `P`, but `P` was already taken in `SQL2GMS` for specifying the password.

An = *string* (parameterName, default = none)

If multiple queries are used in a single `SQL2GMS` call while writing to a GDX file, use this option to specify the name of the nth parameter to be stored in the GDX file (containing the results of the nth query), e.g.

```
Q1="SELECT i, j, value FROM table"
A1=ijValue
Q2="SELECT n, m, value FROM table"
A2=nmValue
```

See section Multi-Query Batch Usage and Example 7 - Multi-Query Batch Example for instance. for instance.

L (Listing, disabled by default)

Embed the data between the `$offListing` and `$onListing` dollar control options, so the data will not be listed in the listing file. This is a quick way to reduce the size of the listing file when including very large data files into the model. Otherwise the listing file would become too large to be handled comfortably.

M (Mute, disabled by default)

Controls if additional information (GAMS and `SQL2GMS` version numbers, number of rows in the data, elapsed time, used query etc.) is written to the log and include file.

B (Quote Blanks, disabled by default)

Quote strings if they contain blanks or embedded quotes. If a string does not contain a blank or embedded quotes, it will remain unquoted. If the string contains a single quote the quotes applied will be double quotes and if the string contains already a double quote, the surrounding quotes will be single quotes. In the special case that the string contains both, the double quotes are replaced by single quotes. For more information see subsection Quotes. This option only applies to an output include file.

@fileName = *string* (fileName, default = none)

Causes the program to read options from an external text file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if they were specified directly on the command-line. See also Parameter Files for some further details.

N = *string* (fileName, default = none)

Use a different INI file than the standard sql2gms.ini located in the same directory as the executable sql2gms.exe.

T1 = *integer* (connectionTimeOut, default = 15)

Indicates how long to wait while establishing a connection before terminating the attempt and generating an error. The value sets, in seconds, how long to wait for the connection to open. Default is 15. If you set the property to zero, ADO will wait indefinitely until the connection is opened. A value of -1 indicates that the default value is to be used. Note: the provider needs to support this functionality.

T2 = *integer* (commandTimeOut, default = 30)

Indicates how long to wait while executing a command before terminating the attempt and generating an error. The value sets, in seconds, how long to wait for a command to execute. Default is 30. If you set the property to zero, ADO will wait indefinitely until the execution is complete. A value of -1 indicates that the default value is to be used. Note: the provider needs to support this functionality.

T = *integer* (timeOut, default = none)

Sets both T1 and T2.

D (Debug, disabled by default)

Generate debug information. This option must be specified in an INI file when using the interactive mode of `SQL2GMS`.

E (Empty sets, disabled by default)

Allow an empty result set; without this option an empty result set will generate an error. This option must be specified in an INI file when using the interactive mode of `SQL2GMS`.

R = *integer* (rowBatchSize, default = 100)

Row batch size; the default is 100 records. This option must be specified in an INI file when using the interactive mode of `SQL2GMS`.

P = *string* (password, default = none)

Specify a password for the database.

W = *string* (wiring, default = none)

By using the `W` option, one can map database columns to GAMS index positions. See model **[Wiring]** for reference (note that `MDB2GMS` is used to access the MS Access database instead of `SQL2GMS`, however this does not affect the `wiring` specification).

6.41.3.2 Example 1 - Reading a single valued Table

Suppose we want to read the distances parameter of the [**trnsport**] model from the GAMS Model Library. The data is stored in the Microsoft Access Database format (file Sample.accdb).

Distances

City1	City2	Distance
SEATTLE	CHICAGO	1,7
SAN-DIEGO	CHICAGO	1,8
SEATTLE	NEW-YORK	2,5
SAN-DIEGO	NEW-YORK	2,5
SEATTLE	TOPEKA	1,8
SAN-DIEGO	TOPEKA	1,4

Figure 6.64 Table: distances

The data can be queried with a simple SQL statement:

```
SELECT city1, city2, distance
FROM   distances
```

By running the following `SQL2GMS` statement, the connection to the database Sample.accdb is established. In addition, the data will be queried and the results are written to a GAMS include file afterwards (.inc).

```
sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT city1, city2, distar
```

The connection string is specified using the argument C. Note that the string is enclosed by quotes, as the string contains blanks. The database file name Sample.accdb is also specified within the connection string. For more information on how to connect to your database, see section Connection Strings. The arguments Q and O are used to specify the query and the output file name (and format).

The generated include file distances.inc looks like:

```
* ------------------------------------------------------
* SQL2GMS 3.0      25.2.0 r67636 ALFA Released 15Aug18 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* ------------------------------------------------------
* ADO version: 10.0
* Connection string: Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb
* Provider: MSDASQL
* Query: SELECT city1, city2, distance FROM distances
* ------------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* ------------------------------------------------------
```

The commented header section summarizes some information about the SQl2GMS resp. GAMS version and about the executed database query. The standard export format is to consider the last column as the value column (containing the distances) and the previous columns as the indices (containing the city names). The indices are separated by a dot, allowing the generated include file to be used as part of a parameter declaration statement in your GAMS model.

Retrieving the data using SQL2GMS from the database and including the queried data in your GAMS model within the parameter declaration statement (at compile time) can be combined in the following way:

```
Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT city1, city2,
Parameter d(i,j) 'distance in thousands of miles' /
$include distances.inc
/;

display d;
```

Finally, the values of the parameter d are displayed:

```
             new-york      chicago       topeka

seattle         2.500        1.700        1.800
san-diego       2.500        1.800        1.400
```

This example is also part of the GAMS Data Utilities Library, see model [**Distances2**] for reference. Note that the query results are written to a GDX file in addition.

6.41.3.3 Example 2 - Reading a multi valued Table

In this scenario, we want two read the three index columns year, loc, prod and the value columns sales and profit from the database file Sample.accdb. Therefore, we use two separate parameters and queries or alternatively a parameter with an extra index position (for sales resp. profit) and a UNION select.

Consider the table with two value columns sales and profit:

data

year	loc	prod	sales	profit
1997	la	hardware	80	8
1997	la	software	60	16
1997	nyc	hardware	110	5
1997	nyc	software	100	10
1997	sfo	hardware	80	9
1997	sfo	software	50	10
1997	was	hardware	120	7
1997	was	software	70	20
1998	la	hardware	70	6
1998	la	software	70	10

Figure 6.65 Table: data (shortened for presentation)

Two separate Parameters

A simple way to import this into GAMS is to use two parameters and two SQL queries. The SQL queries can look like:

```
SELECT year, loc, prod, sales
FROM   data
SELECT year, loc, prod, profit
FROM   data
```

We can generate a include file sales.inc by running the following command:

```
sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" Q="SELECT year, loc,
```

Note that we specify the first query in order to select the sales and the relevant index columns within the Q argument. The query results are written to sales.inc using the O argument. Analogously we generate a include file profit.inc by running the following command while specifying the second query in order to obtain the profits and the relevant index columns:

```
sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" Q="SELECT year, loc,
```

Retrieving the data using `SQL2GMS` from the database Sample.accdb and including the queried data in your GAMS model within the parameter declaration statements of `sales` and `profit` (at compile time) can be combined in the following way:

```
Set
   year 'years'     / 1997*1998 /
   loc  'locations' / nyc, was, la, sfo  /
   prd  'products'  / hardware, software /;

$call sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" Q="SELECT year,
Parameter sales(year,loc,prd) /
$include sales.inc
/;

$call sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" Q="SELECT year,
Parameter profit(year,loc,prd) /
$include profit.inc
/;
```

Single Parameter with extra Index Position

The operation can also be performed in one big swoop by using a different GAMS datastructure, i.e. a single parameter is defined with an extra index `type` to indicate the data type (sales or profit). The index and value columns will be selected by the following SQL statement. Note the UNION operation in order to combine the results and the strings 'sales' resp. 'profit' to identify the data type later on.

```
SELECT year, loc, prod, 'sales', sales
FROM   data
UNION
SELECT year, loc, prod, 'profit', profit
FROM   data
```

The data is accessed, queried and written to data.inc by running the following command:

```
sql2gms @howToRead.txt
```

Note that usage of the external parameter file howToRead.txt shown below in order to increase the readability of the command (one argument per line, quotes can be omitted):

```
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FR
O=data.inc
```

The generated include file data.inc looks like (shortened for presentation):

```
* -----------------------------------------------------
* SQL2GMS 3.0      25.2.0 r67636 ALFA Released 15Aug18 VS8 x86 32bit/MS Windows
* Erwin Kalvelagen, GAMS Development Corp
* -----------------------------------------------------
* ADO version: 10.0
* Connection string: DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb
* Provider: MSDASQL
* Query: SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', pro
* -----------------------------------------------------
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
* -----------------------------------------------------
```

Retrieving the data using `SQL2GMS` from the database and including the queried data in your GAMS model within the parameter declaration (at compile time) can be combined in the following way (note that the parameter has a fourth index `type` in order to access the data type `sales` resp. `profit`):

```
$onEcho > howToRead.txt
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb
Q=SELECT year, loc, prod, 'sales', sales FROM data UNION SELECT year, loc, prod, 'profit', profit FR
O=data.inc
$offEcho

Set
   year 'years'     / 1997*1998 /
   loc  'locations' / nyc, was, la, sfo  /
   prd  'products'  / hardware, software /
   type 'data type' / sales, profit      /;

$call sql2gms @howToRead.txt
Parameter data(year,loc,prd,type) /
$include data.inc
/;
```

This example is also part of the GAMS Data Utilities Library, see model **[SalesProfitDB5]** for reference.

6.41.3.4 Example 3 - Reading a one dimensional Set

This example demonstrates how to read set elements of a one dimensional set from a single column of a database file. Suppose we want to read the column City1 (see table distances) in order to define the set i in the GAMS model. Make sure elements are unique by using the distinct operation within the SQL statement (otherwise there will be an error when including the file within the set definition in the GAMS model, as some set elements will be redefined):

```
SELECT distinct(City1)
FROM   distances
```

The include file city1.inc looks like (header informations are removed in order to shorten the representation):

```
* ----------------------------------------------------
SAN-DIEGO
SEATTLE
* ----------------------------------------------------
```

All steps (data access via SQL2GMS, set definition) can be combined:

```
$call sql2gms C="Provider=MMicrosoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT distinct(city
Set i 'canning plants' /
$include city_i.inc
/;

display i;
```

The display statement generates the following output in the listing file:

```
----     56 SET i

seattle  ,    san-diego
```

6.41.3.5 Example 4 - Reading a multi dimensional Set

When reading a multi dimensional set from database and writing the results to an include file by using the O argument, one has to observe that the elements in the include file must have the correct format in order to be interpreted as element of a multi dimensional set. For instance, a line containing a b c is not recognized as a proper set element of a three dimensional set. In particular, one has to add periods between the single elements, i.e. a.b.c will be interpreted correctly.

Depending on your DBMS (DataBase Management System), these periods must be added explicitly in a different way within the SQL statement. E.g. add a dummy value field by adding a quoted blank to the select clause (index1, index2, index3 and dataTable are some placeholders):

```
SELECT index1, index2, index3, " " FROM dataTable
```

or by adding the periods explicitly within the select clause (|| or & depending on DBMS):

```
SELECT index1&'.'&index2&'.'&index3 FROM dataTable
SELECT index1||'.'||index2||'.'||index3 FROM dataTable
```

For instance, suppose we want to define a two dimensional set

```
Set ij(i,j) 'canning plants - markets';
```

based on the data of the table distances stored in Sample.accdb. The following SQL2GMS statement connects you to the database, queries the columns with the city names and adds an empty value field in order to create periods between the set elements:

```
sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT city1, city2, ' ' FI
```

The include file city_ij.inc looks like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
SEATTLE.NEW-YORK ' '
SAN-DIEGO.NEW-YORK ' '
SEATTLE.CHICAGO ' '
SAN-DIEGO.CHICAGO ' '
SEATTLE.TOPEKA ' '
SAN-DIEGO.TOPEKA ' '
* -----------------------------------------------------
```

Without adding the empty value field, the resulting include file would look like (shortened):

```
* -----------------------------------------------------
SEATTLE NEW-YORK
SAN-DIEGO NEW-YORK
* -----------------------------------------------------
```

Since the periods are missing, the lines are not recognized as valid elements of a two dimensional set. All steps can be combined in the following way:

```
Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT city1, city2,

Set ij(i,j) 'two dimensional set' /
$include city_ij.inc
/;

display ij;
```

The display statement generates the following output in the listing file:

```
----     75 SET ij  two dimensional set

             new-york     chicago      topeka

SAN-DIEGO         YES         YES         YES
SEATTLE           YES         YES         YES
```

Note that there is no need to add periods explicitly when reading multi dimensional sets, if the results are written only to a GDX file by using the X and S resp. Y arguments, i.e. there is no need to modify the query:

```
SELECT index1, index2, index3 FROM datatable
```

when using SQL2GMS in the following way:

```
sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT index1, index2, inde
```

which will generate the file setData.gdx with a three dimensional set named setName containing the results of the query.

6.41.3.6 Example 5 - Reading Sets with Explanatory Text

In this example, we will demonstrate how to read set elements with explanatory text from a database file using SQL2GMS. In the first place, we are going to write the query results to an include file, afterwards we use the Y argument in order to store the query results as a set with explanatory text in a GDX file.

setData

setElement	explText
firstSetElement	Explanatory text for the first 'set element'
secondSetElement	Explanatory text for the second "set element"
thirdSetElement	Explanatory text for the third 'set element"
fourthSetElement	Explanatory text for the fourth set element

Figure 6.66 Table: setData

Note the blanks and the mixed quotes in the column containing the explanatory text. The data can be accessed by the following query:

```
SELECT setElement, explText
FROM   setData
```

Writing the Query Results in an include File

The last column in the select clause will be used as explanatory text. Take in mind to add the argument B in order to handle text strings with embedded blanks or quotes. The following GAMS code accesses the data and writes the results to an include file setData.inc:

```
$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" B Q="SELECT setElement,

Set a /
$include setData.inc
/;
```

The resulting include file will look like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
firstSetElement "Explanatory text for the first 'set element'"
secondSetElement 'Explanatory text for the second "set element"'
thirdSetElement "Explanatory text for the third 'set element'"
fourthSetElement 'Explanatory text for the fourth set element'
* -----------------------------------------------------
```

Note the handling of the quotes according to the description in B.

Writing the Query Results in a GDX File
When storing the results of the query as a set with explanatory text in a GDX file, there is no need to observe embedded blanks or quotes manually, instead one can use the Y argument. The last column specified in the select clause of the SQL statement will be interpreted as explanatory text. The following GAMS code accesses the data and writes the results to a GDX file setData.inc:

```
$call sql2gms C="Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb" Q="SELECT setElement, e

Set b;
$gdxIn setData.gdx
$load b = set_b
$gdxIn
```

Note that the name of the set in the GDX file is `set_b` (specified within the `Y` argument), while the name of the GDX file was specified within the X argument.

6.41.3.7 Example 6 - Index Mapping

In some cases the index elements used in the database are not the same as in the GAMS model. E.g. consider the case where the GAMS model has defined a set as:

```
Set i / NY, DC, LA, SF /;
```

Now assume a data table looks like:

example table

city	value
new york	100
los angeles	120
san francisco	105
washington dc	102

Figure 6.67 Table: example table

This means we have to map 'new york' to 'NY' etc. This mapping can be done in two places: either in GAMS or in the database.

Index Mapping done in GAMS

When we export the table directly, we get the following include file (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
'new york' 100
'los angeles' 120
'san francisco' 105
'washington dc' 102
* -----------------------------------------------------
```

Note that the single quotes are added by activating the option B (quote blanks), as the index elements contain blanks. Accessing the data, importing the resulting include file and converting it to a different index space can be done by the following GAMS code:

```
Set i / NY, DC, LA, SF /;

Set idb 'from database' / 'new york', 'washington dc', 'los angeles', 'san francisco' /;

$call sql2gms C="DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb" B O="city1.inc"
Parameter dbdata(idb) /
$include city1.inc
/;

Set mapindx(i,idb) / NY.'new york', DC.'washington dc', LA.'los angeles', SF.'san francisco' /;

Parameter data(i);
data(i) = sum(mapindx(i,idb), dbdata(idb));
display data;
```

The display statement generates the following output in the listing file:

```
----     47 PARAMETER data

NY 100.000,    DC 102.000,    LA 120.000,    SF 105.000
```

This example is also part of the GAMS Data Utilities Library, see model [**IndexMapping3**] for reference.

Index mapping done in Database

The second approach is to handle the mapping inside the database. We can introduce a mapping table that looks like:

CityMapper

Access City	GAMS City
los angeles	la
new york	ny
san francisco	sf
washington dc	dc

Figure 6.68 Table: CityMapper

This table can be used in a join to export the data in a format we can use by executing the query:

```
SELECT [GAMS City], value
FROM    example_table, CityMapper
WHERE   CityMapper.[Access City]=example_table.city
```

The resulting include file looks like (header informations are removed in order to shorten the representation):

```
* -----------------------------------------------------
la 120
ny 100
sf 105
dc 102
* -----------------------------------------------------
```

All steps can be combined in the GAMS model:

```
Set i / NY, DC, LA, SF /;

$onEcho > howToRead.txt
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb
Q=SELECT [GAMS City], [value] FROM example_table, CityMapper WHERE CityMapper.[Access City]=example_
O=city2.inc
$offEcho

$call sql2gms @howToRead.txt
Parameter data(i) /
$include city2.inc
/;
display data;
```

The display statement generates the following output in the listing file:

```
----     38 PARAMETER data

NY 100.000,    DC 102.000,    LA 120.000,    SF 105.000
```

Note: MS Access allows table names with embedded blanks. In that case the table name can be surrounded by square brackets. Other databases may not allow this.

This example is also part of the GAMS Data Utilities Library, see model [**IndexMapping4**] for reference.

6.41.4 Multi-Query Batch Usage

In some cases a number of small queries need to be performed on the same database. However, several individual `SQL2GMS` execution can become expensive, since there is significant overhead in starting Access and opening the database. For these cases, we have added the option to do multiple queries in one call. To execute several queries in a single `SQL2GMS` call and write several GAMS include files containing the results of the queries, we can use the command-line arguments Qn and On. The structure of a multi-query call looks like:

```
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=sample.accdb

Q1=firstQuery
O1=outputFileName.inc

Q2=secondQuery
O2=outputFileName.inc

Q3=thirdQuery
O3=outputFileName.inc
```

The terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. We see that the argument `Qn` is matched by an argument `On`. That means that the results of the n-th query are written to the n-th output file.

In case we want to store the results of a multi-query call to a single GDX file, we can use the command-line arguments Qn, Sn, An and Yn. The structure of a multi-query call when writing to a GDX file looks like:

```
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=sample.accdb
X=sample.gdx

Q1=firstQuery
S1=setName

Q2=secondQuery
S2=setName

Q3=thirdQuery
A3=parameterName

Q4=fourthQuery
A4=setName
```

Again, the terms `firstQuery`, `secondQuery` etc. are placeholders for some SQL statements. Here we see that a query `Qn` is matched by either a set name `Sn` or a parameter name `An` (the letter `P` was taken already: it is used to specify a password), i.e. the results of the first query will be stored as a set whose name is specified within the `S1` argument, the results of the third query will be stored as a parameter whose name is specified within the `A3` argument etc. The `X` argument is used to specify the name of the GDX file.

For a complete example see section Example 7 - Multi-Query Batch Example .

6.41.4.1 Example 7 - Multi-Query Batch Example

As an example database we use the following Access table (file Sample.mdb):

data

year	loc	prod	sales	profit
1997	la	hardware	80	8
1997	la	software	60	16
1997	nyc	hardware	110	5
1997	nyc	software	100	10
1997	sfo	hardware	80	9
1997	sfo	software	50	10
1997	was	hardware	120	7
1997	was	software	70	20
1998	la	hardware	70	6
1998	la	software	70	10

Figure 6.69 Table: data (shortened for presentation)

We want to extract the following information:

- The set **year**
- The set **loc**
- The set **prd**
- The parameter **sales**
- The parameter **profit**

Output: Several include Files

This can be accomplished using the following GAMS code with multiple queries in a single `SQL2GMS` call (note the usage of the distinct operator in the select clauses of the queries whose results will be used as sets in order to keep the set elements unique):

```
$onEcho > howToRead.txt
C=DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};dbq=Sample.accdb

Q1=SELECT distinct(year) FROM data
O1=year.inc

Q2=SELECT distinct(loc) FROM data
O2=loc.inc

Q3=SELECT distinct(prod) FROM data
O3=prod.inc

Q4=SELECT prod, loc, year, sales FROM data
O4=sales.inc

Q5=SELECT prod, loc, year, profit FROM data
O5=profit.inc
```

```
$offEcho

$call =sql2gms @howToRead.txt

Set y 'years' /
$include year.inc
/;
Set loc 'locations' /
$include loc.inc
/;
Set prd 'products' /
$include prod.inc
/;

Parameter sales(prd,loc,y) /
$include sales.inc
/;
display sales;

Parameter profit(prd,loc,y) /
$include profit.inc
/;
display profit;
```

This example is also part of the GAMS Data Utilities Library, see model [**SalesProfitDB6**] for reference.

Output: A single GDX File

The same example imported through a GDX file can look like:

```
$onEcho > howToRead.txt
C=Provider=Microsoft.ACE.OLEDB.12.0;Data Source=Sample.accdb
X=Sample.gdx

Q1=SELECT distinct(year) FROM data
S1=year

Q2=SELECT distinct(loc) FROM data
S2=loc

Q3=SELECT distinct(prod) FROM data
S3=prd

Q4=SELECT prod, loc, year, sales FROM data
A4=sales

Q5=SELECT prod, loc, year, profit FROM data
A5=profit
$offEcho

$call =sql2gms @howToRead.txt
$call ="%gams.sysdir%studio/studio" Sample.gdx

Set
   y   'years'
   loc 'locations'
```

```
   prd 'products';

Parameter
   sales(prd,loc,y)
   profit(prd,loc,y);

$gdxIn sample.gdx
$load y=year prd loc sales profit
$gdxIn

display sales, profit;
```

The call of the GDXViewer will display the GDX file in the stand-alone GDX viewer. This example is also part of the GAMS Data Utilities Library, see model [**SalesProfitDB7**] for reference.

6.41.5 Interactive Usage

When the tool is called without command-line arguments, it will startup interactively. Using it this way, one can specify options such as the connection string, the query and the final destination file (a GAMS include file or a GDX file) using the built-in interactive environment. The main screen (see figure below) contains a number of buttons and edit boxes, which are explained below.

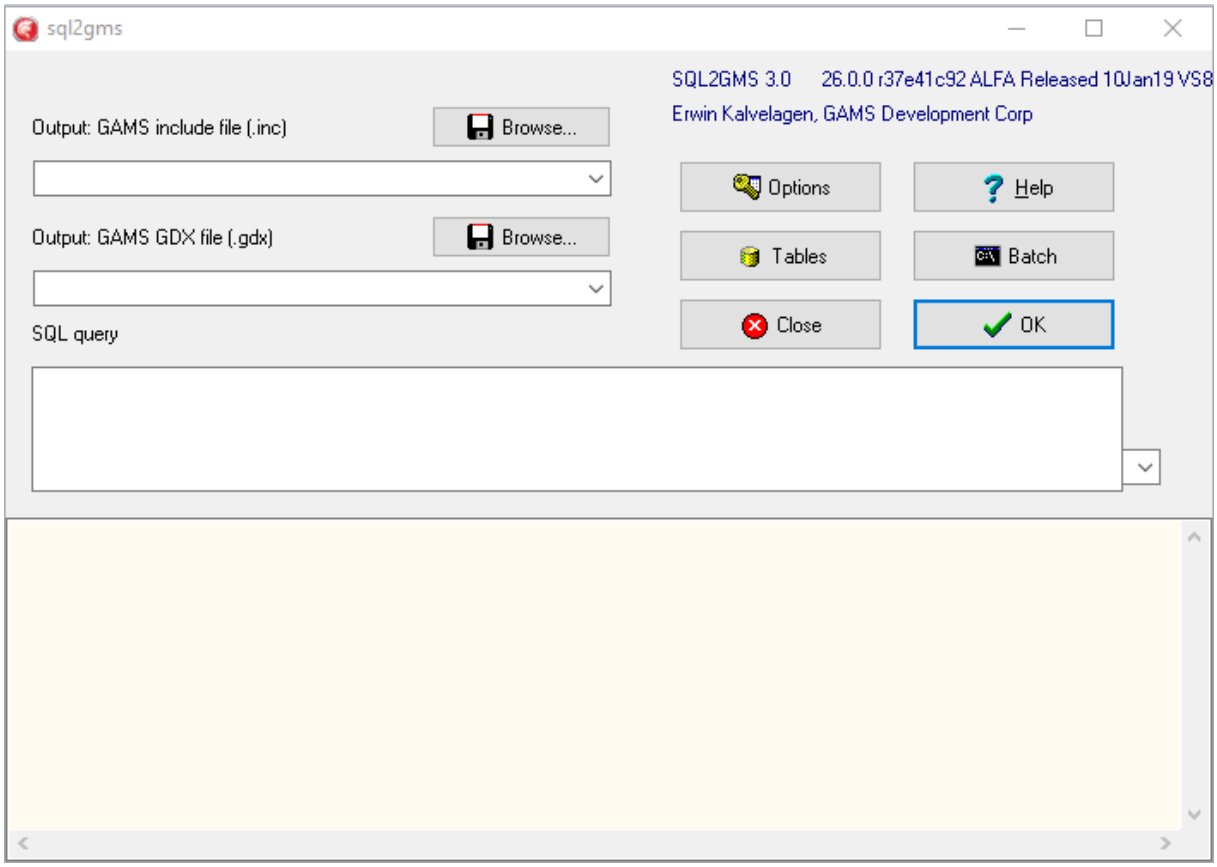
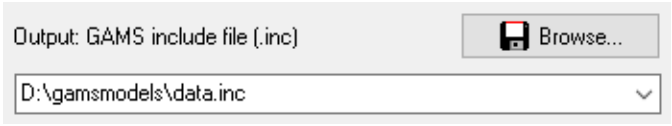
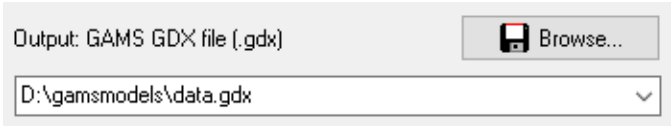


Figure 6.70 SQL2GMS - Graphical User Interface

- **Output GAMS Include file (∗.inc)**. If you want to create a GAMS include file, then specify here the destination file. See also outputIncludeFile for some more detailed remarks.

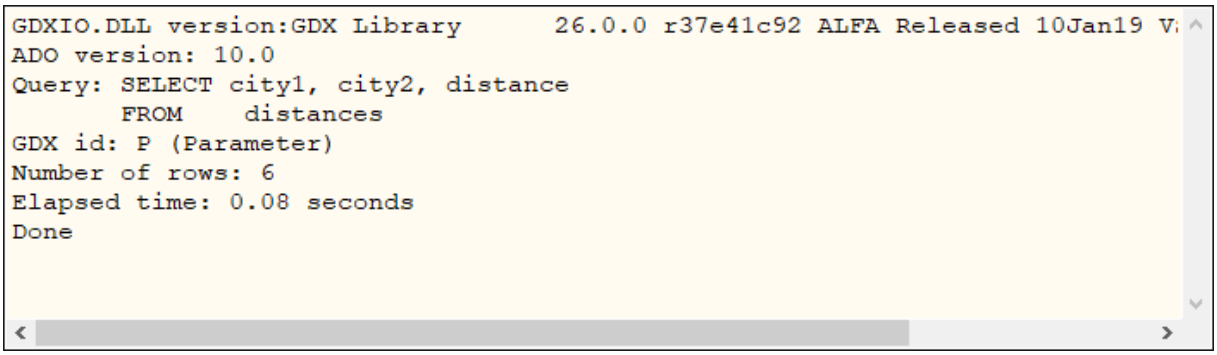


- **Output GDX file (∗.gdx)**. As an alternative to a GAMS include file, the tool can also generate a GDX file. One or both of the output files need to be specified. See also outputGDXFile for some more detailed notes.



- **SQL Query**. The SQL Query box is the place to provide the query. Note that the actual area for text can be larger than is displayed: use the cursor-keys to scroll. See also Q for some more detailed notes.

- **Progress Memo**. This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.



- The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).

- The **Tables button** will pop up a new window with the contents of the database file selected in the input file edit line. This allows you to see all table names and field names needed to specify a correct SQL query. An exception will be generated if no database file name is specified in the input edit line.

- The **Options button** will pop up a window where you can specify a number of options. The connection string is an important option, which needs to be set correctly before a query can be submitted successfully.

- Pressing the **Help button** will show this documentation.

- Pressing the **OK button** will execute the query and an output file will be generated.

- The **Batch button** will give information on how the current extract command can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied onto the clipboard. In the IDE press `Ctrl-C` or choose `Edit|Paste` to copy the contents of the clipboard to a GAMS text file.

- Pressing the **Close button** will exit the application. The current settings will be saved in an INI file so when you run `SQL2GMS` again all current settings will be restored.

6.41.5.1 Options

The **Options** window can be created by pressing the options button:

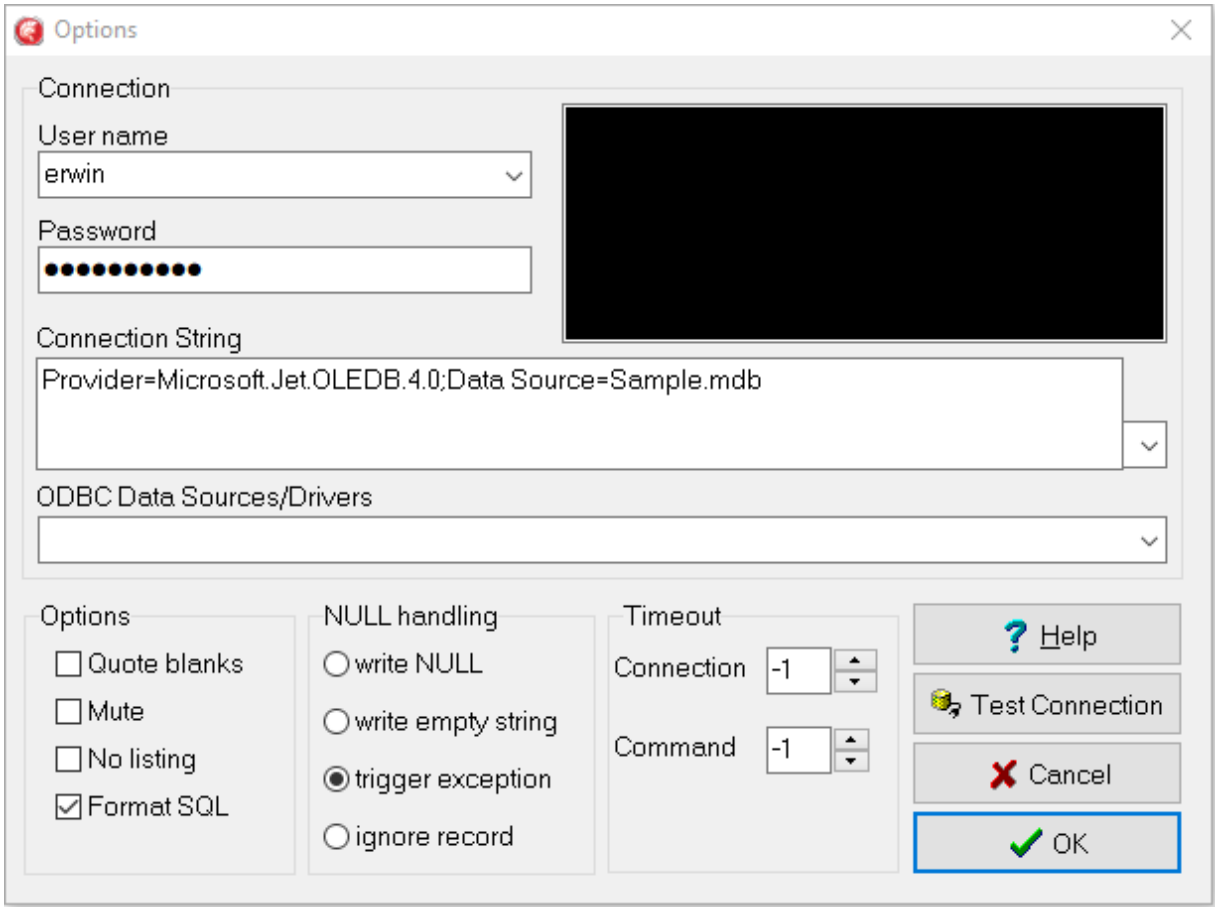


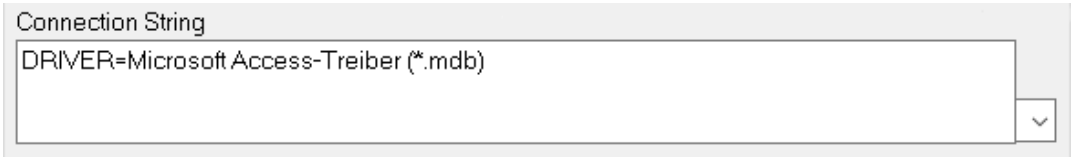
Figure 6.71 Options Menu of the Graphical User Interface

The following options are available in the options window:

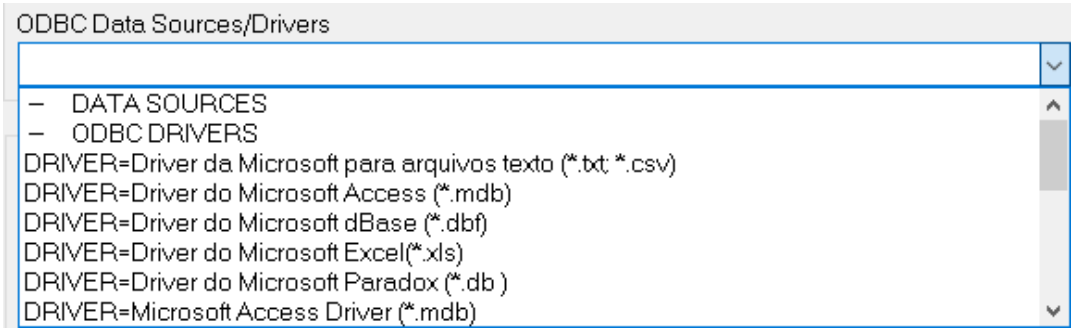
- **User name:** Here you can specify the user name for logging in to the RDBMS. For databases without user authentication, this can be left empty.

- **Password:** This edit box allows you to specify the password for the database system. The characters are echoed as a '∗'.

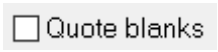
- **Connection String:** The connection string determines how `SQL2GMS` talks to the database. For more informations see Connection Strings.



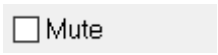
- **ODBC Data Sources/Drivers:** This drop down list can be used to compose a connection string when an ODBC data source or driver is involved. The list will show all configured data sources and all available ODBC drivers.



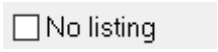
- **Quote blanks:** Quote strings if they contain blanks or embedded quotes. See also B for some more detailed notes.



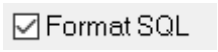
- **Mute:** Don't include the extra informational text (such as used query etc.) in the include file. The equivalent command-line argument is M.



- **No listing:** Surround the include file by `$offListing` and `$onListing` so that the data will not be echoed to the listing file. The equivalent command-line argument is L.

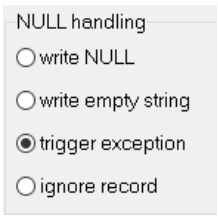


- **Format SQL:** If an SQL text is reloaded in the SQL Edit Box, it will be formatted: keywords will be printed in CAPS and the FROM and WHERE clause will be printed on their own line. If this check box is unchecked this formatting will not take place and SQL queries will be shown as is.

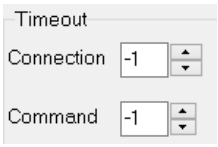


The following options are only needed in special cases:

- **NULL:** This radio box determines how NULL's are handled. A NULL in an index position or a value column will usually make the result of the query useless: the GAMS record will be invalid. To alert you on NULL's the default to throw an exception is a safe choice. In special cases you may want to map NULL's to an empty string or a 'NULL' string.



- **Time-out** values for connection time and command execution time expressed in seconds. If -1 is specified then it will use the default value which is 15 seconds for the connection and 30 for the commands. See also T1, T2 resp. T for some more detailed notes.



The following buttons have an obvious functionality:

- **OK button** will accept the changes made.

- **Cancel button** will ignore the changes made and all option settings will revert to their previous values.

- **Help button** will show this documentation.

- **Test Connection** will try to make a connection to the database using the given connection string.

If the settings are correct you will see something like:

The following options can only be specified in an INI file; there is no interactive equivalent:

Key	Type	Meaning
D		Generate debug output
E		Allow an empty result set; without this option an empty result set will generate an error
R	integer	Row batch size; the default is 100 records

6.41.6 Connection Strings

The connection string determines to which database the tool will try to connect. You can give simply the name of an ODBC Data Source or provide much more elaborate connection strings. Here is an example list:

- ODBC Examples
- **OLE DB Examples**
- **MS Remote Examples**

For more information consult the documentation with your database driver. ODBC drivers can be had from several sources: Microsoft delivers ODBC with a number of drivers; database providers may have likely an ODBC driver for their RDBMS available and finally there are a number of third party ODBC drivers available (e.g. from `http://www.easysoft.com`).

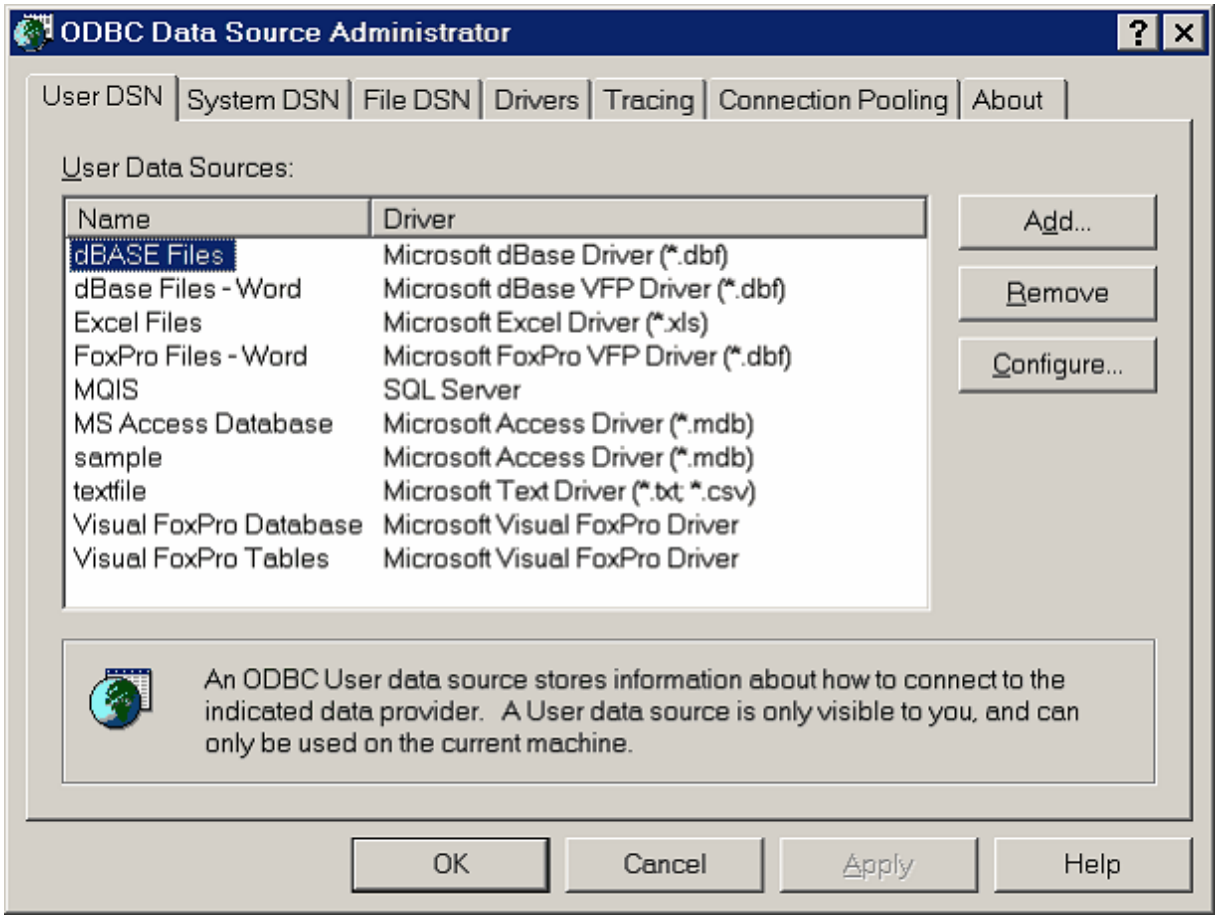
6.41.7 ODBC Examples

In this section we show a few examples using `SQL2GMS` with ODBC data sources.

6.41.7.1 ODBC Driver Manager

To configure ODBC data sources use the `ODBC Data Source Administrator`. This tool can be invoked from the Start button menu: `Settings`|`Control Panel`, and click on the `ODBC Data Sources` icon:

Under Windows XP the sequence is: `Control Panel`|`Performance` and `Maintenance`|`Administrative Tools` and click on the `Data Sources (ODBC)` icon. Under Windows 10, access the `Control Panel` at first, type 'odbc' in the top right box and select either 'Set up ODBC data sources (32-bit)' or 'Set up ODBC data sources (64-bit)'. The ODBC Data Source Administrator tool looks like:



To create a new data source, click the Add button, select a driver, give it a name (this is the DSN name) and configure the data source.

6.41.7.2 Example 8 - Reading from an MS Access Database

There are several ways to export data from an SQL database into a GAMS include file:

1. Export a CSV (Comma Separated Values) file using Access Export. See also CSV Files.
2. Use the MDB2GMS tool.
3. Use `SQL2GMS` with a configured ODBC data source. The connection string will look like:

   ```
   "DSN=mydsn".
   ```

4. Use `SQL2GMS` with a DSN-less ODBC connection. In this case we need to specify both the driver and the location of the database file explicitly in the connection string. The connection string will look like:

   ```
   "Driver=Microsoft Access Driver (*.mdb, *.accdb);dbq=D:\data\mydata.accdb".
   ```

5. Use `SQL2GMS` with an OLE DB driver. The connection string can look like:

   ```
   "Microsoft.ACE.OLEDB.12.0;Data Source=D:\data\mydata.accdb"
   ```

6.41.7.3 Example 9 - Reading from an MS Excel Spreadsheet

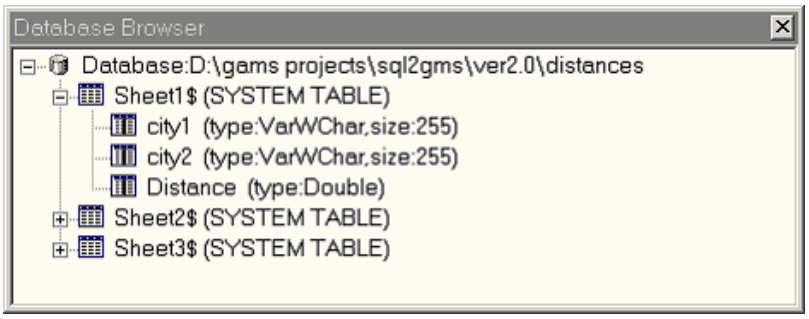
There are numerous ways to export data from an Excel spreadsheet into a GAMS include file:

1. Export a CSV (Comma Separated Values) file using Excel Export.
2. Use the XLS2GMS tool.
3. Use the GDXXRW tool.
4. Use `SQL2GMS` with an Excel ODBC connection. An example is shown below.

Consider the spreadsheet ExcelDist.xls:

	A	B	C	D	E
1					
2		city1	city2	Distance	
3		SEATTLE	NEW-YORK	2.5	
4		SAN-DIEGO	NEW-YORK	2.5	
5		SEATTLE	CHICAGO	1.7	
6		SAN-DIEGO	CHICAGO	1.8	
7		SEATTLE	TOPEKA	1.8	
8		SAN-DIEGO	TOPEKA	1.4	
9					

After configuring a data source ExcelDist that uses the Excel ODBC driver and points to the .xls file containing the above sheet, we can use the connection string: `"DSN=ExcelDist"`. With the database browser we see:



I.e. the table name is `Sheet1$`. We now can formulate the query: `SELECT city1, city2, distance FROM [Sheet1$]`. We need the brackets to protect the funny table name. The result is (header removed in order to shorten the presentation=:

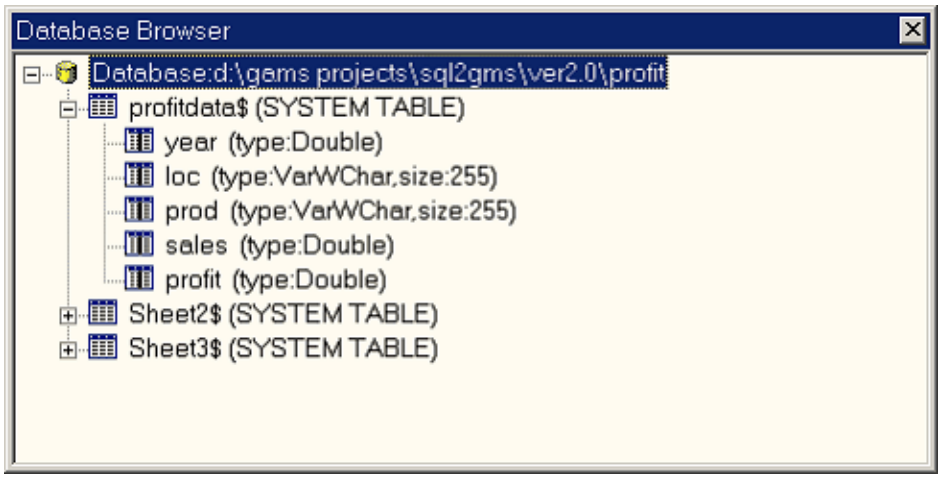
```
* -----------------------------------------------------
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* -----------------------------------------------------
```

Although other tools are often more convenient to use, this approach is useful if you need to select a subsection of the spreadsheet table. It is easy to select just a few columns or rows from a table using a properly formulated SQL query. The skeleton would be: `SELECT columns_to_extract FROM [sheet1$] WHERE rows_to_extract`.

An example of a more complex spreadsheet is (profit.xls):

	A	B	C	D	E	F
1	year	loc	prod	sales	profit	
2	1997	la	hardwar	80	8	
3	1997	la	software	60	16	
4	1997	nyc	hardwar	110	5	
5	1997	nyc	software	100	10	
6	1997	sfo	hardwar	80	9	
7	1997	sfo	software	50	10	
8	1997	was	hardwar	120	7	
9	1997	was	software	70	20	
10	1998	la	hardwar	70	6	
11	1998	la	software	70	10	
12	1998	nyc	hardwar	120	7	
13	1998	nyc	software	120	14	
14	1998	sfo	hardwar	90	12	
15	1998	sfo	software	70	15	
16	1998	was	hardwar	130	12	
17	1998	was	software	80	15	
18						

A DSN-less connection string would be: `"DRIVER=Microsoft Excel Driver (*.xls); DBQ=d:\gams projects\sql2gms\ver2.0\profit.xls"`. The browser will show:



A possible query that maps the two value columns into a GAMS parameter is:

```
SELECT year, loc, prod, 'sales', sales
FROM  [profitdata$]
UNION
SELECT year, loc, prod, 'profit', profit
FROM  [profitdata$]
```

The result is (shortened for presentation):

```
* -----------------------------------------------------
1997.la.hardware.profit 8
1997.la.hardware.sales 80
1997.la.software.profit 16
1997.la.software.sales 60
1997.nyc.hardware.profit 5
1997.nyc.hardware.sales 110
1997.nyc.software.profit 10
1997.nyc.software.sales 100
1997.sfo.hardware.profit 9
1997.sfo.hardware.sales 80
1997.sfo.software.profit 10
1997.sfo.software.sales 50
1997.was.hardware.profit 7
1997.was.hardware.sales 120
1997.was.software.profit 20
1997.was.software.sales 70
1998.la.hardware.profit 6
1998.la.hardware.sales 70
* -----------------------------------------------------
```

This example is also part of the GAMS Data Utilities Library, see model [**Excel**] for reference.

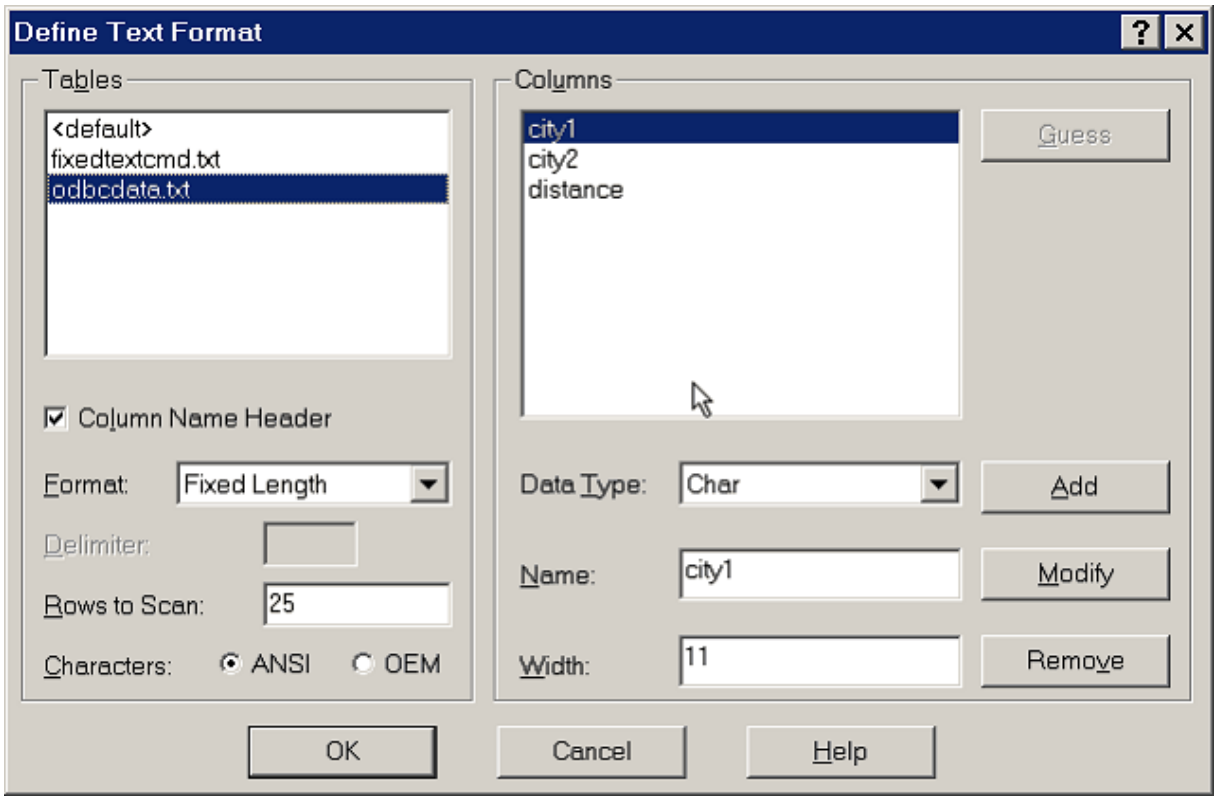
6.41.7.4 Example 10 - Reading from a Text File

Microsoft delivers ODBC with a text file driver which allows you to read a text file as if it is a database table.

A fixed format file such as:

```
City1      City2     Distance
SEATTLE    NEW-YORK  2.5
SAN-DIEGO  NEW-YORK  2.5
SEATTLE    CHICAGO   1.7
SAN-DIEGO  CHICAGO   1.8
SEATTLE    TOPEKA    1.8
SAN-DIEGO  TOPEKA    1.4
```

can be read using Fixed Length setting of the text driver:



The resulting include file will look like:

```
* ----------------------------------------------------
* SQL2GMS Version 2.0, January 2004
* Erwin Kalvelagen, GAMS Development Corp
* ----------------------------------------------------
* ADO version:       2.7
* Connection string: DSN=text
* Query:             SELECT city1, city2, distance FROM odbcdata.txt
* Provider:          MSDASQL
* ----------------------------------------------------
```

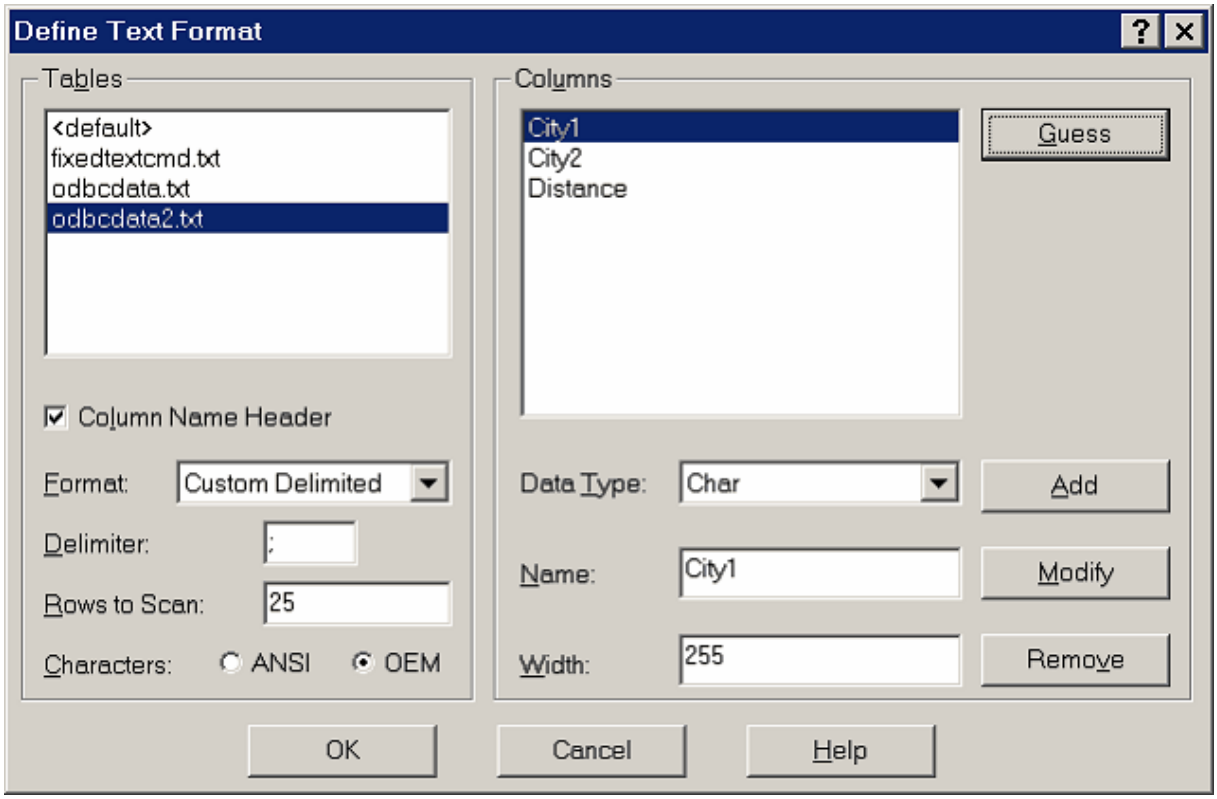
```
SEATTLE.NEW-YORK 2.5
SAN-DIEGO.NEW-YORK 2.5
SEATTLE.CHICAGO 1.7
SAN-DIEGO.CHICAGO 1.8
SEATTLE.TOPEKA 1.8
SAN-DIEGO.TOPEKA 1.4
* ------------------------------------------------------
```

Note that the text file is specified directly within the FROM clause.

A CSV file can be interpreted as a table as well, or any other separated format. We will try to read:

```
City1;City2;Distance
SEATTLE;NEW-YORK;2.5
SAN-DIEGO;NEW-YORK;2.5
SEATTLE;CHICAGO;1.7
SAN-DIEGO;CHICAGO;1.8
SEATTLE;TOPEKA;1.8
SAN-DIEGO;TOPEKA;1.4
```

This can be read using:



The actual formats used are stored by ODBC in an INI file schema.ini (located in the directory of the data files) which can be inspected directly:

```
[odbcdata.txt]
ColNameHeader=True
Format=FixedLength
MaxScanRows=25
CharacterSet=ANSI
Col1=city1 Char Width 11
Col2=city2 Char Width 11
Col3=distance Float Width 10

[odbcdata2.txt]
ColNameHeader=True
Format=Delimited(;)
MaxScanRows=25
CharacterSet=OEM
Col1=City1 Char Width 255
Col2=City2 Char Width 255
Col3=Distance Float
```

This example is also part of the GAMS Data Utilities Library, see model [**Text**] for reference.

6.41.8 Strategies

Including SQL statements to extract data from a database inside your model can lead to a number of difficulties:

- The database can change between runs, leading to results that are not reproducible. A possible scenario is a user calling you with a complaint: "the model is giving strange results". You run the model to verify and now the results are ok. The reason may be because the data in the database has changed.
- There is significant overhead in extracting data from a database. If there is no need to get new data from the database it is better to use a snapshot stored locally in a format directly accessible by GAMS.
- It is often beneficial to look at the extracted data. A first reason, is just to make sure the data arrived correctly. Another argument is that viewing data in a different way may lead to a better understanding of the data. A complete "under-the-hood" approach may cause difficulties in understanding certain model behavior.

Often it is a good strategy to separate the data extraction step from the rest of the model logic.

If the sub-models form a chain or a tree, like in:

```
Data Extraction --> Data Manipulation --> Model Definition --> Model Solution --> Report Writing
```

we can conveniently use the save/restart facility. The individual submodel are coded as:

- **Step 0 - Data Extraction: sr0.gms**

```
$onText
step 0: data extraction from database
execute as: > gams sr0 save=s0
$offText

Set
   i 'suppliers'
   j 'demand centers';

Parameter
   demand(j)
   supply(i)
   dist(i,j) 'distances';

$onEcho > howtoRead.txt
C=Provider=Microsoft.ACE.OLEDB.12.0;Data Source=%system.fp%transportation.accdb

Q1=SELECT name FROM suppliers
O1=i.inc

Q2=SELECT name FROM demandcenters
O2=j.inc

Q3=SELECT name,demand FROM demandcenters
O3=demand.inc

Q4=SELECT name, supply FROM suppliers
O4=supply.inc

Q5=SELECT supplier, demandcenter, distance FROM distances
O5=dist.inc
$offEcho

$call =sql2gms.exe @howtoRead.txt

Set i /
$include i.inc
/;

Set j /
$include j.inc
/;

Parameter demand /
$include demand.inc
/;

Parameter supply /
$include supply.inc
/;

Parameter dist /
$include dist.inc
/;

display i, j, demand, supply, dist;
```

- **Step 1 - Data Manipulation: sr1.gms**

```
$onText
step 1: data manipulation step
execute as: > gams sr1 restart=s0 save=s1
$offText

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*dist(i,j)/1000;
```

- **Step 2 - Model Definition: sr2.gms**

```
$onText
step 2: model definition
execute as: > gams sr2 restart=s1 save=s2
$offText

Variable
   x(i,j) 'shipment quantities in cases'
   z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
   ecost      'define objective function'
   esupply(i) 'observe supply limit at plant i'
   edemand(j) 'satisfy demand at market j';

ecost..      z  =e= sum((i,j), c(i,j)*x(i,j));

esupply(i).. sum(j, x(i,j)) =l= supply(i);

edemand(j).. sum(i, x(i,j)) =g= demand(j);
```

- **Step 3 - Model Solution: sr3.gms**

```
$onText
step 3: model solution
execute as: > gams sr3 restart=s2 save=s3
$offText

option lp = cplex;

Model transport / all /;
solve transport using lp minimizing z;
```

- **Step 4 - Report Writing: sr4.gms**

```
$onText
step 4: report writing
execute as: > gams sr4 restart=s3
$offtext

abort$(transport.modelStat <> 1) "model not solved to optimality";

display x.l, z.l;
```

A model that executes all steps can be written as:

```
execute '=gams.exe sr0 lo=3 save=s0';
abort$errorLevel "step 0 failed";

execute '=gams.exe sr1 lo=3 restart=s0 save=s1';
abort$errorLevel "step 1 failed";

execute '=gams.exe sr2 lo=3 restart=s1 save=s2';
abort$errorLevel "step 2 failed";

execute '=gams.exe sr3 lo=3 restart=s2 save=s3';
abort$errorLevel "step 3 failed";

execute '=gams.exe sr4 lo=3 restart=s3';
abort$errorLevel "step 4 failed";
```

If you only change the reporting step, i.e. generating some output using `PUT` statements, then you only need to change and re-execute step 4. If you change solver or solver options, then only steps 3 and 4 need to be redone. For a small model like this, this exercise may not be very useful, but when the model is large and every step is complex and expensive, this is a convenient way to achieve quicker turn-around times in many cases.

The model [**SQLSr5**] is also part of the GAMS Data Utilities Library.

In some cases the save/restart facility is not appropriate. A more general approach is to save the data from the database in a GDX file, which can then be used by other models. We can use the model from step 0 to store the data in a GDX file:

SQL2GDX1.gms

```
$onText
Store data from Access database into a GDX file.
$offText

execute '=gams.exe sr0 lo=3 gdx=trnsport.gdx';
abort$errorLevel "step 0 failed";

execute '=gdxviewer.exe trnsport.gdx';
```

The model [**SQL2GDX1**] is also part of the GAMS Data Utilities Library.

We can also let `SQL2GMS` create the GDX file:

SQL2GDX2.gms

```
$onText
Store data from Access database into a GDX file.
$offText

$onEcho > howToRead.txt
C=Provider=Microsoft.ACE.OLEDB.12.0;Data Source=%system.fp%transportation.accdb
X=%system.fp%transportation.gdx

Q1=SELECT name FROM suppliers
S1=i

Q2=SELECT name FROM demandcenters
```

```
S2=j

Q3=SELECT name, demand FROM demandcenters
A3=demand

Q4=SELECT name, supply FROM suppliers
A4=supply

Q5=SELECT supplier, demandcenter, distance FROM distances
A5=dist
$offEcho

$call =sql2gms.exe @howToRead.txt
```

The first approach has the advantage that a complete audit record is available from the data moved from the database to the GDX file in the sr0.lst listing file. If someone ever wonders what came out of the database and how this was stored in the GDX file, that file gives the answer.

The model [**SQL2GDX2**] is also part of the GAMS Data Utilities Library.

To load the GDX data the following fragment can be used:

GDXTRNSPORT.gms

This model demonstrates how to load the transportation data from GDX file at compile time.

```
Set
   i 'suppliers'
   j 'demand centers';

Parameter
   demand(j)
   supply(i)
   dist(i,j) 'distances';

$gdxIn transportation.gdx
$load i j demand supply dist
$gdxIn

display i, j, demand, supply, dist;
```

DBTimestamp2.gms

In one application I had to retrieve data from the database each morning, at the first run of the model. The rest of the day, the data extracted that morning could be used. The following logic can implement this:

```
$onText
Retrieve data from data base first run each morning.
$offText

$onEcho > getdate.txt
C=Provider=Microsoft.ACE.OLEDB.12.0;Data Source=%system.fp%transportation.accdb
Q=select day(now())
O=dbtimestamp.inc
$offEcho
```

```
$if not exist dbtimestamp.inc $call "echo 0 > dbtimestamp.inc"

Scalar dbtimestamp 'day of month when data was retrieved' /
$include dbtimestamp.inc
/;

Scalar currentday 'day of this run';
currentday = gday(jnow);

display "compare", dbtimestamp, currentday;

if(dbtimestamp<>currentday,
   execute '=gams.exe sr0 lo=3 gdx=transportation.gdx';
   abort$errorLevel "step 0 (database access) failed";

   execute '=sql2gms.exe @getdate.txt'
);
```

The include file dbtimestamp.inc contains the day of the month (1,..,31) on which the data was extracted from the database. If this file does not exist, we initialize it with 0. We then compare this number with the current day of the month. If the numbers do not agree, we execute the database extraction step and rewrite the dbtimestamp.inc file. This last operation could be done using a `PUT` statement, but in this case we used an SQL statement.

The model [**DBTimestamp2**] is also part of the GAMS Data Utilities Library.

6.41.9 Parameter Files

Parameters can be specified in an external parameter file. This is important if the length of the command-line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command-lines. Instead of specifying a long command line as in:

```
$call =sql2gms C="DSN=sample" O="c:\My Documents\data.inc" Q="SELECT * FROM mytable"
```

we can use a command line like:

```
$call =sql2gms @"c:\My Documents\options.txt"
```

The parameter file

```
c:\My Documents\options.txt
```

can look like:

```
C=DSN=sample
O=c:\My Documents\data.inc
Q=SELECT * FROM mytable
```

It is possible to write the parameter file from inside a GAMS model using the $echo command. The following example will illustrate this:

```
$set cmdfile "c:\windows\temp\commands.txt"
$echo "C=DSN=sample"              >  "%cmdfile%"
$echo "O=E:\models\labor.INC"     >> "%cmdfile%"
$echo "Q=SELECT * FROM labor"     >> "%cmdfile%"
$call =sql2gms @"%cmdfile%"

Parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

Newer versions of GAMS allow the usage of the $onEcho and $offEcho commands:

```
$set cmdfile "c:\windows\temp\commands.txt"
$onEcho  >  "%cmdfile%"
C=DSN=sample
O=E:\models\labor.INC
Q=SELECT * FROM labor
$offEcho

$call =sql2gms @"%cmdfile%"

Parameter p /
$include "E:\models\labor.INC"
/;
display p;
```

If a query becomes very long, it is possible to spread it out over several lines. To signal a setting will continue on the next line insert the character \ as the last character. E.g.:

```
Q=SELECT prod, loc, year, 'sales', sales FROM data \
UNION \
SELECT prod, loc, year, 'profit', profit FROM data
```

6.41.10 Notes

6.41.10.1 ADO

ActiveX Data Objects. Microsoft's data-access object model. An object-oriented architecture for accessing data stored in SQL databases and related data sources. Accessible from a large number of host languages such as VB, C++, Delphi. Supersedes ODBC. Many SQL databases provide ADO access through either OLEDB or ODBC.

6.41.10.2 ODBC

An API and driver manager system for accessing data stored in an RDBMS. The API provides applications a way to talk to databases while the driver manager application allows users to install, configure and manage ODBC drivers for different databases.

6.41.10.3 OLEDB

Driver architecture for SQL databases. A driver is called a OLE DB provider. This is used from ADO.

6.41.10.4 UNC Names

UNC means Unified Naming Convention. UNC names are a Microsoft convention to name files across a network. The general format is:

```
\\<server>\<share>\<path>\<file>
```

Examples:

```
\\athlon\c\My Documents\sql2gms.rtf
```

6.41.10.5 GDX Files

A GDX file contains GAMS data in binary format. The following GAMS commands will operate on GDX files: $gdxIn, $load, execute_load, execute_unload. The GDX=filename command-line option will save all data to a GDX file. A GDX file can be viewed in the GAMS IDE using `File|Open`.

6.41.10.6 MDB2GMS

`MDB2GMS` is a tool to import tables from MS Access databases. This utility directly uses MS Access and DAO (Data Access Objects) resulting in a somewhat simpler interface. It is not needed to specify a connection string, but just a .accdb or .mdb file. The query mechanism is similar: a query is sent as it is to the database server and the result set is translated into a GAMS representation. For more information see MDB2GMS.

6.41.10.7 XLS2GMS

`XLS2GMS` is a tool to import data from an Excel spreadsheet. It considers the content of a selected range as GAMS source code. For more information XLS2GMS.

6.41.10.8 GDXXRW

GDXXRW is a utility to read and write Excel spreadsheet data. GDXXRW can read multiple ranges in a spreadsheet and write the data to a GDX file, or read from a GDX file, and write the data to different ranges in a spreadsheet. For more information GDXXRW.

6.41.10.9 Quotes

Examples of handling of indices when the option B for quoting strings containing blanks is used:

6.41.10.10 $CALL Command

The $call command in GAMS will execute an external program at compile time. There are two forms:

```
$call externalProgram
```

```
$call =externalProgram
```

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=' bypasses the command processor and directly executes the external program. We mention some of the differences:

1. Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE. If you want to execute these commands you will need to use the form $call externalProgram which uses the command processor.
2. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form $call externalProgram.
3. If it is important to stop with an appropriate error message if the external program does not exist, only use the form $call =externalProgram. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: $call =externalProgram.
4. When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: $call start /w externalWindowsProgram. Otherwise, it is preferred to use: $call =externalWindowsProgram.

Attention

In general it is recommended to use the $call =externalProgram version for its better error-handling.

When command-line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

```
$call externalProgram parameter1 parameter2
$call =externalProgram parameter1 parameter2
```

The total length of the command-line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

```
$call '"external program" "parameter 1" "parameter 2"'
$call ="external program" "parameter 1" "parameter 2"
```

It is noted that the first form needs additional quotes around the whole command-line due to bugs in the parsing of the $call in GAMS. The second form work without additional quotes *only if* the = appears outside the double quotes.

6.41.10.11 Compile Time Commands

All $ commands in GAMS are performed at compile time. All other statements are executed at execution time. This means that a compile time command will be executed **before** an execution time command, even if it is below. As an example consider:

```
File batchfile / x.bat /;
putClose batchfile "dir"/;
$call x.bat
```

This fragment does not work correctly as already during compilation, the $call is executed, while the put statements are only executed after the compilation phase has ended and GAMS has started the execution phase. The above code can be fixed by moving the writing of the batch file to compilation time as in

```
$echo "dir" > x.bat
$call x.bat
```

or by moving the external program invocation to execution time:

```
File batchfile / x.bat /;
putClose batchfile "dir"/;
execute x.bat;
```

Notice that all $ commands do not include a semi-colon but are terminated by the end-of-line.

6.42 XLS2GMS

Author

Erwin Kalvelagen, GAMS Development Corp

Version

2.4

Date

May 2004

6.42.1 Overview

Attention

XLS2GMS is deprecated (see GAMS 42 XLS2GMS release notes). Please use Connect agent ExcelReader instead.

XLS2GMS is a tool to convert spreadsheet data from a Microsoft Excel spreadsheet into GAMS readable format. The source is a MS Excel spreadsheet file (∗.XLS) and the target is a GAMS Include File.

When running the executable **XLS2GMS.EXE** without command line parameters the tool will run interactively with a built-in GUI interface. Alternatively XLS2GMS can be run in batch mode which is useful when calling it directly from a GAMS model using the $call command.

The philosophy of the tool is to consider the content of a spreadsheet as **Text**. This text can contain GAMS statements, or parts of GAMS statements (e.g. the data part of a table statement). The text is exported to a GAMS include file where some spacing is introduced to maintain cell boundaries. This allows tables to be exported directly to GAMS include files.

6.42.2 Requirements

XLS2GMS runs only on PC's running Windows (95/98/NT/XP) and with MS Excel installed. Microsoft Excel is included in the MS Office suite.

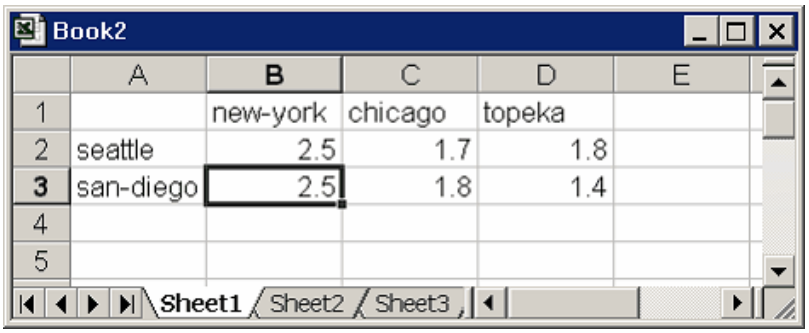
6.42.3 Converting spreadsheet data to GAMS data

Spreadsheet data are often differently organized than is suitable for import into a GAMS model. In some cases the data is scattered around different sheets, and in a format that is not compatible with a more structured multi-dimensional parameter as are used in a GAMS model. To export spreadsheet data to GAMS parameters, tools will either require a strict format to be used inside the spreadsheet or they will need to offer a complex specification step where the data representation in the spreadsheet is described so that it can be understood by the tool. This tool will use the first approach: the modeler is required to lay-out the data in the spreadsheet in a well defined format. Instead of defining a new format, we use the GAMS language syntax as the required representation. In effect the spreadsheet is considered as an alternative editor for GAMS source code.

As an example consider the GAMS table in the model [TRNSPORT] which is part of the GAMS model library:

```
Table d(i,j) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;
```

This table can be expressed comfortably in a spreadsheet as follows:



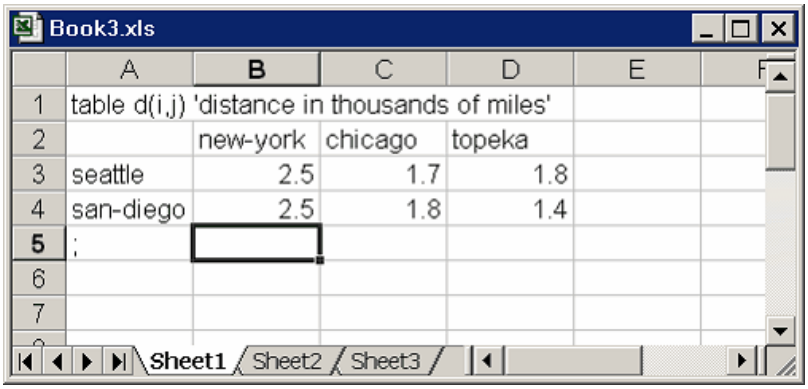
XLS2GMS can convert this table into a GAMS include file, which results in:

```
* ------------------------------------------------------
* XLS2GMS Version 2.4,  May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* ------------------------------------------------------
* Application: Microsoft Excel
* Version:     9.0
* Workbook:    D:\gams projects\xls2gms\ver2.0\Book2.xls
* Sheet:       Sheet1
* Range:       $A$1:$D$3
* ------------------------------------------------------
        new-york chicago topeka
seattle   2.5      1.7     1.8
san-diego 2.5      1.8     1.4
* ------------------------------------------------------
```

The tool will try to keep cells in a column aligned so that table statements can be used in the GAMS model. The above file **book2.inc** can be imported directly into a GAMS model by:

```
Table d(i,j) 'distance in thousands of miles'
$include book2.inc
;
```

As the tool does not expect any special formatting, we could have include the table statement in the spreadsheet, as in:



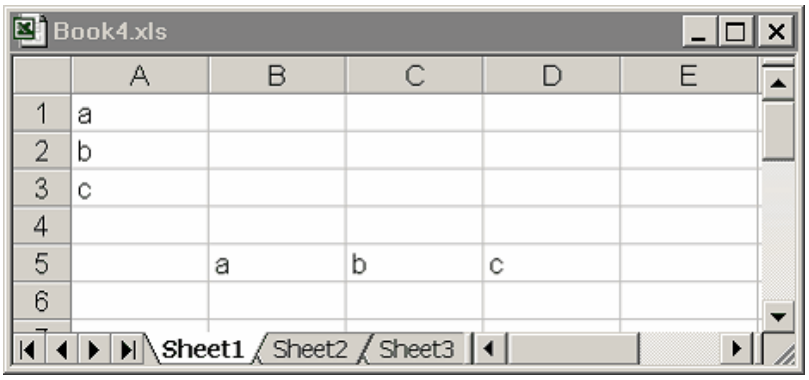
This would result in:

```
* ---------------------------------------------------
* XLS2GMS Version 2.4,  May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* ---------------------------------------------------
* Application: Microsoft Excel
* Version:     9.0
* Workbook:    D:\gams projects\xls2gms\ver2.0\Book3.xls
* Sheet:       Sheet1
* Range:       $A$1:$D$5
* ---------------------------------------------------
table d(i,j) 'distance in thousands of miles'
        new-york chicago topeka
seattle   2.5      1.7     1.8
san-diego 2.5      1.8     1.4
;
* ---------------------------------------------------
```

In some cases the data will need to be copied and massaged to fit into the above format. It is convenient to add a sheet dedicated for this purpose to your workbook. This interface sheet can be filled either manually, with formulas that automatically update values, or by macro's (either recorded or programmed in VBA).

6.42.4 Importing sets

Sets can be directly imported if they are organized vertically. The following picture shows a spreadsheet with two sets with elements {a,b,c} organized vertically (A1:A3) and horizontally (B5:D5).



The first set can be imported directly using:

```
Set i /
$call =xls2gms r=a1:a3 i=book4.xls o=set1.inc
$include set1.inc
/;

display i;
```

The second set is somewhat more difficult, as we need to add a separating comma between the elements. This can be accomplished by:

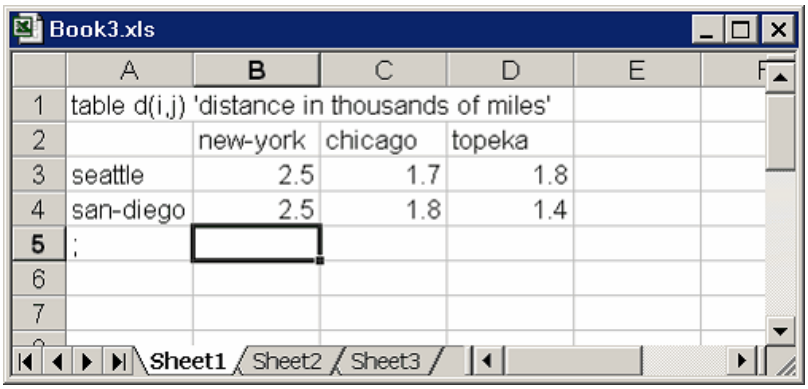
```
Set j /
$call =xls2gms r=b5:d5 s="," i=book4.xls o=set2.inc
$include set2.inc
/;

display j;
```

This will generate the include file:

```
* ------------------------------------------------------
* XLS2GMS Version 2.4,  May 2004
* Erwin Kalvelagen, GAMS Development Corp.
* ------------------------------------------------------
* Application: Microsoft Excel
* Version:     9.0
* Workbook:    D:\gams projects\xls2gms\ver2.0\Book4.xls
* Sheet:       Sheet1
* Range:       $B$5:$D$5
* ------------------------------------------------------
a,b,c
* ------------------------------------------------------
```

6.42.5 Importing sets and tables

The table



can be considered to contain three pieces of GAMS data:

- The `set i (seattle, san-diego)`
- The `set j (new-york, chicago, topeka)`
- The `distances`

All this information can be read as follows:

```
Set i /
$call =xls2gms r=a3:a4 i=book3.xls o=seti.inc
$include seti.inc
/;

Set j /
$call =xls2gms r=b2:d2 s="," i=book3.xls o=setj.inc
$include setj.inc
/;

Table d(i,j)
$call =xls2gms r=a2:d4 i=book3.xls o=pard.inc
$include pard.inc
;

display i, j, d;
```

The above $call statements can be combined onto one as follows:

```
$onEcho > book3a.txt
i=%system.fp%book3.xls
r1=a3:a4
o1=seti.inc
r2=b2:d2
o2=setj.inc
s2=","
r3=a2:d4
o3=pard.inc
$offEcho

$call =xls2gms @book3a.txt

Set i /
```

```
$include seti.inc
/;

Set j /
$include setj.inc
/;

Table d(i,j)
$include pard.inc
;

display i, j, d;
```

The command file generated by the $onecho statement looks like:

```
i=E:\wtools\ver000\examples\book3.xls
r1=a3:a4
o1=seti.inc
r2=b2:d2
o2=setj.inc
s2=","
r3=a2:d4
o3=pard.inc
```

6.42.6 Multidimensional parameters

Consider the data table:

data.txt

	A	B	C	D	E	F	G	H	I	J
1	livestock yield time series (kg per head)									
2					1974	1975	1976	1977	1978	1979
3										
4		sheep	.	meat	10.6	11.42	10.6	9.38	8.97	6.93
5		sheep	.	milk	23.7	24.1	24.2	24.2	24	23.9
6		sheep	.	wool	1.3	1.3	1.3	1.3	1.3	1.3
7		sheep	.	hide	0.5	0.6	0.6	0.5	0.6	0.4
8		goat	.	meat	6.39	7.31	8.68	7.31	6.39	6.85
9		goat	.	milk	37.7	38.1	38.2	38.2	38.3	37.8
10		goat	.	wool	0.6	0.6	0.6	0.6	0.6	0.6
11		goat	.	hide	0.2	0.3	0.3	0.3	0.2	0.3
12		angora	.	meat	1.77	1.77	2.66	2.21	1.77	1.77
13		angora	.	milk	14.9	15.2	14.8	15.2	14.8	15
14		angora	.	wool	1.6	1.6	1.6	1.6	1.6	1.4
15		angora	.	hide	0.1	0.1	0.1	0.1	0.1	0.1
16		cattle	.	meat	24.59	25.12	21.42	23	18.25	25.12
17		cattle	.	milk	210	208.1	219.8	213.8	214.8	217.5
18		cattle	.	hide	3.3	3.4	2.9	3	2.6	3.3
19		buffalo	.	meat	43.73	45.42	40.61	37.21	32.2	32.68
20		buffalo	.	milk	267.1	269.2	263.8	219.6	275.5	285.1
21		buffalo	.	hide	4.1	3.4	3	2.4	2.5	2.6
22		poultry	.	meat	2.24	2.24	2.24	2.24	2.24	2.24
23		poultry	.	egg	62.4	62.2	64.2	78.3	76.4	73.3
24										

data

In this spreadsheet the first two columns are index columns. To make this valid GAMS syntax we need to insert a dot between the index elements. A simple way is to insert a (narrow) column with a dot in each cell. This way we can import this table as:

```
Set
   l  'livestock types'      / sheep, goat, angora, cattle, buffalo, mule, poultry /
   cl 'livestk comm'         / meat, milk, wool, hide, egg /
   ty 'time periods - years' / 1974*1979 /;

$onEcho > yield.txt
I="%system.fp%yield.xls"
R=data!B2:J23
O=yield.inc
$offEcho

$call =xls2gms @yield.txt

Table yieldtl(l,cl,ty) 'livestock "yield" time series (kg per head)'
$include yield.inc
;

display yieldtl;
```

6.42.7 Interactive use

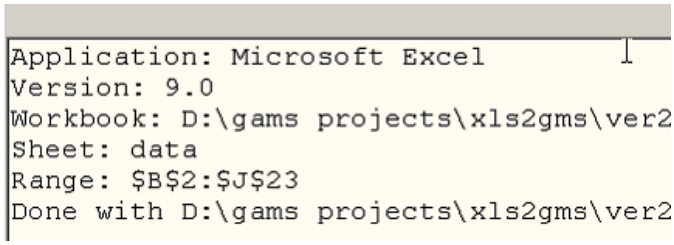
When the tool is called without command line parameters, it will startup interactively. Using it this way, one can specify the spreadsheet file (.XLS file), the range and the final destination file (a GAMS include file) using the built-in interactive environment. The main screen contains a number of buttons and edit boxes, which are explained below.

- **Input file (∗.XLS)**. This is the combo box to specify the input file. The file must be a valid MS Excel spreadsheet file (∗.XLS). The browse button can be used to launch a file open dialog which makes it easier to specify a file. The file may be located on a remote machine using the notation `\machine\directory\file.xls`.

- **Range**. The range can be left empty in which case the whole first sheet is taken. Otherwise the range can be a single cell (e.g. A1), a block (e.g. B2:J23), or a region within a sheet (e.g. Sheet1!A1:C10). The range can also be a name if the spreadsheet contains named ranges. The browse button will start Excel allowing you to interactively select a range.

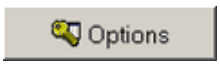
- **Output GAMS Include file (∗.INC)**. If you want to create a GAMS include file, then specify here the destination file. The include file will be an ASCII file that can be read by GAMS using the $include command. If the include file already exists, it will be overwritten.

- **Progress Memo**. This memo field is used to show progress of the application. Also error messages from the database are printed here. This is a read-only field.



- The edit boxes above all have a drop down list which can be used to access quickly file names and queries that have been used earlier (even from a previous session).

- **options button** will pop up a window where you can specify a number of options.



- **help button** will show this help.

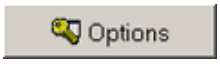
- the **OK button** is pressed the query will be executed and an include file will be generated.

- the **batch button** will give information on how the current extract command can be executed directly from GAMS in a batch environment. The batch call will be displayed and can be copied onto the clipboard. In the IDE press Ctrl-C or choose Edit|Paste to copy the contents of the clipboard to a GAMS text file.

- **close button** will exit the application. The current settings will be saved in an INI file so when you run XLS2GMS again all current settings will be restored.

6.42.8 Options

The **Options** window can be created by pressing the options button:

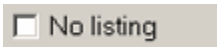


The following options are available in the options window:

- **Quote blanks:** Quote strings if they contain blanks or embedded quotes. If a string does not contain a blank or embedded quotes, it will remain unquoted. If the string contains a single quote the quotes applied will be double quotes and if the string contains already a double quote, the surrounding quotes will be single quotes. (In the special case the string contains both, double quotes are replaced by single quotes). For more information see this example.

- **Mute:** Don't include the extra informational text (such as used query etc.) in the include file.

- **No listing:** Surround the include file by `$offlisting` and `$onlisting` so that the data will not be echoed to the listing file. This can be useful for very large data files, where the listing file would become too large to be handled comfortably.



- **Separator.** This option allows you to set a separator string to be written between cell entries. By default this is a blank. In some cases it can be useful to make this a comma. See example for an example where this is used to import sets. Note: when this option is set to an empty string, the results may not be syntactically correct for GAMS. As it is difficult to see the difference between a single blank and an empty string, the user interface will give some feedback for these cases. When an empty string is used, a warning is written to the include file.

- **Range Seperator.** Multiple ranges can be separated by a range separator symbol. By default this is a semi-colon. When certain non-US locales are used, the semi-colon is a list separator symbol which can be used in multi-area ranges. In case of such a conflict, it is possible to change the range separator symbol.

- **Append.** Append to the output file.

- **Browse Read-Only.** When the Browse Range button is pressed, we launch Excel and try to load the currently specified input file. If this option is checked then the input file is loaded as read-only. If this option is not checked the file is loaded normally, in which case you can change and save it.

The buttons have an obvious functionality:

- **OK button** will accept the changes made.

- **Cancel button** will ignore the changes made, and all option settings will revert to their previous values.

- **Help button** will show this help text.

6.42.9 Batch use

When calling **XLS2GMS** directly from GAMS we want to specify all command and options directly from the command line or from a command file. An example is:

```
C:\tmp>xls2gms "I=c:\My Documents\test.xls" O=test.inc
```

This call will perform its task without user intervention. The batch facility can be used from inside a GAMS model, e.g.:

```
Table c(i,j) 'data from spreadsheet' /
$call =xls2gms I=C:\tmp\test.xls O=C:\tmp\data.inc R=B1:E10
$include C:\tmp\data.inc
/;
```

The $call statement is rather error prone and you will need to spend a little it of time to get the call correct and reliable.

All the possible command line options are listed in command line arguments section. A proper batch call will at least contain the following command line parameters:

1. I=inputfilename
2. O=outputincludefile

If you only specify **I=inputfilename** then the interactive user interface is started with an initial setting of the input file name edit box equal to the name given in the command line argument. Only if an input file and an output file is provided, the call will be considered a batch invocation.

6.42.10 Command-line Arguments

Argument	Description
I=inputfile	This option is required and specifies the name of the .XLS file containing the Access database. If the file contains blanks the name should be surrounded by double quotes. It is advised to use absolute paths, so Access has no confusion what file to open. If only the file name is used without a path, the file is searched in the current directory (this is the project directory when running under the IDE). On a network UNC names can be used, and files from another computer can be accessed, e.g. `"\\hostname\c\my documents\a.xls."` This option is required for batch processing.
O=outputincludefile	option specifies the name of the output file. The format of the output file will be a GAMS include file for a parameter statement. Make sure the directory is writable. UNC names can be used. An output file must be specified for batch operation.
R=range	range is an optional argument. If not specified the whole first sheet is taken. Otherwise the range can be a single cell (e.g. A1), a block (e.g. B2:J23), or a region within a sheet (e.g. Sheet1!A1:C10). To specify a whole sheet use: Sheet2!. The range can also be a name if the spreadsheet contains named ranges. Both global names (e.g. R=rangename) and sheet specific range names (e.g. R=Sheet2!rangename) are recognized. To select multiple ranges in one go, you can specify: R=range1;range2;range3. This is just a short-hand for three separate invocations of xls2gms. A multiple-area range can be specified by R=area1,area2,area3. Before exporting, a new range is created consisting of the union of the areas. This can be used to drop certain rows or columns from a table.
D	Debug. This option can be used for debugging purposes. If specified the import filter will no run minimized but "restored", i.e. as a normal window. In addition the program will not terminate until the user clicks the Close button. This allows you to monitor possible errors during execution of **xls2gms**.
B	If this parameter is specified, strings that have blanks in them will be quoted. If the string is already quoted this step is not performed. If the name contains an embedded single quote, the surrounding quotes will be double quotes. If the name already contains a double quote, the surrounding quotes will be single quotes. If both single and double quotes are present in the string, then all double quotes are replaced by single quotes and the surrounding quotes will be double quotes. By default this option is turned off.
M	Run in modest or mute mode: no additional information, such as version number etc. is written to the listing file.
L	Embed the data in $offlisting, $onlisting. A quick way to reduce the size of the listing file.
@filename @"file name"	Causes the program to read options from a file. If the file name contains blanks, it can be surrounded by double quotes. The option file contains one option per line, in the same syntax as if it were specified on the command line.
N=inifilename	Use a different Inifile than the standard **xls2gms.ini** located in the same directory as the executable **xls2gms.exe**.
A	Append to output files instead of overwriting them.
G="x"	Sets the range separator symbol
S="x"	Sets the output separator symbol

As invocations of **xls2gms** are reasonably expensive (a copy of Excel is started), there is a way to optimize related calls. From version 1.4, xls2gms allows multiple ranges to be read and multiple include files to be written in one swoop. The syntax is best explained by showing an illustrative example:

```
$call =xls2gms I=c:\tmp\x.xls R1=range1 R1=range2 R2=range3 O1=c:\tmp\f1.inc O2=c:\tmp\f2.inc
```

In this example the ranges 'range1' and 'range2' are written to the file 'f1.inc' while the range 'range3' will go to file 'f2.inc'. In general the ranges specified with Rn will be written to the file specified with On. If multiple ranges are specified, they are written sequentially to the output file.

```
$call =xls2gms I=c:\tmp\x.xls R=range1 R=range2 O=c:\tmp\f1.inc
```

In this example the ranges 'range1' and 'range2' are written to 'f1.inc'.

6.42.11 $CALL command

The **$CALL** command in GAMS will execute an external program at compile time. There are two forms:

```
$call externalprogram
```

```
$call =externalprogram
```

The version without the leading '=' calls the external through the command processor (command.com or cmd.exe). The second version with the '=', bypasses the command processor and directly executes the external program. We mention some of the differences:

- Some commands are not external programs but built-in commands of the command processor. Examples are COPY, DIR, DEL, ERASE, CD, MKDIR, MD, REN, TYPE.
- If you want to execute these commands you will need to use the form `$call externalprogram` which uses the command processor. If you want to execute a batch file (.bat or .cmd file) then you will need to use the form `$call externalprogram`.
- If it is important to stop with an appropriate error message if the external program does not exist, only use the form `$call =externalprogram`. The other form is not reliable in this respect. This can lead to surprising results and the situation is often difficult to debug, so in general we would recommend to use the form: `$call =externalprogram`.
- When calling pure Windows programs it is important to call the second form. The first form will not wait until the external Windows program has finished. If it is important to use a command processor in the invocation of a Windows program, use the START command, as in: `$call start /w externalwindowsprogram`. Otherwise, it is preferred to use: `$call =externalwindowsprogram`.

Attention

> In general it is recommended to use the `$call =externalprogram` version for its better error-handling.

When command line arguments need to be passed to the external program, they can be added to the line, separated by blanks:

```
$call externalprogram parameter1 parameter2
```

```
$call =externalprogram parameter1 parameter2
```

The total length of the command line can not exceed 255 characters. If the program name or the parameters contain blanks or quotes you will need to quote them. You can use single or double quotes. In general the following syntax will work:

```
$call '"external program" "parameter 1" "parameter 2"'
```

```
$call ="external program" "parameter 1" "parameter 2"
```

It is noted that the first form needs additional quotes around the whole command line due to bugs in the parsing of the $call in GAMS. The second form work without additional quotes `only if` the = appears outside the double quotes.

6.42.12 Command files

Parameters can be specified in a command file. This is important if the length of the command line exceeds 255 characters, which is a hard limit on the length that GAMS allows for command lines. Instead of specifying a long command line as in:

```
$call =xls2gms I="c:\My Documents\test.xls" O="c:\My Documents\data.inc" R="Sheet2!A1:F15"
```

we can use a command line like:

```
$call =xls2gms @"c:\My Documents\options.txt"
```

The command file `c:\My Documents\options.txt` can look like:

```
I=c:\My Documents\test.xls
O=c:\My Documents\data.inc
R=Sheet2!A1:F15
```

It is possible to write the command file from inside a GAMS model using the $echo command. The following example will illustrate this:

```
$set cmdfile "trnsport.txt"
$echo "I=trnsport.xls" > "%cmdfile%"
$echo "O=trnsport.inc" >> "%cmdfile%"
$call =xls2gms @"%cmdfile%"

$include trnsport.inc
```

Newer versions of GAMS allow:

```
$set cmdfile trnsport.txt
$onEcho > "%cmdfile%"
I=trnsport.xls
O=trnsport.inc
$offEcho
$call =xls2gms @"%cmdfile%"

$include trnsport.inc
```

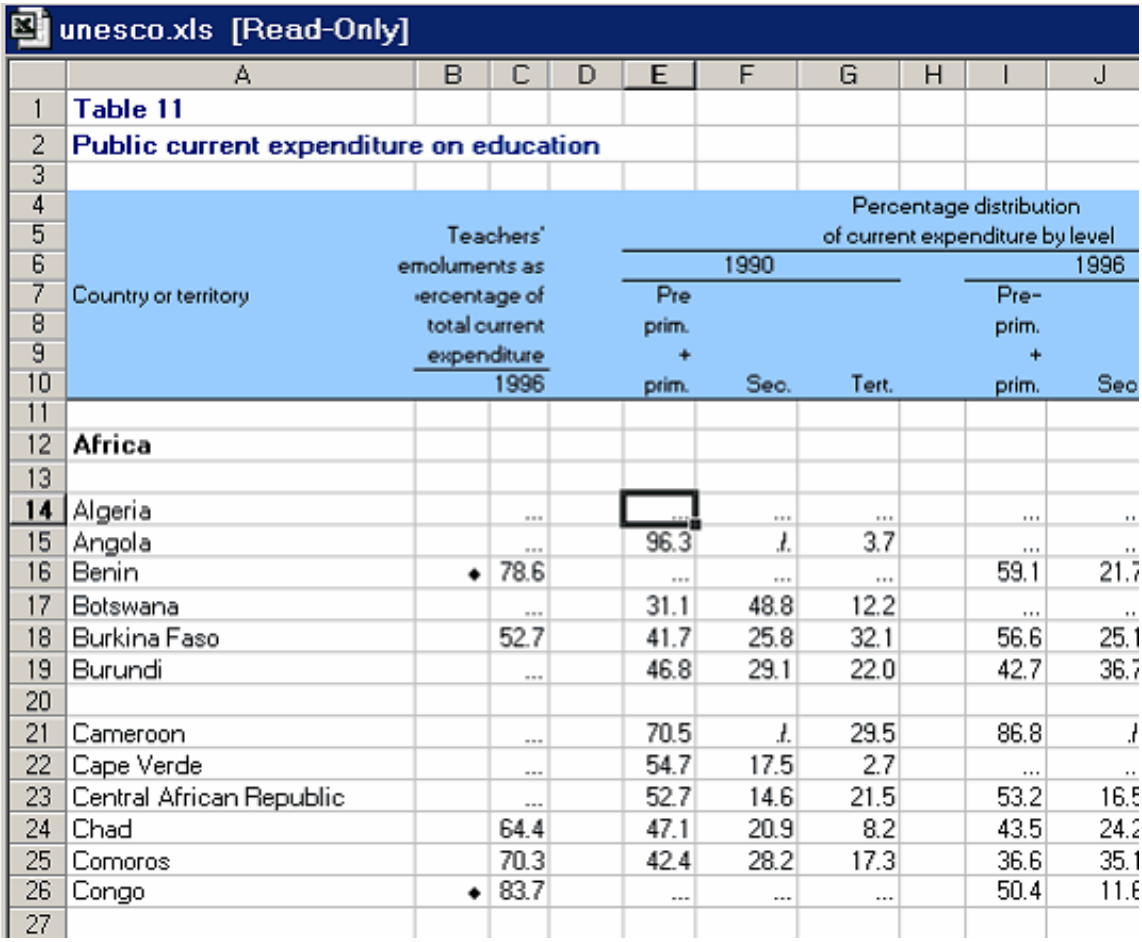
6.42.13 Multiple-area ranges and post-processing

The following fragment is from a spreadsheet from Unesco:

unesco.xls [Read-Only]

	A	B	C	D	E	F	G	H	I	J
1	**Table 11**									
2	**Public current expenditure on education**									
3										
4					Percentage distribution of current expenditure by level					
5		Teachers' emoluments as ercentage of total current expenditure			1990				1996	
6	Country or territory		1996		Pre prim. + prim.	Sec.	Tert.		Pre-prim. + prim.	Sec
7										
8	**Africa**									
9										
10	Algeria		...		...	...	...		...	..
11	Angola		...		96.3	.I.	3.7		...	..
12	Benin	◆	78.6		...	...	...		59.1	21.7
13	Botswana		...		31.1	48.8	12.2		...	..
14	Burkina Faso		52.7		41.7	25.8	32.1		56.6	25.1
15	Burundi		...		46.8	29.1	22.0		42.7	36.7
16										
17	Cameroon		...		70.5	.I.	29.5		86.8	.I
18	Cape Verde		...		54.7	17.5	2.7		...	..
19	Central African Republic		...		52.7	14.6	21.5		53.2	16.5
20	Chad		64.4		47.1	20.9	8.2		43.5	24.2
21	Comoros		70.3		42.4	28.2	17.3		36.6	35.1
22	Congo	◆	83.7		...	...	...		50.4	11.6
23										

Assume we want to extract the 1990 percentage distribution of current expenditure for the countries Algeria through Congo. The range to select is not a contiguous area but consists of several pieces. In Excel we can use the mouse and the Ctrl key to make a multiple selection:



unesco.xls [Read-Only]

	A	B	C	D	E	F	G	H	I	J
1	**Table 11**									
2	**Public current expenditure on education**									
3										
4					Percentage distribution of current expenditure by level					
5		Teachers'								
6		emoluments as			1990				1996	
7	Country or territory	ercentage of			Pre				Pre-	
8		total current			prim.				prim.	
9		expenditure			+				+	
10		1996			prim.	Sec.	Tert.		prim.	Sec
11										
12	**Africa**									
13										
14	Algeria		...		...	...	...		...	..
15	Angola		...		96.3	./.	3.7		...	..
16	Benin	◆	78.6		...	...	...		59.1	21.7
17	Botswana		...		31.1	48.8	12.2		...	..
18	Burkina Faso		52.7		41.7	25.8	32.1		56.6	25.1
19	Burundi		...		46.8	29.1	22.0		42.7	36.7
20										
21	Cameroon		...		70.5	./.	29.5		86.8	./
22	Cape Verde		...		54.7	17.5	2.7		...	..
23	Central African Republic		...		52.7	14.6	21.5		53.2	16.5
24	Chad		64.4		47.1	20.9	8.2		43.5	24.2
25	Comoros		70.3		42.4	28.2	17.3		36.6	35.1
26	Congo	◆	83.7		...	...	...		50.4	11.6
27										

The range is `A10,E10:G10,A14:A19,E14:G19,A21:A26,E21:G26,` where the comma's indicate the range is a multiple-area range. In this case we have six pieces. It is important that Excel understands that the union of the pieces forms a rectangular area. If this is not the case an error will be raised. (You can check this yourself by selecting a multi-area range and copying it to a new sheet: this operation will fail if the areas together don't form a rectangle).

The extracted text file will look like:

```
* -----------------------------------------------------
* XLS2GMS Version 2.3,  March 2004
* Erwin Kalvelagen, GAMS Development Corp.
* -----------------------------------------------------
* Application: Microsoft Excel
* Version:     9.0
* Workbook:    D:\gams projects\xls2gms\ver2.0\unesco.xls
* Sheet:       Sheet1
* Range:       $A$10,$E$10:$G$10,$A$14:$A$19,$E$14:$G$19,$A$21:$A$26,$E$21:$G$26
* -----------------------------------------------------
                          prim. Sec. Tert.
Algeria                    ...   ...  ...
Angola                    96.3  ./.  3.7
Benin                      ...   ...  ...
Botswana                  31.1  48.8 12.2
'Burkina Faso'            41.7  25.8 32.1
Burundi                   46.8  29.1 22
Cameroon                  70.5  ./.  29.5
'Cape Verde'              54.7  17.5 2.7
```

```
'Central African Republic' 52.7  14.6 21.5
Chad                        47.1  20.9 8.2
Comoros                     42.4  28.2 17.3
Congo                       ...   ...  ...
* ---------------------------------------------------
```

This file is not completely suitable for using in a GAMS model. The following edits would need to be made:

1. The header labels should get rid of the trailing dot.
2. Cells with ... should be replaced by blanks.
3. Cells with ./. should be replaced by blanks

In the GAMS distribution a subdirectory **gbin** contains lots of interesting Unix utilities. Some of these are very suited to do string processing on text files, such as **sed** and **awk**. In this case we can use sed with some substitution commands:

Command	Description
`s/prim\./prim /`	Replace "prim." by "prim ". We need to be careful to keep the table alignment correctly, so we replace the dot by a blank instead of just removing the dot. A dot is a special character in sed (it means "any character") so we escape it by specifying "\.".
`s/Sec\./sec /`	Replace "Sec." by "sec ".
`s/Tert\./tert /`	Replace "Tert." by "tert ".
`s/\.\.\./ /g`	Replace "..." by " ". The dots are escaped. We add g to indicate there there may be multiple instances on a line that must be replaced.
`s/\.\/\./ /g`	Replace "./." by " ". Both dots and '/' needs to be escaped.

The complete GAMS formulation can look like:

```
$onText
New version XLS2GMS ver 2.1 can handle
multiple-area ranges.
$offText

Set
   c   'countries'
       / Algeria, Angola, Benin, Botswana, 'Burkina Faso', Burundi, Cameroon
        'Cape Verde', 'Central African Republic', Chad, Comoros, Congo       /
   exp 'percentage distribution of current expenditure'
       / prim, sec, tert /;

$onEcho > commands.txt
I=%system.fp%unesco.xls
R=A10,E10:G10,A14:A19,E14:G19
O=unesco.inc
B
$offEcho

$call =xls2gms @commands.txt

$onEcho > sedcommands.txt
```

```
s/prim\./prim /
s/Sec\./sec /
s/Tert\./tert /
s/\.\.\./   /g
s/\.\/\./   /g
$offEcho

$call sed.exe -f sedcommands.txt unesco.inc > unescox.inc

Table distr(c,exp)
$include unescox.inc
;

display distr;
```

6.43 XLSDUMP

Attention

> `XLSDump` is deprecated (see GAMS 42 XLSDump release notes). Please use Connect agent RawExcelReader instead.

This program will write all worksheets of an Excel workbook to a gdx file. Unlike GDXXRW, the program does not require that Excel is installed. Windows platforms only.

6.43.1 Usage

```
XLSDump infile outfile
```

where

infile

> A valid Excel workbook

outfile

> Optional. The output gdx file. If no output file is specified, the name of the input file will be used to construct the name the output file.

6.43.2 Example

Consider the following working file 'test2.xlsx'.

Converting this workbook using xlsdump:

```
xlsdump test2.xlsx
```

Will generate the file `test2.gdx`. Showing this gdx file in the GAMS IDE:

The parameter VF:

6.44 Multi-Objective Optimization (MOO)

6.44.1 Introduction

In Multi-Objective Optimization (MOO) there is more than one objective function and there is no single optimal solution that simultaneously optimizes all the objective functions. In MOO the concept of optimality is replaced by *Pareto* efficiency or optimality. Pareto efficient (or optimal, nondominated, etc.) solutions are solutions that cannot be improved in one objective without deteriorating in at least one of the other objectives. The mathematical definition of an efficient solution is the following:

Without loss of generality assume that all objective functions f_k with $k = 1, ..., p$ are for minimization. A feasible solution x of a MOO problem is (strongly) efficient if there is no feasible solution x' such as $f_k(x') \leq; f_k(x)$ for $k = 1, ..., p$ with at least one strict inequality. A feasible solution x is weakly efficient if there is no feasible solution x' such as $f_k(x') < f_k(x)$ for $k = 1, ..., p$.

In case of a non-convex feasible objective and decision space (e.g. MIP), the set of efficient solutions can be further partitioned into supported and non-supported efficient solutions. Supported efficient solutions are optimal solutions of the weighted sum single objective problem for any non-negative weights (corner solutions of the feasible space). See the following graph that illustrates the difference between supported and non-supported efficient solutions:

A simple MOO method is *lexicographic optimization*. Lexicographic optimization presumes that the decision-maker has *lexicographic preferences*, ranking the efficient solutions to a *lexicographic order* of the objective functions. A lexicographic minimization problem can be written as:

$lex\ min\ f_1(x), f_2(x), ..., f_p(x)$
s.t.
$x \in S$

where x is the vector of decision variables, S is the feasible region, and $f_1(x), f_2(x), ..., f_p(x)$ are the p objective functions ordered from the most to the least important. A lexicographic minimization problem with p objective functions can be solved using a sequence of p single-objective optimization problems as follows:

For $i = 1, ..., p$ **do**

- Solve single-objective optimization problem:

  $min\ z_i$
  s.t.
  $f_k(x) = z_k\ \forall k = 1, ..., i-1$
  $x \in S$

  where z_k are the objective function variables
- Fix the value of objective function variable z_i for the following iterations

End for

The MOO methods provided by this LibInclude file use lexicographic optimization to calculate a *payoff* table to obtain e.g. ranges for the objectives. The $p \times p$ payoff table contains the results of p lexicographic optimizations with orders $\{f_1(x), f_2(x), ..., f_p(x)\}, \{f_2(x), f_3(x), ..., f_p(x), f_1(x)\}, ..., \{f_p(x), f_1(x), ..., f_{p-1}(x)\}$.

This LibInclude file provides a collection of methods to generate Pareto efficient solutions for MOO models. All implemented methods are so-called a-posteriori (or generation) methods for generating Pareto efficient solutions. After the Pareto efficient set or a representative subset of the Pareto efficient set has been generated, it is returned to the user, who then analyses the solutions and selects the most preferred among the alternatives.

6.44.2 Usage

LibInclude file `moo.gms` provides methods for multi-objective optimization in GAMS and can be used as follows:

```
$libinclude moo method modelName modelType objectiveSet direction objectiveVar points paretoPoints [
```

Where:

Argument	Description
`method`	Specify the name of the method to use. Available methods are `EpsConstraint`, `RandomWeighting`, and `Sandwiching`. See Methods for details.
`modelName`	Specify the name of the multi-objective optimization model that should be solved.
`modelType`	Specify the corresponding model type, e.g. `LP`.
`objectiveSet`	Specify the set of objectives.
`direction`	One-dimensional parameter that contains the optimization direction for each `objectiveSet` element - min (-1) or max (+1).
`objectiveVar`	Specify the objective function variable. Compared to the traditional scalar objective function variable with a single objective optimization problem in GAMS, this objective function variable is one-dimensional.
`points`	Specify the set of pareto points. The number of members in the set defines the maximum number of returned pareto points. Therefore, make sure to define the set large enough.
`paretoPoints`	Specify the two-dimensional (`points`,`objectiveSet`) parameter that will be used to save the objective values of pareto points.
`optList`	Optional list of `-optkey=optval` option pairs, e.g. `-savepoint=1 -optfile_main=99`. See available options below.

Method	Option	Description
All:	`debug`	Set to 1 to activate debugging. This will create additional information being written to the log as well as creating files to support debugging, e.g. if savepoint is activated a file `savepoint_iteration_map.csv` is written to the `savepoint_dir` that contains the mapping of savepoint files to the corresponding iteration. Default: 0.
	`optfile_init`	Solver option file for initial solves. In particular, solves to calculate the payoff table. Default: 0.
	`optfile_main`	Solver option file for main solves, i.e. the main part of the methods. Default: 0.
	`plot`	Set to 1 to activate plots for multi objective problems with two or three objective functions, e.g. a plot of the pareto points. Python package `matplotlib` needs to be installed. Default: 0.

Method	Option	Description
	plot_dir	Set the directory where plots should be saved. Per default the GAMS working directory will be used.
	pre	Prefix for names of GAMS symbols used in moo.gms. Allows to make GAMS symbol names unique and avoids problems with redefined symbols. The default is MOO_<MOO___COUNTER>, where MOO___COUNTER is a (global) compile-time variable that counts the MOO runs in the GAMS program, so each run has unique GAMS symbol names. This option allows to change the default.
	savepoint	If set to 1, a point format GDX file that contains information on the solution point will be saved for each pareto point. Default is 0, where no solution points will be saved.
	savepoint_dir	Set the directory where save point files should be saved. Per default the GAMS working directory will be used.
	savepoint_filename	Set the (base) name of the save point files. The file name gets extended by the points set to generate unique file names as follows: <savepoint_filename>_<points>.gdx. Per default this option is set to the name of the method. If an empty string is provided the file name matches exactly with the points set: <points>.gdx.
	solver	Default solver for all model types that the solver is capable to process.
EpsConstraint:	gridpoints	Specify the number of grid points per objective function used as a constraint (default: 10). As an example, consider a model with three objective functions, the Epsilon Constraint method will optimize one objective function using the other two as contraints. With 10 grid points (default) the maximum number of runs will therefore be 10x10. In general, increasing the number of grid points increases the density of the approximation.
RandomWeighting:	iterations	Terminating condition in terms of iterations. Default: inf.
	seed	Set the seed for the pseudo random number generator that generates the weights. Set to None to set the seed to a random number. Default: 3141.
	time	Terminating condition in terms of time in seconds. Default: 3600.

Method	Option	Description
Sandwiching:	`gap`	Terminating condition in terms of a gap (default: 0.01). The gap is calculated by the current maximum distance between the inner and outer approximation divided by the initial maximum distance.
	`time`	Terminating condition in terms of time in seconds. Default: 3600.

6.44.3 Methods

This section gives an introduction and details on the implemented methods for MOO. The implemented methods are so-called a-posteriori (or generation) methods for generating Pareto efficient solutions. After the Pareto efficient set or a representative subset of the Pareto efficient set has been generated, it is returned to the user, who then analyses the solutions and selects the most preferred among the alternatives.

6.44.3.1 Augmented Epsilon Constraint [EpsConstraint]

The Epsilon Constraint method optimizes one of the objective functions using the other objective functions as constraints. By parametrical variation in the Right-Hand Side (RHS) of the constrained objective functions the efficient solutions of the problem are obtained. The implemented Augmented Epsilon Constraint method is based on:

> Mavrotas, G., Effective implementation of the eps-constraint method in Multi-Objective Mathematical Programming problems. Applied Mathematics and Computation 213, 2 (2009), 455-465.

> Mavrotas, G., and Florios, K., An improved version of the augmented eps-constraint method (AUGMECON2) for finding the exact Pareto set in Multi-Objective Integer Programming problems. Applied Mathematics and Computation 219, 18 (2013), 9652-9669

The Augmented Epsilon Constraint method is a variation of the Epsilon Constraint method that produces only efficient solutions (no weakly efficient solutions) and also avoids redundant iterations as it can perform early exit from the relevant loops (that lead to infeasible solutions) and exploits information from slack variables in every iteration.

Compared to the Random Weighting method the runs are exploited more efficiently and almost every run produces a different efficient solution, thus obtaining a better representation of the efficient set. Allowed model types: **LP**, **MIP** as well as corresponding **Non Linear** cases. The method is able to find the exact Pareto set for integer programming problems by appropriately tuning the number of grid points. The Augmented Epsilon Constraint method is able to produce supported and unsupported efficient solutions.

Implementation details:

- The Augmented Epsilon Constraint method optimizes one of the objective functions using the other objective functions as constraints, incorporating the constraint part of the model as follows:

  $max(f_1(x) + eps \times (s_2/r_2 + 10^{-1} \times s_3/r_3 + ... + 10^{-(p-2)} \times s_p/r_p))$

  s.t.
  $f_2(x) - s_2 = e_2$
  $f_3(x) - s_3 = e_3$
  ...
  $f_p(x) - s_p = e_p$
  $x \in S$ and $s_k \in \mathbb{R}^+$

  where x is the vector of decision variables, S is the feasible region, and $f_1(x), f_2(x), ..., f_p(x)$ are the p objective functions. e_k are the parameters for the RHS for the specific iteration drawn from the grid points of the objective functions k ($k = 2, 3, ..., p$). s_k are slack or surplus variables, r_k is the range of the k-th objective function (calculated from the payoff table), and $eps \in [10^{-6}, 10^{-3}]$.

- The method is implemented as follows: From the payoff table we obtain the range of the $p-1$ objective functions that are used as constraints. The range of the k-th objective function is divided into q_k equal intervals using $(q_k - 1)$ intermediate equidistant grid points. Thus, we have in total $(q_k + 1)$ **grid points** that are used to vary parametrically the RHS (e_k) of the k-th objective. The total number of runs becomes $(q_2 + 1) \times (q_3 + 1) \times ... \times (q_p + 1)$.

  The discretization step for objective function k:

  $step_k = r_k/q_k$

  The RHS of the corresponding constraint in the i-th iteration in the specific objective function will be:

  $e_{ki} = fmin_k + i_k \times step_k$

  where $fmin_k$ is the minimum obtained from the payoff table and i_k the counter for the specific objective function.

- At each iteration we check the surplus variable that corresponds to the innermost objective function (in this case $p = 2$) and calculate the bypass coefficient as:

  $b = floor(s_2/step_2)$

  Where $floor()$ is a function that always rounds down and returns the largest integer less than or equal to a given number. When the surplus variable s_2 is larger than the $step_2$ it is implied that in the next iteration the same solution will be obtained with the surplus variable being $s_2 - step_2$. Therefore, the bypass coefficient actually indicates how many consecutive iterations can be bypassed.

- Early exit from the loop when the problem becomes infeasible for some combination of e_k. The bounding strategy for each one of the objective functions starts from the more relaxed formulation (lower bound for maximization objective function or upper bound for a minimization) and move to the most strict. If we arrive to an infeasible solution there is no need to perform the remaining runs of the loop, as the problem will become even stricter.

6.44.3.2 Random Weighting [RandomWeighting]

With the Random Weighting method, the weighted sum of the objective functions is optimized. By randomly varying the weights different pareto points are obtained.

Allowed model types: All model types with an objective variable, in particular **LP**, **MIP**, **NLP**, and **MINLP**. Note that the Random Weighting method may spend a lot of runs that are redundant in the sense that there can be a lot of combinations of weights that result in the same efficient point. Also note that the Random Weighting method cannot produce unsupported efficient solutions.

Implementation details:

- For Random Weighting the scaling of the objective functions has a strong influence on the obtained results. Therefore, the objective functions are scaled between 0 and 1 using the objective function ranges obtained from the payoff table.

6.44.3.3 Sandwiching

Sandwiching algorithms are used to approximate the pareto front by sandwiching the nondominated set between an inner and outer approximation. Sandwiching algorithms iteratively improve both the Inner and Outer approximation of the Pareto Set (IPS and OPS) to minimize the distance between the approximations. The implemented Sandwiching method is based on:

> Koenen, M., Balvert, M., and Fleuren, HA. (2023). A Renewed Take on Weighted Sum in Sandwich Algorithms: Modification of the Criterion Space. (Center Discussion Paper; Vol. 2023-012). CentER, Center for Economic Research.

An advantage of the Sandwiching method is that the distance between the IPS and OPS provides an upper bound on the approximation error which can also be used as a termination criterion for the algorithm. Please be aware that for more than two objectives the distance using the definition of the Koenen et al. (2023) can sometimes increase over the iterations. Allowed model type: **LP**.

Implementation details:

- The Sandwiching algorithm of Koenen et al. (2023) is based on a weighted sum approach, i.e. the weighted sum of the objective functions is optimized. By varying the weights at each iteration the efficient points are obtained. The algorithm selects weights derived from the IPS that have the potential to reduce the approximation error the most.
- The IPS is a convex hull encompassed in the objective space formed by a set of efficient points and the OPS is formed by supporting halfspaces of those efficient points.
- The basic procedure of the algorithm can be described as follows:
  1. **Initialize the IPS and OPS** using the pseudo nadir point and so called anchor points. The pseudo nadir point and anchor points are obtained from the payoff table and also define the initial bounding box of the objective space.
  2. **Expand the IPS based on its distance to the OPS**. For each relevant hyperplane of the convex hull of the IPS determine the distance to the OPS and find the hyperplane for which the current distance from the IPS to the OPS is maximal. The normal of this hyperplane will be used as weights in the current iteration. If a new efficient point is found, update the IPS and OPS, otherwise set the distance of the current hyperplane to zero. The procedure is repeated until a stopping criterion is met (see options `gap` or `time`).
- The calculation of the distance between IPS and OPS is based on SUB (see Koenen et al. (2023)).
- The scaling of the objective functions has a strong influence on the obtained results. Therefore, the objective functions are scaled between 0 and 1 using the objective function ranges obtained from the payoff table.

6.44.4 Examples

6.44.4.1 Example 1: Solve scalable multi-objective knapsack model

The following example demonstrates how to solve a multi-objective knapsack model (MIP) using the `moo` LibInclude:

```
$if not set NBITM $set NBITM 50
$if not set NBDIM $set NBDIM  2
$if not set NBOBJ $set NBOBJ  2

Set
   i 'items'                / i1*i%NBITM% /
   j 'weight dimensions'    / j1*j%NBDIM% /
   k 'value dimensions'     / k1*k%NBOBJ% /
   p 'pareto points'        / point1*point1000 /;

Parameter
   a(i,j)           'weights of item i'
   c(i,k)           'values of item i'
   b(j)             'knapsack capacity for weight j'
   dir(k)           'direction of the objective functions 1 for max and -1 for min' / #k 1/
   pareto_obj(p,k)  'objective values of the pareto points'
;
a(i,j) = UniformInt(1,100);
c(i,k) = UniformInt(1,100);
b(j)   = UniformInt(1,100) * %NBITM%/4;

Variable
   Z(k) 'objective variables'
   X(i) 'decision variables';

Binary Variable X;

Equation
   objfun(k) 'objective functions'
   con(j)    'capacity constraints';

objfun(k).. sum(i, c(i,k)*X(i)) =e= Z(k);

con(j)..    sum(i, a(i,j)*X(i)) =l= b(j);

Model example / all /;

$libinclude moo EpsConstraint example MIP k dir Z p pareto_obj -gridpoints=20

display pareto_obj;
```

The example is easily scalable by allowing to set the number of items, the number of weight dimensions and the number of objectives through double dash parameters. This example is also part of the GAMS Data Utilities Library, see model **[moo01]** for reference. The model is solved using `EpsConstraint` method with 20 `gridpoints`. The objective function values of the pareto points found are saved in parameter `pareto_obj`:

```
----    618 PARAMETER pareto_obj  objective values of the pareto points

                k1          k2

point1     953.000    1076.000
point2     956.000    1062.000
point3     972.000    1052.000
point4    1005.000    1037.000
point5    1030.000    1024.000
point6    1049.000    1004.000
```

```
point7     1063.000     990.000
point8     1088.000     955.000
point9     1091.000     946.000
point10    1105.000     932.000
point11    1121.000     876.000
point12    1130.000     827.000
point13    1137.000     783.000
```

If `plot` is set to 1 and Python package `matplotlib` is installed, a plot of the pareto points is generated:

Setting `--NBOBJ=3` with model [**moo01**] allows to solve a knapsack problem with three objectives. These are the resulting pareto points:

6.44.4.2 Example 2: Solve multi-objective power generation model

The following example demonstrates how to solve a multi-objective power generation model (LP) using the `moo` LibInclude:

```
$inlineCom [ ]

$if not set NBOBJ $set NBOBJ  2
$if not set METHOD $set METHOD Sandwiching

Set
   p       'power generation units' / Lignite, Oil, Gas, RES /
   i       'load areas'             / base, middle, peak     /
   pi(p,i) 'availability of unit for load types' / Lignite.(base,middle)
                                                   Oil.(middle,peak), Gas.set.i
                                                   RES.(base, peak)            /
   es(p)   'endogenous sources'  / Lignite, RES /
   k       'objective functions' / cost, CO2emission, endogenous /
   points  'pareto points' / point1*point1000 /;

$set min -1
$set max +1

Parameter dir(k) 'direction of the objective functions 1 for max and -1 for min'
                 / cost %min%, CO2emission %min%, endogenous %max% /
          pareto_obj(points,k) 'objective values of the pareto points';

Set pheader / capacity, cost, CO2emission /;

Table pdata(pheader,p)
                          Lignite    Oil    Gas    RES
   capacity [GWh]           61000  25000  42000  20000
   cost [$/MWh]                30     75     60     90
   CO2emission [t/MWh]       1.44   0.72   0.45      0;

Parameter
   ad        'annual demand in GWh'          / 64000 /
   df(i)     'demand fraction for load type' / base 0.6, middle 0.3, peak 0.1 /
   demand(i) 'demand for load type in GWh';

demand(i) = ad*df(i);
```

```
Variable
   z(k)      'objective function variables';

Positive Variable
   x(p,i)    'production level of unit in load area in GWh';

Equation
   objcost   'objective for minimizing cost in K$'
   objco2    'objective for minimizing CO2 emissions in Kt'
   objes     'objective for maximizing endogenous sources in GWh'
   defcap(p) 'capacity constraint'
   defdem(i) 'demand satisfaction';

objcost..   sum(pi(p,i), pdata('cost',p)*x(pi)) =e= z('cost');

objco2..    sum(pi(p,i), pdata('CO2emission',p)*x(pi)) =e= z('CO2emission');

objes..     sum(pi(es,i), x(pi)) =e= z('endogenous');

defcap(p).. sum(pi(p,i), x(pi))  =l= pdata('capacity',p);

defdem(i).. sum(pi(p,i), x(pi))  =g= demand(i);

Model example / all /;

Set kk(k) 'active objective functions';
kk(k) = yes;
$if %NBOBJ%==2 kk('endogenous') = no;

$libinclude moo %METHOD% example LP kk dir z points pareto_obj -iterations=20 -gridpoints=5 -savepoi
execute 'gdxmerge savepoints%system.DirSep%*.gdx > %system.NullFile%';

display pareto_obj;
```

This example is also part of the GAMS Data Utilities Library, see model **[moo02]** for reference. The model is solved using the `Sandwiching` method with a `gap` of 0.01 (default). The objective function values of the pareto points found are saved in parameter `pareto_obj`:

```
----   1164 PARAMETER pareto_obj  objective values of the pareto points

              cost  CO2emissi~

point1 2112000.000   85824.000
point2 3180000.000   50580.000
point3 4470000.000   20340.000
```

In addition, `savepoint` is activated and thus, for each pareto point a point format GDX file that contains information on the solution point will be written. Using `gdxmerge` all savepoint GDX files can be merged into a single GDX file.

If `plot` is set to 1 and Python package `matplotlib` is installed, a plot of the pareto front is generated:

The plot shows the pareto efficient extreme points as well as the lines connecting these points (the IPS). An animation of the algorithm shows how the IPS and OPS are updated at each iteration:

Starting with the initial points from the payoff table, the IPS and the OPS are constructed (iteration 0). In iteration 1, a new point is found and the IPS and OPS are updated. In the last two iterations no new

point is found but the OPS is updated so the upper bound on the approximation error (distance between the IPS and OPS) decreases and the gap of 0.01 is reached.

Setting --NBOBJ=3 with model [**moo02**] allows to solve a power generation problem with three objectives. This is the resulting pareto front:

6.45 Model Instances (pyEmbMI)

Libinclude file pyembpy.gms for the GAMS Python API class GAMSModelInstance. An instance of this class provides access to a model instance that can be modified and resolved without regenerating the model over and over.

Usage:

```
$libinclude pyEmbMI myPyMI 'solve statement' [optlist] updateList
```

Where:

Argument	Description
myPyMI	Name of the GAMSModelInstance Python identifier.
solve statement	Part of the solve statment that gives model name, solver type and optimization direction / variable, e.g. transport minimize z us lp.
optlist	Optional list of -optkey=optval option pairs, e.g. -reslim=100 -all_model_types=cplex that build update a GamsOption object used in the instantiate call. See **gams.control.options.GamsOptions Class Reference** for details.
updateList	List of update tuples (see details below): Variable update tuples: gmsModelVar.Lower|Upper|Fixed|Primal|Dual.gmsPar.Zero|BaseCa Parameter update pairs: gmsModelPar.Zero|BaseCase|Accumulate

The update tuples contain instructions how to deal with updating variable/equation attributes (mostly bounds) and parameters: If you want to change a parameter the entries holds the syntax: x.action where x is the GAMS name of the parameter and action is the update type of the parameter. This identifies how a parameter is updated if there is no data corresponding to the set intersection defining the parameter.

- parametername.Zero will set all parameter values that are not explicitly set to zero.
- parametername.BaseCase will set all parameter values that are not explicitly set to the value from the first model instantiation.
- parametername.Accumulate will keep all parameter values that are not explicitly set to their last known value.

To update an attribute of a variable/equation 4 things need to be considered:

- What variable or equation to work with

- What attribute to change (e.g. upper and lower corresponding bounds)
- What parameter contains the new values
- What to do with zeros which is the `Zero`, `BaseCase`, and `Accumulate` from above. This is expressed in the `tuple variableName.attribute.parameterWithNewValues.zeroKeyword`. For example `x.Upper.xUp.BaseCase` would reset upper bounds for `x` to the values in the parameter `xup` and set the values that are not explicitly set to their last known value.

Hence a call to this include might look like this:

```
$libinclude pyEmbMI miTransport 'transport minimizing z using lp' -all_model_types=cplex  x.Upper.xU
```

Embedded Code Model Instance Example

The GAMS Embedded Code facility allows to execute foreign code (e.g. Python) while GAMS runs and to exchange data with GAMS without any disk access (e.g. GDX).

In this example we combine the power of the embedded code facility with the GAMS Python OO-API class `GAMSModelInstance`. An instance of this class provides access to a model instance that can be modified and resolved without regenerating the model over and over.

Here we generate the model instance once by using the libinclude `pyEmbMI`. The arguments to this call provide all necessary information to instantiate an instance of a `GAMSModelInstance`. In particular we provide the relevant part of the solve statement as well as a list of modifiers. These are the parameters in the model that are subject to change. In addition we can provide some options belonging to a GAMS/Python OO-API class `GAMSOptions` via the `-key=value` pairs. See more about the use of `GAMSModelInstance` and `GAMSOptions` in the GAMS/Python OO-API.

Background:

A traditional GAMS implemenation of a scenario loop looks like this:

```
loop(ScenariosToRun,
   a(i) = newsupply(ScenariosToRun,i);
   b(j) = newdemand(ScenariosToRun,j);
   solve transport using lp minimizing z;
   resultantx(ScenariosToRun,i,j) = x.l(i,j)
);
```

With the embedded code/GAMSModelInstance solution this loop looks as follows:

```
$libInclude pyEmbMI tMI 'transport us lp min z' -all_model_types=cplex a.Zero b.Zero
loop(ScenariosToRun,
   a(i) = newsupply(ScenariosToRun,i);
   b(j) = newdemand(ScenariosToRun,j);
   continueEmbeddedCode:
   gams.db['a'].copy_symbol(tMI.sync_db['a'])
   gams.db['b'].copy_symbol(tMI.sync_db['b'])
   tMI.solve()
   tMI.sync_db['x'].copy_symbol(gams.db['x'])
   pauseEmbeddedCode x
   resultantx(ScenariosToRun,i,j) = x.l(i,j);
);
```

With a little helper function `solveMI` (see below) this code becomes even more similar:

```
$libInclude pyEmbMI tMI 'transport us lp min z' -all_model_types=cplex a.Zero b.Zero
loop(ScenariosToRun,
   a(i) = newsupply(ScenariosToRun,i);
   b(j) = newdemand(ScenariosToRun,j);
   continueEmbeddedCode:
   solveMI(tMI,['a','b'],['x'])
   pauseEmbeddedCode x
   resultantx(ScenariosToRun,i,j) = x.l(i,j);
);
```

In contrast to GUSS/Scenario Solver here we implement the loop logic in GAMS and execute in the loop body the solve method of the `GAMSModelInstance` class inside the embedded Python code. Rather than using the `gams.get|set` method of the embedded code facility we use `GAMSDatabase.copy_symbol` to move data between GAMS (`gams.db`) and the `GAMSModelInstance.sync_db`.

Note

Even though we don't exercise the ability in this example, the combination of `GAMSModelInstance` and embedded code provides a way of defining the scenario n+1 based on the result (primal and dual) of the nth scenario. This is not possible in GUSS/Scenario Solver.

Example:

```
* GMSPYTHONLIB gets automatically set to use the internal Python installation in sysdir/GMSPython.
$if not setEnv GMSPYTHONLIB $abort.noError Embedded code Python not ready to be used
$log --- Using Python library %sysEnv.GMSPYTHONLIB%

Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

Parameter
   a(i) 'capacity of plant i in cases'
        / seattle    350
          san-diego  600 /

   b(j) 'demand at market j in cases'
        / new-york   325
          chicago    300
          topeka     275 /;

Table d(i,j) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;

Scalar f 'freight in dollars per case per thousand miles' / 90 /;

Parameter c(i,j) 'transport cost in thousands of dollars per case';
c(i,j) = f*d(i,j)/1000;

Variable
   x(i,j) 'shipment quantities in cases'
```

```
   z      'total transportation costs in thousands of dollars';

Positive Variable x;

Equation
   cost      'define objective function'
   supply(i) 'observe supply limit at plant i'
   demand(j) 'satisfy demand at market j';

cost..      z =e= sum((i,j), c(i,j)*x(i,j));

supply(i).. sum(j, x(i,j)) =l= a(i);

demand(j).. sum(i, x(i,j)) =g= b(j);

Model transport / all /;

Set s 'scenarios to run' / base, run1, run2 /;

Table newsupply(s,i) 'updater for a (capacity)'
         seattle  san-diego
   base      350        600
   run1      300        650
   run2      400        550;

Table newdemand(s,j) 'updater for b (demand)'
         new-york  chicago  topeka
   base       325      300     275
   run1       325      300     275
   run2       350      300     250;

$set solverlog
$if set useSolverLog $set solverlog output=sys.stdout
embeddedCode Python:
def solveMI(mi, symIn=[], symOut=[]):
  for sym in symIn:
    gams.db[sym].copy_symbol(mi.sync_db[sym])
  mi.solve(%solverlog%)
  for sym in symOut:
    try:
      gams.db[sym].clear() # Explicitly clear the symbol to ensure setting "writtenTo" flag for sym
      mi.sync_db[sym].copy_symbol(gams.db[sym])
    except:
      pass
pauseEmbeddedCode
abort$execerror 'Python error. Check the log';

$libInclude pyEmbMI tMI 'transport us lp min z' -all_model_types=cplex a.Zero b.Zero

Parameter repX(s,i,j) 'collector for level of x';

loop(s,
   a(i) = newsupply(s,i);
   b(j) = newdemand(s,j);
   continueEmbeddedCode:
   solveMI(tMI,['a','b'],['x'])
   pauseEmbeddedCode x
   repX(s,i,j) = x.l(i,j);
```

```
);

option  repX:0:1:2;
display repX;

Set error(s) 'empty solution';
error(s) = sum((i,j), repX(s,i,j)) = 0;
abort$card(error) 'Missing solution for some scenarios', error;
```

The complete example is also part of the GAMS Model Library, see model [**embmiex1**] for reference.

6.46 Sorting (rank)

Libinclude file `rank.gms` for ranking one-dimensional numeric data.

Developed by Thomas F. Rutherford, Department of Economics, University of Colorado

Usage:

```
$libInclude rank v s r [p]
```

Where:

Type	Argument	Description
Input:	*v(s)*	Array of values to be ranked.
	s	A one-dimensional set, the domain of array *v*.
Output:	*r(s)*	Rank order of element *v(s)*, an integer between 1 and card(s), ranking from smallest to largest.
Optional: (input and output)	*p(*)*	On input this vector specifies percentile levels to be computed. On output, it returns the linearly interpolated percentiles.

Note:

- `rank` only works for numeric data. You cannot sort sets.
- The first invocation must be outside of a loop or if block. This routine may be used within a `loop` or `if` block only if it is first initialized with blank invocations (`"$libInclude rank"` in a context where set and parameter declarations are permitted (See Example 3).
- The following names are used within these routines and may not be used in the calling program:

  `rank_tmp` `rank_u` `rank_p`

- This routine returns rank values and does not return `sorted vectors`, however rank values are easily used to produce a sorted array. This can be done using computed "leads" and "lags" in GAMS' ordered set syntax, as illustrated in examples 1 and 3 below.

6.46.1 Examples

6.46.1.1 Example 1: Rank a vector, and display the data in sorted order

```
Set
   i 'Set on which random data are defined'   / a, b, d, c, e, f /
   k 'Ordered set for displaying sorted data' / 1*6 /;

Parameter
   x(i)   'Random data to be sorted'
   r(i)   'Rank values'
   s(k,i) 'Sorted data';

x(i) = uniform(0,1);

$libInclude rank x i r
display x;

* Generate a sorted list using the ordered set k.

* This assignment statement illustrates how the rank orders
* can be used to sort output for display in proper order.  This
* statement uses GAMS support for computed "leads" and "lags"
* on the ordered set k.  The loop is used to improve execution
* speed for larger dimensional sets:

loop(k$sameas(k,"1"),
   s(k+(r(i)-1),i) = x(i);
);

option  s:3:0:1;
display s;
```

This example is also part of the GAMS Data Utilities Library, see model **[rank01]** for reference. It writes the following lines to ex1.lst:

```
----     11 PARAMETER x  Random data to be sorted

a 0.172,    b 0.843,    d 0.550,    c 0.301,    e 0.292,    f 0.224


----     75 PARAMETER s  Sorted data

1.a 0.172
2.f 0.224
3.e 0.292
4.c 0.301
5.d 0.550
6.b 0.843
```

The `rank` libinclude can be used to also sort variable levels as the following example shows:

```
Set
   i 'Set on which random data are defined'   / a, b, d, c, e, f /
```

```
   k 'Ordered set for displaying sorted data' / 1*6 /;

Variable
   x(i)   'Random data to be sorted'
Parameter
   r(i)   'Rank values'
   s(k,i) 'Sorted data';

x.l(i) = uniform(0,1);

$onDotL
$libInclude rank x i r
display x.l;

loop(k$sameas(k,"1"),
   s(k+(r(i)-1),i) = x(i);
);

option  s:3:0:1;
display s;
```

Before the "`$libInclude rank`" the dollar control option $onDotL needs to be activated.

6.46.1.2 Example 2: Generate percentiles for a random vector

```
Set
   i 'Set on which random data are defined' / a, b, d, c, e /
   p 'Percentiles (all of them)'            / 0*100 /;

Parameter x(i) 'Random data to be sorted';

* Generate the random data on set i:
x(i) = uniform(0,1);
display x;

Parameter
   r(i)   'Rank values'
   pct(*) 'Percentiles to be computed' / 20 20.0, median 50.0, 75 75.0 /;

* Generate ranks and compute the specified percentiles (Note that
* the rank array, r, is required, even if the values are not used.)

$libInclude rank x i r pct

* Display three percentiles:
display pct;
```

This example is also part of the GAMS Data Utilities Library, see model [**rank02**] for reference. The random data are displayed as follows in the listing file:

```
----     11 PARAMETER x  Random data to be sorted

a 0.172,    b 0.843,    d 0.550,    c 0.301,    e 0.292
```

The interpolated percentiles are computed as follows:

```
----    103 PARAMETER pct  Percentiles

20     0.268,    75     0.550,    median 0.301
```

The following code evaluates a full set of percentiles, from 1 to 100. The GAMS special value of EPS is used to represent zero in the percentile calculation. (Percentiles between zero and one are not permitted to avoid misunderstandings about how percentiles are scaled.) The code makes use of Tom Rutherfords plot libInclude available at http://www.mpsge.org/gnuplot/index.html

```
pct(p) = (ord(p) - 1) + eps;
pct("median") = 0;
display pct;

$libInclude rank x i r pct
display pct;

* Plot the results using GNUPLOT:

Set pl(p) / 20, 40, 60, 80, 100 /;
$setGlobal domain p
$setGlobal labels pl
$libInclude plot pct
```

This is the generated plot:

6.46.1.3 Example 3: Use RANK to report multisectoral Monte Carlo results

One of the most perplexing challenges in economic modeling with GAMS is to present multisectoral results in an easily interpreted format. One simple idea is to present sectoral results in a sorted sequence to make it easier to identify the most seriously affected sectors. The presentation of results in a multisectoral model is made even more challenging when model results are generated for a randomized set of scenarios. A summary of Monte Carlo results involves reporting both mean results and their sensitivity. One means of characterizing the sensitivity of model results is to report functions of the sample distribution such as the upper and lower quartiles.

This example illustrates how `rank` can be used to help report results from the Monte Carlo analysis of a multisectoral model.

The complete model used as example here is part of the GAMS Data Utilities Library, see model **[rank03]** for reference. The following extract of that model generates the result we want to look at.

```
* Determine ranking of sectors by mean impact:
mean(i) = qvalue(i,"mean");
$libInclude rank mean i meanrank

* The following statement creates a tuple matching the ordered
* set, ki, to the set of sectors, i.  In this tuple, the sequence of
* assignments corresponds to increasing mean impacts:
imap(ki+(meanrank(i)-ord(ki)),i) = yes;

* Evaluate quartiles of sectoral impacts for each sector:
```

```
loop(i,
   x(s) = v(s,i);

*  Load quartile with the perctiles to be
*  evaluated (25th, 50th and 75th):
   quartile(qtl) = qv(qtl);

$  libInclude rank x s r quartile

*  Save the quartile values:
   qvalue(i,qtl) = quartile(qtl);
);
display qvalue;

Parameter results(ki,i,*) 'Summary of impacts (sorted)';
results(ki,i,"mean")$imap(ki,i) = mean(i);
results(ki,i,qtl)$imap(ki,i)    = qvalue(i,qtl);
display results;
```

The program produces the following display output:

```
              q25         q50         q75        mean

1 .FOO     -13.767     -12.110     -10.671     -12.259
2 .MWO     -13.496     -11.982     -10.512     -12.035
3 .SCS     -12.244     -10.418      -8.738     -10.588
4 .CLO      -8.763      -7.407      -6.158      -7.480
5 .ADM      -7.865      -5.601      -3.470      -5.812
6 .CNM      -6.437      -5.320      -4.404      -5.461
7 .OTH      -5.732      -4.781      -4.012      -4.892
8 .PIP      -3.142      -1.930      -0.836      -2.145
9 .OIN      -2.124      -1.339      -0.563      -1.358
10.TPP      -2.527      -0.988       0.521      -1.041
11.GEO      -1.542      -0.826      -0.196      -0.883
12.AGF      -1.178      -0.624      -0.072      -0.625
13.ECM       0.137       0.466       0.796       0.441
14.SSM       0.303       0.711       1.079       0.684
15.CON       0.604       1.071       1.479       1.041
16.PST       0.814       1.924       2.966       1.858
17.RLW       1.885       2.787       3.632       2.738
18.OFU       2.789       3.300       3.829       3.330
19.OLE       3.155       3.553       3.964       3.554
20.ELE       3.480       3.896       4.318       3.892
21.PSM       3.733       4.134       4.591       4.168
22.CAT       3.216       4.417       5.461       4.369
23.TRO       3.953       5.123       6.549       5.282
24.OLP       4.614       5.262       5.909       5.295
25.TRD       5.694       6.454       7.250       6.476
26.AIR       5.227       7.182       9.276       7.320
27.FIN       4.919       7.301      10.175       7.640
28.COA       6.581       7.818       9.125       7.878
29.TMS       5.668       8.530      11.086       8.591
30.CHM       8.071       9.299      10.534       9.319
31.TRK       7.031       9.367      11.851       9.577
32.MAR       9.485      13.072      16.424      13.049
33.GAS       4.437      13.183      22.311      13.462
34.FME      14.794      16.617      18.382      16.642
35.NFM      23.299      26.398      29.281      26.263
```

The tabular report is helpful, but it does not convey the results as immediately as a picture. GNUPLOT's errorbar plot format is a convenient graphical format for portraying this information. The libinclude interface to GNUPLOT does not support this type of plot, so the continuation of the program produces the GNUPLOT command and data files before invoking the GNUPLOT program:

```
* Write out a GNUPLOT file to generate a chart of the results:

File kplt / ex3.gnu /;
put  kplt;
kplt.lw = 0;

put "reset"/;
put 'set title "Sectoral Impacts with Quartiles"'/;
put "set linestyle 1 lt 8 lw 1 pt 8 ps 0.5"/;
put "set grid"/;
put 'set ylabel "% change"'/;
put "set xzeroaxis"/;
put "set bmargin 4"/;
put "set xlabel 'sector'"/;
put 'set xrange [1:',card(i),']'/;
put 'set xtics rotate (';
loop(ki, loop(i$imap(ki,i), put '\'/' "',i.tl,'" ', ord(ki):0,',';));
put @(file.cc-1) ')'/;
put "plot 'ex3.dat' notitle with errorbars ls 1"/;
putClose;

File kpltdata / ex3.dat /;
put  kpltdata;
kpltdata.nr =  2;
kpltdata.nw = 14;
kpltdata.nd =  6;
loop(ki, loop(i$imap(ki,i), put ord(ki):0,qvalue(i,"mean"), qvalue(i,"q25"), qvalue(i,"q75")/;));
putClose;
execute 'wgnupl32 ex3.gnu -';
```

This is the generated plot:

6.46.1.4 Example 4: Repeated computation of percentiles within a loop

The following code extract shows how to do repeated computation of percentiles within a loop while the data is changing. This is an extract of the model **[rank04]** from the GAMS Data Utilities Library.

```
* Do several iterations, computing percentiles in each step:

loop(iter,
*  Substitute a call to the NLP solver by a call to the random
*  number generator.  In many applications, this substitution
*  produces profoundly more sensible results.
*
*  solve catchment using nlp maximizing max;

   z(week) = uniform(0,1);

*  If you want to retrieve percentile values, you need to reassign
```

```
*  the percentiles that you wish to retrieve at this point in the
*  program.  If pct() were not reassigned at this point, the INPUT
*  values would correspond to the OUTPUTs from the previous call.

   pct(pctl) = pct0(pctl);

$  libInclude rank z week rnk pct

   pctval(iter,pctl) = pct(pctl);
);

display pctval;
```

Output:

```
----     58 Here are the INPUT values of PCT0 and PCT prior to the call to rank:

----     58 PARAMETER pct0  Percentiles to be computed

50 50.000,    75 75.000,    80 80.000,    95 95.000


----     58 PARAMETER pct  Percentiles to be computed (input) and those values (output)

50 50.000,    75 75.000,    80 80.000,    95 95.000


----    153 Here are the values of PCT0 and PCT after the call to rank:

----    153 PARAMETER pct0  Percentiles to be computed

50 50.000,    75 75.000,    80 80.000,    95 95.000


----    153 Note that rank has changed the OUTPUT value of pct

----    153 PARAMETER pct  Percentiles to be computed (input) and those values (output)

50 0.424,    75 0.662,    80 0.712,    95 0.864


----    273 PARAMETER pctval  Percentile values in successive iterations

               50          75          80          95

iter1       0.345       0.594       0.638       0.922
iter2       0.385       0.633       0.672       0.941
iter3       0.474       0.705       0.766       0.962
iter4       0.627       0.796       0.823       0.978
iter5       0.428       0.682       0.793       0.904
iter6       0.422       0.690       0.729       0.958
iter7       0.558       0.716       0.756       0.902
iter8       0.451       0.638       0.726       0.942
iter9       0.464       0.704       0.755       0.916
iter10      0.564       0.805       0.831       0.974
```

6.46.1.5 Example 5: Generating percentiles for heterogenous households.

```
$title Percentile ranking of household expenditure data with heterogenous household size

Set h / 0*100 /;

Parameter
   y(h)    'Aggregate expenditure associated with household type h'
   n(h)    'Number of persons associated with household type h'
   ypc(h)  'Per-capita expenditure of household type h'
   rank(h) 'Rank of household in per-capita expenditure';

* Assign some random values:
y(h)   = uniform(0.2,1.2);
n(h)   = uniform(1,6);
ypc(h) = y(h)/n(h);

* Assign ranks to household based on per-capita expenditures:
$libInclude rank ypc h rank

* Now determine percentile ranking of the households taking into account
* differences in numbers of members and household representation:
Set r 'Temporary set used for ranking' / r0*r100 /;

Parameter
   pcttmp(r) 'Temporary array for computing percentiles'
   pct(h)    'Percentile rankings for households';

Set r0(r) / r0 /;

* First, create an array with households assigned
loop((r0(r),h), pcttmp(r+(rank(h)-1)) = n(h););
loop(r,         pcttmp(r) = pcttmp(r) + pcttmp(r-1););
pcttmp(r) = pcttmp(r)/sum(h, n(h));

loop((r0(r),h), pct(h) = pcttmp(r+(rank(h)-1)););

Parameter ranking 'Ranking of households and expenditures';

loop((r0(r),h),
   ranking(r+(rank(h)-1),h,"n")   = n(h);
   ranking(r+(rank(h)-1),h,"ypc") = ypc(h);
   ranking(r+(rank(h)-1),h,"pct") = pct(h);
);

display ranking;
```

This example is also part of the GAMS Data Utilities Library, see model [**rank05**] for reference. It produces the following output:

```
                    n         ypc         pct
r0  .43         5.662       0.044       0.018
r1  .8          5.682       0.047       0.036
r2  .58         4.477       0.052       0.050
r3  .14         5.742       0.058       0.068
r4  .55         4.815       0.063       0.083
```

```
r5   .65        3.702       0.063       0.094
r6   .61        5.880       0.064       0.113
r7   .54        4.463       0.064       0.127
....
r98  .62        1.134       0.640       0.991
r99  .10        1.673       0.716       0.997
r100.73         1.053       1.076       1.000
```

Chapter 7

Application Programming Interfaces

The Application Programing Interfaces (APIs) to GAMS allow the seamless integration of GAMS into an application. The APIs by provides appropriate classes for the interaction with GAMS. The `GAMSDatabase` class for in-memory representation of data can be used for convenient exchange of input data and model results. Models written in GAMS can be run with the `GAMSJob` class and by using the `GAMSModelInstance` class a sequence of closely related model instances can be solved in the most efficient way. Note that the GAMS object-oriented API was not designed to offer modeling capability. The model needs to be formulated in GAMS and passed on to the `GAMSJob` class.

Five versions of object-oriented GAMS APIs :

- **.NET API** with Target Framework Version 4.6.2,
- C++ API works with C++17 or later,
- Java API works with Java SE 11 or later. The non-mainstream version of Java API that works with Java SE 8 is still available with maintenance support, but the functionality that is newer than GAMS 43 will be not available in this version. See Java Release Notes and Java Tutorial for more details.
- Python API works with Python 3.8 to 3.12.
- Matlab API works with Matlab 2017b (or later) and Octave 5.2 (or later).

In addition to the object-oriented GAMS APIs, there exist an R API and expert-level (or low-level) GAMS APIs in which its usage requires advanced knowledge of GAMS component libraries.

See Executing GAMS from other Environments if you want to execute GAMS directly from an application without using APIs.

7.1 Object-oriented APIs

The Reference Manuals give an overview of the features provides by GAMS APIs, release notes, and a list of all classes with descriptions. The Tutorials provide an overview of the basic functionality of the GAMS APIs and the Examples provides a list of examples available with GAMS distribution with detailed description. See also Release Notes and Supported Platforms.

7.1.1 Reference Manuals

The GAMS APIs reference manuals give an overview of the features provides by GAMS APIs, release notes, and a list of all classes with descriptions.

- .NET API
- C++ API
- Java API
- Python API
- Matlab API

7.1.2 Tutorials

GAMS API Tutorials provide an overview of basic functionalities of GAMS APIs and allow a user to get started to work with the GAMS API.

- .NET API Tutorial
- C++ API Tutorial
- Java API Tutorial
- Python API Tutorial
- Matlab API Tutorial

7.1.3 Examples

There are several examples avaiable for the different programming languages: .NET, C++, Java, and Python. Below we give a description of each example together with its link to the detailed page in different languages.

Examples	.NET	C++	Java	Python
Alias	Alias		Alias	alias.py
Benders2Stage	Benders2Stage, Benders2StageMT		Benders2Stage, Benders2StageMT	benders_2stage.py, benders_2stage_mt.py
Clad	Clad		Clad	clad.py
CutStock	Cutstock, SimpleCutstock		CutstockModel, SimpleCutstock, Cutstock	cutstock_class.py, simple_cutstock.py, cutstock.py
Domain Checking	DomainChecking	domainchecking.cpp	DomainCheck	domain_checking.py
Graphical User Interface (GUI)				
- CutstockGUI	CutStockGUI			

Examples	.NET	C++	Java	Python
- FarmGUI	FarmGUI			
- InterruptGUI	InterruptGui		InterruptGUI	interrupt_gui.py
- TransportGUI	TransportGUI		TransportGUI	
GAMS Remote Object	GAMSServer, GAMSClient			
Interrupt	Interrupt		ConsoleInterrupt	interrupt.py
Markowitz	Markowitz			markowitz.py
MessageReceiver Window	MessageReceiverWindow			
Special Values	SpecialValues		SpecialValues	special_values.py
Transport Model Sequence	Transport, Transport1 - Transport14	transport.cpp, transport1.cpp - transport14.cpp	TransportModel, Transport1 - Transport14	transport_class.py, transport1.py - transport14.py
Traveling Salesman Problem	Tsp		Tsp	tsp.py
Warehouse	Warehouse	warehouse.cpp	Warehouse	warehouse.py
Unit Test	NUnitTest			

These examples are available with the GAMS distribution in `[Path/To/GAMS]/apifiles/` directory. In that directory, the file `readme.txt` describes how to build and execute these examples. A GAMS script to compile and execute each example is available in `[Path/To/GAMS]/apilib_ml/` directory with a **table of contents**.

7.1.3.1 Alias

The Object-oriented API does not have the concept of a GAMS alias. An alias cannot be entered into a GAMSDatabase by API methods. Nevertheless, if the GAMSDatabase is loaded from a GDX container (e.g. the GAMSJob.OutDB), the database can contain aliases. Such an alias can be retrieved as a GAMSSet and consists of the elements of the aliased set. The method to find out if a GAMSSet in a GAMSDatabase is indeed an alias is to check if the symbol name is different from the lookup name (i.e. bool isAlias = db.GetSet("ii").Name != "ii"). This example model goes through the logic of how aliases are handled in the Object-oriented API.

See Alias example in [**.NET**, **Java**, **Python**].

7.1.3.2 Benders2Stage

This example implements a simple Benders decomposition method for a stochastic linear program. The underlying model implements a simple distribution system with stochastic demand data. This example actually consists of two programs: A sequential and a parallel implementation. In the sequential version, the master and the subproblems are implemented with the GAMSModelInstance object which allows resolving the model with modified input without regenerating the model. A GAMSModelInstance has a fixed model rim, so this provides a challenge for Benders master problem because every iteration adds new constraints (the Benders cuts) to the master. We get around this limitation of GAMSModelInstance by initializing the GAMSModelInstance of the master with a fixed number of empty (i.e. non-binding) placeholders constraints and during the run of the algorithm turn these placeholder constraints into valid Benders cuts. The parallel version extends the Benders2Stage example by solving the independent subproblems in parallel. For that we need to instantiate a separate GAMSModelInstance for each parallel

worker. We use the efficient GAMSModelInstance. CopyInstance method to accomplish this in the most effective way. The number of demand scenarios can be larger than the number of parallel workers. The distribution of work is handles through a work queue. The parallel execution of the subproblems is done in separate threads (the MT in the name of the example stands for Multi Threading), so there is very little overhead from disk activity.

See Bender2Stage example in [**.NET**, **Java**, **Python**] and Bender2StageMT example in [**.NET**, **Java**, **Python**].

7.1.3.3 Clad

This example demonstrates how to implement a complex termination criterion for a difficult MIP using GAMS/Cplex. We would like to achieve a globally optimal solution (relative gap 0%) but if solution time becomes larger than n1 seconds, we can compromise for a 10% gap, and if this is not achieved after n2 seconds, we compromise for a 20% gap, and again if this is not accomplished in n3 seconds we take whatever the solver has done so far and terminate the solve. This is implemented by executing GAMSJob.run in an independent thread and providing new tolerances for the relative gap in the main thread by supplying new GAMS/Cplex option files and triggers the processing of the new tolerance option by GAMS/Cplex through the GAMSJob.Interrupt method.

See Clad example in [**.NET**, **Java**, **Python**].

7.1.3.4 CutStock

This example contains two programs that implement a column generation approach to solve the cutting stock problem. In the program SimpleCutsock, the column generation scheme has been implemented in GAMS. Moreover the GAMS model with the input and output data has been wrapped in a class that separates all interaction with GAMS from the driving application. In the second example, the column generation approach has been entirely implemented in the program using GAMSJob for the master and GAMSModelInstance for the pricing problem. GAMS is used to build the master and pricing problems. The logic of the column generation method is in the application program.

See Cutstock Model example in [**.NET**, **Java**, **Python**], SimpleCustock example in [**.NET**, **Java**, **Python**], and Custock example in [**.NET**, **Java**, **Python**].

7.1.3.5 Domain Checking

Enforcing referential integrity also known in the GAMS lingo as domain checking is an essential and important part of GAMS. The Object-oriented API does a delayed domain checking of symbols. So you can add records to a domain controlled parameter (e.g. p(i)) even though the GAMSSet i does not contain the label (yet). The user can trigger an explicit check of the referential integrity by calling the GAMSDatabase.CheckDomains (or GAMSSymbol.CheckDomains). The Object-oriented API provides methods to access the records that violate the referential integrity (see GAMSDatabaseDomainViolation for details). Domain checking is implicitly done when the GAMSDatabase is exported to a GDX file via the GAMSDatabase.Export method or for databases provided in the GAMSJob.Run method. The implicit domain check can be suppressed (and left to GAMS when importing data) via the GAMSDatabase.SuppressAutoDomainChecking property. This example demonstrates how to trigger domain checking and how to access the records that violate the referential integrity.

See Domain checking example in [**.NET**, **C++**, **Java**, **Python**].

7.1.3.6 GUI Examples

Modern development frameworks like Microsoft's Visual Studio, Python, and Java allow to quickly build graphical user interfaces. Every example consists of a simple GUI that provides data entry, executing of a GAMSJob and graphically represents the results of a GAMS model.

- **CutstockGUI**: This example allows to either load data from a database or to enter data manually. A cutting stock problem is solved by using both the GAMSJob and the GAMSModelInstance class in order to implement a decomposition approach. See also CutStock Example. See this example in [**.NET**].
- **InterruptGUI**: This small example demonstrates how to run a GAMS model in a graphical user interface. This has all rudimentary features we know from the GAMS IDE: starting a job, capture the GAMS log in a window, and providing a button to interrupt. The underlying mechanism to interrupt the job is similar to the Interrupt Example (GAMSJob.Interrupt), but the trigger mechanism is very different. See this example in [**.NET**, **Java**, **Python**].
- **FarmGUI**: For this farming model, the input data can be entered directly into several tables. The GAMS model itself is executed by using one single GAMSJob instance. Results are displayed in different charts and tables. See this example in [**.NET**].
- **TransportGUI**: In this example a series of transportation problems is solved using GAMSModelInstance. Data can be entered into tables or can be loaded from a database. When data is entered into a table, related tables are updated automatically. Results are shown in tables and/or a bar chart. See also Transport Model Sequence Example. See this example in [**.NET**, **Java**].

7.1.3.7 GAMS Remote Object

This example demonstrates how to implement a simple GAMS Server. The example has two parts: GAMSServer and GAMSClient. The example is configured to run the client and server on the same machine, but can easily be altered to run on different machines. Both client and server need access to the GAMSRemoteClass that implements the server. The method to communicate is very simple. The client sets up the ingredients for a GAMSJob (model text, GDX input, parameters) and sends them serialized (via a byte representation of the GDX and parameter file) to the server. The server recreates the GAMS objects from the serialized representations, runs the model, and ships GAMSJob.OutDB serialized back to the client for further processing.

See GAMSServer example in [**.NET**] and GAMSClient in [**.NET**].

7.1.3.8 Interrupt

Ctrl-C in a regular console application results in the interrupt of the entire job. GAMS users are used to the fact that Ctrl-C interrupts a solve but then continues with the execution of the remaining GAMS job. The example demonstrates how to alter some console properties to get this behavior of Ctrl-C.

See Interrupt example in [**.NET**, **Java**, **Python**].

7.1.3.9 Markowitz

This is a small graphical program that plots the efficient frontier of Markowitz' portfolio selection problem with the two objectives return and risk. The example utilizes the GAMSModelInstance class to solve the parameterized objective max lambda∗return - (1-lambda)∗risk in the most efficient way.

See Markowitz example in [**.NET**, **Python**].

7.1.3.10 MessageReceiver Window

The little example demonstrates how to implement a custom visual log. From GAMS one can send messages to the form started by this program via the put_utility "winmsg". The form also understands some commands (a message that starts with @) to save the content of the form, to put the content into the clipboard, or to terminate the program. The MessageReceiverWindow is also distributed as an executable in the GAMS system directory ready to be used. The model mrw01 in the GAMS Test Library demonstrates the use of the program.

See MessageReceiver Window example in [**.NET**].

7.1.3.11 Special Values

This example shows how special values of the programming language (e.g. infinity) percolate down to GAMS. Infinity and NaN (not a number) are well defined. The GAMS Undefined and EPS need special considerations.

See Special Values example in [**.NET**, **Java**, **Python**].

7.1.3.12 Transport Model Sequence

This set of examples demonstrates the use of the Object-oriented API on the simple transport model **trnsport.gms : A Transportation Problem** [trnsport]. The detailed description of the various examples is provided in the Object-oriented API tutorials in the GAMS documentation.

See Transport Model example in [**.NET**, **C++**, **Java**, **Python**].

7.1.3.13 Traveling Salesman Problem

This example demonstrates how to use a GAMSModelInstance to implement the subtour elimination algorithm for the Traveling Salesman Problem (TSP) problem. Similar to Benders2Stage example, we have a placeholder for the subtour elimination constraint that gets generated in each iteration of the algorithm. In contrast to the Benders example, here we regenerate the GAMSModelInstance if the original number of placeholders was not big enough. We continue this process until all subtours are eliminated.

See Traveling Salesman Problem example in [**.NET**, **Java**, **Python**].

7.1.3.14 Unit Test

This example contains all the unit tests we run on the .Net Object-oriented API. See Unit Test example in [**.NET**].

7.1.3.15 Warehouse

This example demonstrates how to solve a simple GAMS model to assign stores to warehouses for different data sets in parallel. The model has been parameterized. The data can be derived from a few numbers namely the number of warehouses, stores, and some fixed cost scalar. The results of the model are written into a single result database that is protected across the parallel threads via a mutex.

See Warehouse example in [**.NET**, **C++**, **Java**, **Python**].

7.1.4 Release Notes

GAMS API Release Notes provide an overview of changes to the GAMS APIs in releases of the GAMS distribution.

- Java API Release Notes

7.1.5 Supported Platforms

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
GAMS C++	✓	✓	✓	✓
GAMS Java	✓	✓	✓	✓
GAMS.NET	✓	✓	✓	✓
GAMS Python	✓	✓	✓	✓
GAMS Matlab	✓	✓	✓	✓

7.2 Expert-Level APIs

In addition to the object-oriented GAMS API, there exist expert-level (or low-level) APIs to some component libraries of the GAMS system. These APIs are used internally in GAMS for a long time and help in the deployment of GAMS models. While they offer high performance and flexibility, they also require advanced knowledge and it is recommended that the object-oriented GAMS API is used if possible. The expert-level APIs are offered for *C*, *C#*, *Delphi*, *Fortran*, *Java*, *Python*, and *Visual Basic*.

The DCT, GEV, and GMO components are mainly used to build solver links to GAMS (see, e.g., the `GAMSLinks project`. In fact, all professional GAMS solver links are based on these components. They provide access to an instance of a model generated by a `solve` statement. If you plan to build advanced algorithms for models or need to solve a sequence of very similar models these components might represent an alternative to an implementing using the GAMS language.

While we also strive for backward compatibility with our component libraries, but programming interfaces are commonly subject to change, so programs based on these APIs might require adjustment when updating to a new GAMS version. The Release Notes have a section on these API changes.

Overviews of the exported function in pseudo code are available in the following summary pages:

- `CFG API` (GAMS Configuration Object)
- `DCT API` (GAMS Dictionary Object)
- `GDX API` (GAMS Data Exchange Object) which is also available as `open source project on GitHub` with `additional documentation`.
- `GEV API` (GAMS Environment Object) with additional GAMS Environment Object Options.

- GMO API (GAMS Modeling Object) with additional information on philosophy and design.
- IDX API (GAMS IDX Object)
- OPT API (GAMS Option Object)
- PAL API (GAMS Auditing and Licensing Object)

Note

Without having a GAMS system directory in the PATH environment variable, the xyzCreateD functions (e.g. gdxCreateD, optCreateD, ...) might cause some issues finding libraries that are only in the GAMS system directory on Windows.

7.2.1 Supported Platforms

	x86 64bit MS Windows	x86 64bit Linux	x86 64bit macOS	arm 64bit macOS
C	✓	✓	✓	✓
C#	✓			
Delphi	✓			
Fortran∗	✓	✓	✓	✓
Java	✓	✓	✓	✓
Python	✓	✓	✓	✓
VBA	✓			
VB.NET	✓			

∗ Fortran API files are based on C code to load dynamic libraries. For Windows pure Fortran API files for Intel and Lahey Fortran compilers are included

7.3 C++ API

GAMS C++ API provides a convenient way to exchange input data and model results with in-memory representation of data (**GAMSDatabase**), and to create and run GAMS models (**GAMSJob**) that can be customized by GAMS options (**GAMSOptions**). Furthermore, they introduce a way to solve a sequence of closely related model instances in a more efficient way (**GAMSModelInstance**).

7.4 Java API

GAMS Java API provides a Java programming interface to the General Algebraic Model System (GAMS). GAMS Java API objects allow a convenient way to exchange input data and model results with in-memory representation of data (GAMSDatabase), and to create and run GAMS models (GAMSJob) that can be

customized by GAMS options (GAMSOptions). Furthermore, they introduce a way to solve a sequence of closely related model instances in a more efficient way (GAMSModelInstance).

The underlying GAMS engine relies to some extent on file based communication (e.g. the listing file) and other unmanaged resources. The use of external resources in the Java environment requires special attention. Hence, some objects need to be properly disposed before the Java garbage collector does its job.

A GAMS program can include other source files (e.g. $include), load data from GDX files (e.g. $GDXIN or execute_load), and create PUT files. All these files can be specified with a (relative) path and therefore an anchor into the file system is required. The base object GAMSWorkspace manages the anchor to the file system.

With the exception of GAMSWorkspace the objects in the gams namespace cannot be accessed across different threads unless the instance is locked. The classes themselves are thread safe and multiple objects of the class can be used from different threads (see below for restrictions on solvers that are not thread safe within the GAMSModelInstance class).

Note

If you use multiple instances of the GAMSWorkspace in parallel, you should avoid using the same WorkingDirectory. Otherwise you may end up with conflicting file names.

Currently only Cplex, Gurobi, and SoPlex fully utilize the power of solving GAMSModelInstances. Some solvers will not work in a multi-threaded application using GAMSModelInstances. For some solvers this is unavoidable because the solver library is not thread safe (e.g. MINOS), other solvers are in principle thread safe but the GAMS link is not (e.g. SNOPT). Moreover, GAMSModelInstances are not available for quadratic model types (QCP, MIQCP, RMIQCP).

This version of the GAMS Java API also does not provide support for the following GAMS components: acronyms, GAMS compilation/execution errors, structured access to listing file, and solver options.

7.5 Python API

The GAMS API is a Python package that contains several sub-modules that enable the control of the GAMS system as well as the movement of data between GAMS and Python. Currently the API supports the Python versions 3.8 to 3.12. The following table gives an overview of all available sub-modules:

Sub-Module	Description
connect (beta)	Used primarily by GMSPython to digest YAML syntax to Extract, Transform and Load (ETL) data into GAMS, but can be used in native Python environments.
control	Enables full control of the GAMS System
core	Core GAMS API tools used to connect to GDX, GMD, GMO and other GAMS objects. Requires expert level knowledge.
engine	GAMS Engine API (OpenAPI compliant), manages jobs with GAMS Engine
magic (beta)	Enables the use of GAMS from within Jupyter notebooks
tools (beta)	Code base for the GAMS tools library
transfer	Data Only API – Allows GAMS data to be maintained outside a GAMS script

Note

Due to compatibility issues the GAMS Python API does not work with the Python interpreter from the Microsoft Store.

To install the API please visit: Getting Started.

7.5.1 Migrate import statements

With the release of GAMS 42 the GAMS Python ecosystem has been restructured – the new structure has many benefits (easier/safer (un)installs, cleaner module namespaces, etc.).

Attention

The new API structure cannot be used to simply "update" previous versions – users should build new python environments from scratch before attempting to install.

Restructuring of the GAMS python API ecosystem was confined to the creation of the new nested structure – class, method and other variables names were not modified. The import statements in legacy code will need to be updated if using the new system. Best practice will be to use the new package structure to import different sub-modules as needed (and avoid `from <module_name> import *` syntax). We provide a mapping between the old syntax and the new to aid in the transition to the new API structure:

Old `import` statement	New `import` recommendation(s)
`from gams import GamsWorkspace`	`from gams import GamsWorkspace`
`from gams import *`	`import gams`

`from gdxcc import *`	`from gams.core import gdx`
	-or-
	`import gams.core.gdx as gdx`

`from optcc import *`	`from gams.core import opt`
	-or-
	`import gams.core.opt as opt`

`import gamstransfer as gt`	`from gams import transfer as gt`
	-or-
	`import gams.transfer as gt`

`import gams_engine`	'import gams.engine'

Note

Jupyter users will need to migrate their `reload_ext gams_magic` -> `reload_ext gams.magic` and `load_ext gams_magic` -> `load_ext gams.magic`

7.5.1.1 Testing for Old vs New API

Users may be running the same python code with different versions of the Python API. If this is the case, it might be beneficial to include conditional import statements. It is possible to test for the GAMS major version number with the `GamsWorkspace.api_major_rel_number` property:

```
from gams import GamsWorkspace
if GamsWorkspace.api_major_rel_number<42:  # old API structure
    import gdxcc as gdx
    from gams import *
    import gamstransfer as gt
else:  # new API structure
    import gams.core.gdx as gdx
    from gams.control import *
    import gams.transfer as gt
```

Attention

While conditional import statements can be helpful, users are strongly encouraged to modify their code to use the new structure.

7.5.2 GAMS Python API Structure

7.5.3 Magic (Jupyter Notebooks)

Note

This feature is currently in beta status.

7.5.3.1 Introduction

GAMS Jupyter Notebooks allow to use notebook technology in combination with GAMS. If you just want to learn GAMS there are probably better ways doing this. Notebooks allow you to combine GAMS and Python. The former works great with well structured data and optimization models, while the latter is very rich in features to retrieve, manipulate, and visualize data that comes in all sort of ways. Combining GAMS and Python in a notebook it is relatively easy to tell an optimization story with text, data, graphs, math, and models.

7.5.3.2 Getting Started

The first step in getting started with GAMS Jupyter Notebooks is to make your Python 3 installation aware of the GAMS Python API described in the Getting Started section of the API tutorial. We recommend to follow the steps below which are specifically tailored for getting started with GAMS Jupyter notebooks. While any Python 3.8 to 3.12 installation is supported, we recommend the use of `miniconda` Python distributions.

Attention

All core third party dependencies will be installed if the user supplies the optional `pip` syntax (`pip install gamsapi[magic]`).

In addition to the GAMS Python API collection (and the core `magic` dependencies), the examples located in `[PATH TO GAMS]/api/python/examples/magic` will require the additional packages: `jupyterlab`, `matplotlib`, and `tabulate`. The following code section shows how to create and set up a `conda` environment for GAMS Jupyter notebooks:

The notebooks `Millco.ipynb` and `Introduction.ipynb` located in `api/python/examples/magic` are good starting points to get familiar with Jupyter notebooks and GAMS. The remainder of this section gives the dialog of the `Introduction.ipynb` notebook. Please see also the other notebook examples:

- `PickStock Example`
- `Filling grids with (distinct) polyominos`
- `DICE Model from 2018 Nobel laureate William D. Nordhaus`
- `PickStock Analysis using R`

7.5.3.3 Tutorial

7.5.3.4 Converting GAMS Jupyter Notebooks into Python Scripts

Sometimes it can be useful to execute the logic of a GAMS Jupyter notebook in a standalone Python script. This can be achieved by using the `gams.magic.GamsInteractive` class which implements the back-end logic of `gams.magic` and does require neither `IPython` nor `jupyter`. Translating GAMS magic commands into Python method equivalents is stright forward. First, the Python equivalent of `%reload_ext gams.magic` needs to be added to the beginning of a Python script:

```
from gams.magic import GamsInteractive
gams = GamsInteractive()
```

Afterwards, each magic command or method available in GAMS Jupyter notebooks has an equivalent in `GamsInteractive` with the same name. GAMS magic commands like `%gams` or `%gams_reset` can be translated into methods of the exact same name - in this case `GamsInteractive.gams()` and `GamsInteractive.gams_reset()`. Methods and properties which are accessed directly (without GAMS magic command) in a Jupyter notebook like `gams.exchange_container` or `gams.activate()` can be used in the exact same way from within `GamsInteractive`. Options and parameters of GAMS magic commands can be used with corresponding arguments of the equivalent methods.

While GAMS magic commands can be translated very easily, certain interactive functionality like chart plotting might require a different mechanism when being translated. Also `display()` might need to be changed into `print()` or similar to be working in a standalone Python script. In addition, Jupyter notebooks display data which is returned by the last command of a Python cell. In Python we need to use another `print()` of the return value to get it as output.

For a complete example of a translated GAMS Jupyter notebook see `[PATH TO GAMS]/api/python/examples/magic/millc` which is a translation of the `Millco.ipynb` notebook.

7.5.4 Transfer

`transfer` is a tool to maintain GAMS data outside a GAMS script in a programming language like Python or Matlab. It allows the user to add GAMS symbols (Sets, Aliases, Parameters, Variables and Equations), to manipulate GAMS symbols, as well as read/write symbols to different data endpoints. `transfer`'s main focus is the highly efficient transfer of data between GAMS and the target programming language, while keeping those operations as simple as possible for the user. In order to achieve this, symbol records – the actual and potentially large-scale data sets – are stored in native data structures of the corresponding programming languages. The benefits of this approach are threefold: (1) The user is usually very familiar with these data structures, (2) these data structures come with a large tool box for various data operations, and (3) optimized methods for reading from and writing to GAMS can transfer the data as a bulk – resulting in the high performance of this package. This documentation describes, in detail, the use of `transfer` within a Python environment.

Data within `transfer` will be stored as Pandas DataFrame. The flexible nature of Pandas DataFrames makes them ideal for storing/manipulating sparse data. Pandas includes advanced operations for `indexing and slicing`, `reshaping`, `merging` and even `visualization`.

Pandas also includes a number of advanced data `I/O tools` that allow users to generate DataFrames directly from CSV (`.csv`), JSON (`.json`), HTML (`.html`), Microsoft Excel (`.xls`, `.xlsx`), SQL , pickle (`.pkl`), SPSS (`.sav`, `.zsav`), SAS (`.xpt`, `.sas7bdat`), etc.

Centering `transfer` around the Pandas DataFrame gives GAMS users (on a variety of platforms – macOS, Windows, Linux) access to tools to move data back and forth between their favorite environments for use in their GAMS models.

The goal of this documentation is to introduce the user to `transfer` and its functionality. This documentation is not designed to teach the user how to effectively manipulate Pandas DataFrames; users seeking a deeper understanding of Pandas are referred to the extensive `documentation`.

Experienced GAMS API users seeking detailed documentation and examples are directed to Main Classes and Additional Topics.

7.5.4.1 Recommended Import

Users can access the `transfer` sub-module with either of the following (equivalent) import statements once the GAMS Python API has been installed:

```
import gams.transfer as gt
from gams import transfer as gt
```

7.5.4.2 Design

Storing, manipulating, and transforming sparse data requires that it lives within an environment – this data can then be linked together to enable various operations. In `transfer` we refer to this "environment" as the `Container`, it is the main repository for storing and linking our sparse data. Symbols can be added to the `Container` from a variety of GAMS starting points but they can also be generated directly within the Python environment using convenient function calls that are part of the `transfer` package; a symbol can only belong to one container at a time.

The process of linking symbols together within a container was inspired by typical GAMS workflows but leverages aspects of object oriented programming to make linking data a natural process. Linking data enables data operations like implicit set growth, domain checking, data format transformations (to dense/sparse matrix formats), etc – all of these features are enabled by the use of ordered `pandas.CategoricalDtype` data types. All of these details will be discussed in the following sections.

7.5.4.3 Naming Conventions

Methods – functions that operate on a object – are all verbs (i.e., `getMaxAbsValue()`, `getUELs()`, etc.) and use camel case for identification purposes. Methods are, by convention, tools that "do things"; that is they involve some, potentially expensive, computations. Some `transfer` methods accept arguments, while others are simply called using the `()` notation. Plural arguments (`columns`) hint that they can accept lists of inputs (i.e., a list of symbol names) while singular arguments (`column`) will only accept one input at a time.

Properties – inherent attributes of an object – are all nouns (i.e., `name`, `number_records`, etc.) and use snake case (lower case words separated by underscores) for identification purposes. Object properties (or "object attributes") are fundamental to the object and therefore they are not called like methods; object properties are simply accessed by other methods or user calls. By convention, properties only require trival amounts of computation to access.

Classes – the basic structure of an object – are all singular nouns and use camel case (starting with a capital first letter) for identification purposes.

7.5.4.4 Install

The user must download and install the latest version of GAMS in order to install `transfer`. `transfer` is installed when the GAMS Python API is built and installed. The user is referred HERE for instructions on how to install the Python API files. `transfer` and all GAMS Python API files are compatible with environment managers such as Anaconda.

7.5.4.5 Examples

GDX Read

Reading in all symbols can be accomplished with one line of code (we reference data from the **`trnsport.gms`** example).

```
import gams.transfer as gt
m = gt.Container("trnsport.gdx")
```

All symbol data is organized in the data attribute – `m.data[<symbol_name>].records` (the `Container` is also subscriptable, `m[<symbol_name>].records` is an equivalent statement) – records are stored as Pandas DataFrames.

Write Symbol to CSV

Writing symbol records to a CSV can also be accomplished with one line.

```
m["x"].records.to_csv("x.csv")
```

Write a New GDX

There are six symbol classes within `transfer`: 1) Sets, 2) Parameters, 3) Variables, 4) Equations, 5) Aliases and 6) UniverseAliases. For purposes of this quick start, we show how to recreate the `distance` data structure from the **`trnsport.gms`** model (the parameter `d`). This brief example shows how users can achieve "GAMS-like" functionality, but within a Python environment – `transfer` leverages the object oriented programming to simplify syntax.

```
import gams.transfer as gt
import pandas as pd
m = gt.Container()
# create the sets i, j
i = gt.Set(m, "i", records=["seattle", "san-diego"], description="supply")
j = gt.Set(m, "j", records=["new-york", "chicago", "topeka"], description="markets")
# add "d" parameter -- domain linked to set objects i and j
d = gt.Parameter(m, "d", [i, j], description="distance in thousands of miles")
# create some data as a generic DataFrame
dist = pd.DataFrame(
    [
        ("seattle", "new-york", 2.5),
        ("seattle", "chicago", 1.7),
        ("seattle", "topeka", 1.8),
        ("san-diego", "new-york", 2.5),
        ("san-diego", "chicago", 1.8),
        ("san-diego", "topeka", 1.4),
    ],
    columns=["from", "to", "thousand_miles"],
)
# setRecords will automatically convert the dist DataFrame into a standard DataFrame format
d.setRecords(dist)
# write the GDX
m.write("out.gdx")
```

This example shows a few fundamental features of `transfer`:

1. An empty Container is analogous to an empty GDX file
2. Symbols will always be linked to a Container (notice that we always pass the Container reference `m` to the symbol constructor)
3. Records can be added to a symbol with the `setRecords()` method or through the `records` constructor argument (internally calls `setRecords()`). `transfer` will convert many common Python data structures into a standard format.
4. Domain linking is possible by passing domain set objects to other symbols
5. Writing a GDX file can be accomplished in one line with the `write()` method.

Full Example

It is possible to use everything we now know about `transfer` to recreate the **`trnsport.gms`** results in GDX form. As part of this example we also introduce the `write` method (and generate `new.gdx`). We will discuss it in more detail in the following section: GDX Read/Write.

```
import gams.transfer as gt
# create an empty Container object
m = gt.Container()
# add sets
i = gt.Set(m, "i", records=["seattle", "san-diego"], description="supply")
j = gt.Set(m, "j", records=["new-york", "chicago", "topeka"], description="markets")
# add parameters
a = gt.Parameter(m, "a", ["*"], description="capacity of plant i in cases")
b = gt.Parameter(m, "b", j, description="demand at market j in cases")
d = gt.Parameter(m, "d", [i, j], description="distance in thousands of miles")
```

```
f = gt.Parameter(
    m, "f", records=90, description="freight in dollars per case per thousand miles"
)
c = gt.Parameter(
    m, "c", [i, j], description="transport cost in thousands of dollars per case"
)
# set parameter records
cap = pd.DataFrame([("seattle", 350), ("san-diego", 600)], columns=["plant", "n_cases"])
a.setRecords(cap)
dem = pd.DataFrame(
    [("new-york", 325), ("chicago", 300), ("topeka", 275)],
    columns=["market", "n_cases"],
)
b.setRecords(dem)
dist = pd.DataFrame(
    [
        ("seattle", "new-york", 2.5),
        ("seattle", "chicago", 1.7),
        ("seattle", "topeka", 1.8),
        ("san-diego", "new-york", 2.5),
        ("san-diego", "chicago", 1.8),
        ("san-diego", "topeka", 1.4),
    ],
    columns=["from", "to", "thousand_miles"],
)
d.setRecords(dist)
# c(i,j) = f * d(i,j) / 1000;
cost = d.records.copy(deep=True)
cost["value"] = f.records.loc[0, "value"] * cost["value"] / 1000
c.setRecords(cost)
# add variables
q = pd.DataFrame(
    [
        ("seattle", "new-york", 50, 0),
        ("seattle", "chicago", 300, 0),
        ("seattle", "topeka", 0, 0.036),
        ("san-diego", "new-york", 275, 0),
        ("san-diego", "chicago", 0, 0.009),
        ("san-diego", "topeka", 275, 0),
    ],
    columns=["from", "to", "level", "marginal"],
)
x = gt.Variable(
    m, "x", "positive", [i, j], records=q, description="shipment quantities in cases",
)
z = gt.Variable(
    m,
    "z",
    records=pd.DataFrame(data=[153.675], columns=["level"]),
    description="total transportation costs in thousands of dollars",
)
# add equations
cost = gt.Equation(m, "cost", "eq", description="define objective function")
supply = gt.Equation(m, "supply", "leq", i, description="observe supply limit at plant i")
demand = gt.Equation(m, "demand", "geq", j, description="satisfy demand at market j")
# set equation records
cost.setRecords(
    pd.DataFrame(data=[[0, 1, 0, 0]], columns=["level", "marginal", "lower", "upper"])
)
supplies = pd.DataFrame(
    [
        ("seattle", 350, "eps", float("-inf"), 350),
        ("san-diego", 550, 0, float("-inf"), 600),
    ],
    columns=["from", "level", "marginal", "lower", "upper"],
)
supply.setRecords(supplies)
demands = pd.DataFrame(
    [
        ("new-york", 325, 0.225, 325),
        ("chicago", 300, 0.153, 300),
        ("topeka", 275, 0.126, 275),
    ],
    columns=["from", "level", "marginal", "lower"],
)
demand.setRecords(demands)
m.write("new.gdx")
```

7.5.4.6 Extended Examples

Get HTML data

```
import gams.transfer as gt
```

```
import pandas as pd
url = "https://www.fdic.gov/resources/resolutions/bank-failures/failed-bank-list"
dfs = pd.read_html(url)
# pandas will create a list of dataframes depending on the target URL, we just need the first one
df = dfs[0]
m = gt.Container()
b = gt.Set(m, "b", ["*"], records=df["Bank NameBank"].unique(), description="Bank Name")
s = gt.Set(
    m,
    "s",
    ["*"],
    records=df["StateSt"].sort_values().unique(),
    description="States (alphabetical order)",
)
c = gt.Set(
    m,
    "c",
    ["*"],
    records=df["CityCity"].sort_values().unique(),
    description="Cities (alphabetical order)",
)
c_to_s = gt.Set(
    m,
    "c_to_s",
    [c, s],
    records=df[["CityCity", "StateSt"]]
    .drop_duplicates()
    .sort_values(by=["StateSt", "CityCity"]),
    description="City/State pair",
)
bf = gt.Parameter(
    m,
    "bf",
    b,
    records=df[["Bank NameBank", "FundFund"]]
    .drop_duplicates(subset="Bank NameBank")
    .sort_values(by=["Bank NameBank"]),
    description="Bank Namd & Fund #",
)
In [1]: m.isValid()
Out[1]: True
```

Note

Users can chain Pandas operations together and pass those operations through to the `records` argument or the `setRecords` method.

Get PostgreSQL data (w/ sqlalchemy)

```
import gams.transfer as gt
from sqlalchemy import create_engine
import pandas as pd
# connect to postgres (assuming a localhost)
engine = create_engine("postgresql://localhost:5432/" + <database_name>)
df = pd.read_sql(<sql_table_name>, con=engine, index_col=0)
# create the Container and add symbol
m = Container()
p = Parameter(m, <sql_table_name>)
# we need to figure out the symbol dimensionality (potentially from the shape of the dataframe)
r, c = df.shape
p.dimension = c - 1
# set the records
p.setRecords(df)
# write out the GDX file
m.write("out.gdx")
```

7.5.4.7 Main Classes

Container

The main object class within `transfer` is called `Container`. The `Container` is the vessel that allows symbols to be linked together (through their domain definitions), it enables implicit set definitions, it enables structural manipulations of the data (matrix generation), and it allows the user to perform different read/write operations.

Constructor

Constructor Arguments

Argument	Type	Description	Required	Default
load_from	str or PathLike object, GMD Object Handle, GamsDatabase object, Container	Points to the source of the data being read into the Container	No	None
system_directory	str	Absolute path to GAMS system directory	No	Attempts to find the GAMS installation by creating a GamsWorkspace object and loading the system_directory attribute.

Creating a Container is a simple matter of initializing an object. For example:

```
import gams.transfer as gt
m = gt.Container()
```

This new Container object, here called m, contains a number of convenient properties and methods that allow the user to interact with the symbols that are in the Container. Some of these methods are used to filter out different types of symbols, other methods are used to numerically characterize the data within each symbol.

Properties

Property	Description	Type	Special Setter Behavior
data	main dictionary that is used to store all symbol data (case preserving)	CasePreservingDict	-
modified	Flag that identifies if the Container has been modified in some way. Container.modifed=False will reset this flag for all symbols in the container as well as the container itself.	bool	-
summary	returns a brief summary of the Container	dict	-

Symbols are organized in the Container under the data Container attribute. The dot notation (m.data) is used to access the underlying dictionary. Symbols in this dictionary can then be retrieved with the standard bracket notation (m.data[<symbol_name>]). The Container is also subscriptable (i.e., m["i"] will return the i Set object just as if the user called m.data["i"]). The behavior of the data dictionary is has been customized to be case-insensitive (which mimics the behavior of GAMS) – m["i"] and m["I"] will return the same object.

```
In [1]: m.data
Out[1]:
{'i': <Set 'i' (0x7f95b8d63e80)>,
```

```
 'j': <Set 'j' (0x7f95b8d63a60)>,
 'a': <Parameter 'a' (0x7f95b8d63ee0)>,
 'b': <Parameter 'b' (0x7f95b8d63d00)>,
 'd': <Parameter 'd' (0x7f95b8da86a0)>,
 'f': <Parameter 'f' (0x7f95b8da8670)>,
 'c': <Parameter 'c' (0x7f95b8da83d0)>,
 'x': <Positive Variable 'x' (0x7f95b8da83a0)>,
 'z': <Free Variable 'z' (0x7f95b8da8400)>,
 'cost': <Eq Equation 'cost' (0x7f95b8da82b0)>,
 'supply': <Leq Equation 'supply' (0x7f95b8da8280)>,
 'demand': <Geq Equation 'demand' (0x7f95b8da8580)>}
```

Symbol existence in the `Container` can be tested with an overloaded Python `in` operator. The following (case-insensitive) syntax is possible:

```
In [1]: 'i' in m
Out[1]: True
In [2]: 'I' in m
Out[2]: True
In [3]: i in m
Out[3]: True
```

Note

The final example assumes the existence of a separate symbol object called `i`.

The `Container` can also iterate through the symbols (will return a tuple of `(symbol_name, symbol_object)` using a `for-loop` as in the following example. The `list` and `dict` methods can be useful in creating lists and dictionaries of all symbols in the `Container`.

```
In [1]: m = gt.Container("out.gdx")
In [2]: for name, obj in m:
   ...:     print(name, obj)
   ...:
i <Set 'i' (0x7f9038533fa0)>
j <Set 'j' (0x7f9038533fd0)>
a <Parameter 'a' (0x7f9038533340)>
b <Parameter 'b' (0x7f9038533f10)>
d <Parameter 'd' (0x7f9038533f70)>
f <Parameter 'f' (0x7f9038533160)>
c <Parameter 'c' (0x7f90583b4bb0)>
x <Positive Variable 'x' (0x7f90583b4070)>
z <Free Variable 'z' (0x7f90583b5570)>
cost <Eq Equation 'cost' (0x7f90583b44f0)>
supply <Leq Equation 'supply' (0x7f9008c33df0)>
demand <Geq Equation 'demand' (0x7f9008c33b80)>
In [3]: list(m)
Out[3]:
[('i', <Set 'i' (0x7fced87d3790)>),
 ('j', <Set 'j' (0x7fced882e530)>),
 ('a', <Parameter 'a' (0x7fced882e9b0)>),
 ('b', <Parameter 'b' (0x7fced882ebf0)>),
 ('d', <Parameter 'd' (0x7fced882e800)>),
 ('f', <Parameter 'f' (0x7fced882e9e0)>),
 ('c', <Parameter 'c' (0x7fced882e320)>),
 ('x', <Positive Variable 'x' (0x7fced882e650)>),
 ('z', <Free Variable 'z' (0x7fced882e410)>),
 ('cost', <Eq Equation 'cost' (0x7fcec8918640)>),
 ('supply', <Leq Equation 'supply' (0x7fcec8918610)>),
 ('demand', <Geq Equation 'demand' (0x7fcec89183a0)>)]
In [5]: dict(m)
Out[5]:
{'i': <Set 'i' (0x7fced87d3790)>,
 'j': <Set 'j' (0x7fced882e530)>,
 'a': <Parameter 'a' (0x7fced882e9b0)>,
 'b': <Parameter 'b' (0x7fced882ebf0)>,
 'd': <Parameter 'd' (0x7fced882e800)>,
 'f': <Parameter 'f' (0x7fced882e9e0)>,
 'c': <Parameter 'c' (0x7fced882e320)>,
 'x': <Positive Variable 'x' (0x7fced882e650)>,
 'z': <Free Variable 'z' (0x7fced882e410)>,
 'cost': <Eq Equation 'cost' (0x7fcec8918640)>,
 'supply': <Leq Equation 'supply' (0x7fcec8918610)>,
 'demand': <Geq Equation 'demand' (0x7fcec89183a0)>}
```

Note

The `len` (length) function can be used to quickly find out how many symbols exist in the `Container` (`len(m)`).

Methods

Method	Description	Arguments/Defaults	Returns
addAlias	Container method to add an Alias	name (str) alias_with (Set, Alias)	Alias object
addEquation	Container method to add an Equation	name (str) type (str) domain=[] (str, list) records=None (pandas.DataFrame, numpy.ndarry, None) domain_forwarding=False (bool) description="" (str) uels_on_axes=False (bool)	Equation object
addParameter	Container method to add a Parameter	name (str) domain=None (str, list, None) records=None (pandas.DataFrame, numpy.ndarry, None) domain_forwarding=False (bool) description="" (str) uels_on_axes=False (bool)	Parameter object
addSet	Container method to add a Set	name (str) domain=None (str, list, None) is_singleton=False (bool) records=None (pandas.DataFrame, numpy.ndarry, None) domain_forwarding=False (bool) description="" (str) uels_on_axes=False (bool)	Set object
addUniverseAlias	Container method to add a UniverseAlias	name (str)	UniverseAlias object

Method	Description	Arguments/Defaults	Returns
addVariable	Container method to add an Variable	name (str) type="free" (str) domain=[] (str, list) records=None (pandas.DataFrame, numpy.ndarry, None) domain_forwarding=False (bool) description="" (str) uels_on_axes=False (bool)	Variable object
capitalizeUELs	will capitalize all UELs in the Container or a subset of specified symbols, can be chained with other *UELs string operations	symbols=None (str, list, None) - if None, assumes all symbols	self
casefoldUELs	will casefold all UELs in the Container or a subset of specified symbols, can be chained with other *UELs string operations	symbols=None (str, list, None) - if None, assumes all symbols	self
countDomainViolations	get the count of how many records contain at least one domain violation for symbols (if symbols=None assume all symbols)	symbols=None (str, list, None)	dict
countDuplicateRecords	returns the count of how many duplicate records exist for symbols (if symbols=None assume all symbols)	symbols=None (str, list, None)	dict
describeAliases	create a summary table with descriptive statistics for Aliases	symbols=None (None, str, list) - if None, assumes all aliases	pandas.DataFrame
describEquations	create a summary table with descriptive statistics for Equations	symbols=None (None, str, list) - if None, assumes all equations	pandas.DataFrame
describeParameters	create a summary table with descriptive statistics for Parameters	symbols=None (None, str, list) - if None, assumes all parameters	pandas.DataFrame
describeSets	create a summary table with descriptive statistics for Sets	symbols=None (None, str, list) - if None, assumes all sets	pandas.DataFrame
describeVariables	create a summary table with descriptive statistics for Variables	symbols=None (None, str, list) - if None, assumes all variables	pandas.DataFrame

Method	Description	Arguments/Defaults	Returns
dropDomainViolations	drop records that have domain violations for symbols (if symbols=None assume all symbols)	symbols=None (str, list, None)	None
dropDuplicateRecords	drop records with duplicate domains from symbols in the Container – keep argument can take values of "first" (keeps the first instance of a duplicate record), "last" (keeps the last instance of a record), or False (drops all duplicates including the first and last)	symbols=None (str, list, None) - if None, assumes all symbols keep="first" (str, False)	None
getAliases	return all alias objects in the container (is_valid=None), return all valid alias objects (is_valid=True), return all invalid alias objects (is_valid=False) in the container	is_valid=None (bool, None)	list
getDomainViolations	gets domain violations that exist in the data for symbols (if symbols=None assume all symbols); returns a list of DomainViolation objects (or None if no violations)	symbols=None (str, list, None)	list or None
getEquations	return all equation objects (is_valid=None), return all valid equation objects (is_valid=True), return all invalid equation objects (is_valid=False) in the container	is_valid=None (bool, None) types=None (list of equation types) - if None, assumes all types	list
getParameters	return all parameter objects in the container (is_valid=None), return all valid parameter objects (is_valid=True), return all invalid parameter objects (is_valid=False) in the container	is_valid=None (bool, None)	list

Method	Description	Arguments/Defaults	Returns
getSets	return all set objects in the container (is_valid=None), return all valid set objects (is_valid=True), return all invalid set objects (is_valid=False) in the container	is_valid=None (bool, None)	list
getSymbols	returns a list of object references for symbols	symbols (str, list)	list
getUELs	gets UELs from all symbols. If symbols=None and ignore_unused=False, return the full universe set. If symbols=None and ignore_unused=True, return a universe set that contains UELs that only appear in data.	symbols=None (str, list, None) ignore_unused=False (bool)	list
getVariables	return all variable objects (is_valid=None), return all valid variable objects (is_valid=True), return all invalid variable objects (is_valid=False) in the container	is_valid=None (bool, None) types=None (list of variable types) - if None, assumes all types	list
hasDomainViolations	returns True if there are domain violations in the records for symbols (if symbols=None assume all symbols), returns False if not.	symbols=None (str, list, None)	bool
hasDuplicateRecords	returns True if there are duplicate records for symbols (if symbols=None assume all symbols), False if not.	symbols=None (str, list, None)	bool
isValid	True if symbols in the Container are valid (if symbols=None assume all symbols)	symbols=None (str, list, None)	bool
ljustUELs	will left justify all UELs in the Container or a subset of specified symbols, can be chained with other *UELs string operations	length (int) fill_character=None - if None, assumes " " symbols=None (int, list, or None) - if None, assumes all symbols	self

Method	Description	Arguments/Defaults	Returns
`listAliases`	list all aliases (`is_valid=None`), list all valid aliases (`is_valid=True`), list all invalid aliases (`is_valid=False`) in the container	`is_valid=None` (`bool`, `None`)	`list`
`listEquations`	list all equations (`is_valid=None`), list all valid equations (`is_valid=True`), list all invalid equations (`is_valid=False`) in the container	`is_valid=None` (`bool`, `None`) `types=None` (`list` of equation types) - if `None`, assumes all types	`list`
`listParameters`	list all parameters (`is_valid=None`), list all valid parameters (`is_valid=True`), list all invalid parameters (`is_valid=False`) in the container	`is_valid=None` (`bool`, `None`)	`list`
`listSets`	list all sets (`is_valid=None`), list all valid sets (`is_valid=True`), list all invalid sets (`is_valid=False`) in the container	`is_valid=None` (`bool`, `None`)	`list`
`listSymbols`	list all symbols (`is_valid=None`), list all valid symbols (`is_valid=True`), list all invalid symbols (`is_valid=False`) in the container	`is_valid=None` (`bool`, `None`)	`list`
`listVariables`	list all variables (`is_valid=None`), list all valid variables (`is_valid=True`), list all invalid variables (`is_valid=False`) in the container	`is_valid=None` (`bool`, `None`) `types=None` (`list` of variable types) - if `None`, assumes all types	`list`
`lowerUELs`	will lowercase all UELs in the `Container` or a subset of specified `symbols`, can be chained with other `*UELs` string operations	`symbols=None` (`str`, `list`, `None`) - if `None`, assumes all symbols	`self`
`lstripUELs`	will left strip whitespace from all UELs in the `Container` or a subset of specified `symbols`, can be chained with other `*UELs` string operations	`symbols=None` (`str`, `list`, `None`) - if `None`, assumes all symbols	`self`

Method	Description	Arguments/Defaults	Returns
read	main method to read load_from, can be provided with a list of symbols to read in subsets, records controls if symbol records are loaded or just metadata, mode controls if data is first read into categorical data structures or object type structures, encoding specifies the original file encoding and to properly decode special characters	load_from (str or PathLike object, GMD Object Handle, GamsDatabase object, Container) symbols="all" (str, list) records=True (bool) mode="category" (category or string) encoding=None (str or None)	None
removeSymbols	symbols to remove from the Container, also sets the symbols container to None. If symbols=None, will remove all symbols.	symbols=None (str, list, None)	None
removeUELs	removes UELs from all symbols in all dimensions. If uels is None only unused UELs will be removed. If symbols is None UELs will be removed from all symbols.	uels (str, list, None) symbols=None (str, list, None)	None
renameSymbol	rename a symbol in the Container	old_name (str), new_name (str)	None
renameUELs	renames UELs (case-sensitive) that appear in symbols (for all dimensions). If symbols=None, rename UELs in all symbols. If allow_merge=True, the categorical object will be re-created to offer additional data flexibility. ** All trailing whitespace is trimmed **	uels (dict) symbols=None (str, list, None) allow_merge=False (bool)	None
reorderSymbols	reorder symbols in order to avoid domain violations	-	None
rjustUELs	will right justify all UELs in the Container or a subset of specified symbols, can be chained with other *UELs string operations	length (int) fill_character=None - if None, assumes " " symbols=None (int, list, or None) - if None, assumes all symbols	self

Method	Description	Arguments/Defaults	Returns
rstripUELs	will right strip whitespace from all UELs in the Container or a subset of specified symbols, can be chained with other *UELs string operations	symbols=None (str, list, None) - if None, assumes all symbols	self
stripUELs	will strip whitespace from all UELs in the Container or a subset of specified symbols, can be chained with other *UELs string operations	symbols=None (str, list, None) - if None, assumes all symbols	self
titleUELs	will title (capitalize all individual words) in all UELs in the Container or a subset of specified symbols, can be chained with other *UELs string operations	symbols=None (str, list, None) - if None, assumes all symbols	self
upperUELs	will uppercase all UELs in the Container or a subset of specified symbols, can be chained with other *UELs string operations	symbols=None (str, list, None) - if None, assumes all symbols	self
write	main bulk write method to a write_to target	write_to (str, PathLike object, GamsDatabase object, GMD Object) symbols=None (None, str, list) - if None, assumes all symbols compress=False (bool) uel_priority=None (str, list) merge_symbols=None (None, str, list) mode=None (None, string, category) - if None, assumes string mode writing eps_to_zero=True (bool) - default behavior will convert all -0.0 (EPS) values into 0.0 (drop the sign bit)	None

Set

There are two different ways to create a GAMS set and add it to a Container.

1. Use `Set` constructor
2. Use the `Container` method `addSet` (which internally calls the `Set` constructor)

Constructor

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the `Container` object that the symbol is being added to	Yes	-
`description`	`str`	Description of symbol	No	""
`domain`	`list`, `str`, or `Set/Alias`	List of domains given either as string ('*' for universe set) or as reference to a Set/Alias object	No	`["*"]`
`domain_forwarding`	`bool` or `list`	Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of `bool` to control which domains to forward.	No	False
`is_singleton`	`bool`	Indicates if set is a singleton set (`True`) or not (`False`)	No	False
`name`	`str`	Name of symbol	Yes	-
`records`	many	Symbol records	No	None
`uels_on_axes`	`bool`	Instructs `setRecords` to assume symbol domain information is contained in the axes of the Pandas object	No	False

Note

Set records can be updated through the object constructor (a new object will not be created) if a symbol of the same name already exists in the container, has the same domain, and has the same `is_singleton` and `domain_forwarding` state. The symbol description will only be updated if new text is provided.

Properties

Property	Description	Type	Special Setter Behavior
`container`	reference to the `Container` that the symbol belongs to	`Container`	-
`description`	description of symbol	`str`	-

Property	Description	Type	Special Setter Behavior
`dimension`	dimension of symbol	`int`	setting is a shorthand notation to create `["*"] * n` domains in symbol
`domain`	list of domains given either as string (* for universe set) or as reference to the Set/Alias object	`list`, `str`, or `Set/Alias`	-
`domain_forwarding`	flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of `bool` to control which domains to forward.	`bool` or `list`	no effect after records have been set
`domain_labels`	column headings for the `records` DataFrame	`list` of `str`	will add a `_<dimension>` tag to user supplied column names (if not unique)
`domain_names`	string version of domain names	`list` of `str`	-
`domain_type`	`none`, `relaxed` or `regular` depending on state of domain links	`str`	-
`is_singleton`	`bool` if symbol is a singleton set	`bool`	-
`modified`	Flag that identifies if the `Set` has been modified	`bool`	-
`name`	name of symbol	`str`	sets the GAMS name of the symbol
`number_records`	number of symbol records (i.e., returns `len(self.records)` if not `None`)	`int`	-
`records`	the main symbol records	`pandas.DataFrame`	responsive to `domain_forwarding` state
`summary`	output a `dict` of only the metadata	`dict`	-

Methods

Method	Description	Arguments/Defaults	Returns
`addUELs`	adds UELs to the symbol `dimensions`. If `dimensions` is `None` then add UELs to all dimensions. ** All trailing whitespace is trimmed **	`uels` (`str`, `list`) `dimensions=None` (`int`, `list`, `None`)	`None`

Method	Description	Arguments/Defaults	Returns
capitalizeUELs	will capitalize all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
casefoldUELs	will casefold all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
countDomainViolations	returns the count of how many records contain at least one domain violation	-	int
countDuplicateRecords	returns the count of how many (case insensitive) duplicate records exist	-	int
dropDomainViolations	drop records from the symbol that contain a domain violation	-	None
dropDuplicateRecords	drop records with (case insensitive) duplicate domains from the symbol – keep argument can take values of "first" (keeps the first instance of a duplicate record), "last" (keeps the last instance of a record), or False (drops all duplicates including the first and last)	keep="first" (str, False)	None
equals	Used to compare the symbol to another symbol. If check_uels=True then check both used and unused UELs and confirm same order, otherwise only check used UELs in data and do not check UEL order. If check_element_text=True then check that all set elements have the same descriptive element text, otherwise skip. If check_meta_data=True then check that symbol name and description are the same, otherwise skip. If verbose=True will return an exception from the asserter describing the nature of the difference.	check_uels=True (bool) check_element_text=True (bool) check_meta_data=True (bool) verbose=False (bool)	bool

Method	Description	Arguments/Defaults	Returns
`findDomainViolations`	get a view of the records DataFrame that contain any domain violations	-	`pandas.DataFrame`
`findDuplicateRecords`	get a view of the records DataFrame that contain any (case insensitive) duplicate domains – `keep` argument can take values of "first" (finds all duplicates while keeping the first instance as unique), "last" (finds all duplicates while keeping the last instance as unique), or `False` (finds all duplicates)	`keep="first"` (str, `False`)	`pandas.DataFrame`
`generateRecords`	convenience method to set standard `pandas.DataFrame` formatted records given domain set information. Will generate records with the Cartesian product of all domain sets. The `density` argument can take any value on the interval `[0,1]`. If `density` is <1 then randomly selected records will be removed. `density` will accept a `list` of length `dimension` -- allows users to specify a density per symbol dimension. Random number state can be set with `seed` argument.	`density=1.0` (`float`, `list`) `seed=None` (`int`, `None`)	`None`
`getDomainViolations`	returns a `list` of `DomainViolation` objects if any (`None` otherwise)	-	`list` or `None`
`getSparsity`	get the sparsity of the symbol w.r.t the cardinality	-	`float`

Method	Description	Arguments/Defaults	Returns
getUELs	gets UELs from symbol dimensions. If dimensions is None then get UELs from all dimensions (maintains order). The argument codes accepts a list of str UELs and will return the corresponding int; must specify a single dimension if passing codes. Returns only UELs in the data if ignore_unused=True, otherwise return all UELs.	dimensions=None (int, list, None) codes=None (int, list, None) ignore_unused=False (bool)	list
hasDomainViolations	returns True if there are domain violations in the records, returns False if not.	-	bool
hasDuplicateRecords	returns True if there are (case insensitive) duplicate records in the symbol, returns False if not.	-	bool
isValid	checks if the symbol is in a valid format, throw exceptions if verbose=True, recheck a symbol if force=True	verbose=False force=True	bool
ljustUELs	will left justify all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	length (int) fill_character=None - if None, assumes " " dimensions=None (int, list, or None) - if None, assumes all symbols	self
lowerUELs	will lowercase all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
lstripUELs	will left strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self

Method	Description	Arguments/Defaults	Returns
pivot	Convenience function to pivot records into a new shape (only symbols with >1D can be pivoted). If index is None then it is set to dimensions [0..dimension-1]. If columns is None then it is set to the last dimension. Missing values in the pivot will take the value provided by fill_value	index=None (str, list, None) columns=None (str, list, None) fill_value=None (int, float, str)	pd.DataFrame
renameUELs	renames UELs (case-sensitive) that appear in the symbol dimensions. If dimensions is None then operate on all dimensions of the symbol. If allow_merge=True, the categorical object will be re-created to offer additional data flexibility. ** All trailing whitespace is trimmed **	uels (str, list, dict) dimensions (int, list, None) allow_merge=False (bool)	None
reorderUELs	reorders the UELs in the symbol dimensions. If uels is None, reorder UELs to data order and append any unused categories. If dimensions is None then reorder UELs in all dimensions of the symbol.	uels (str, list, dict, None) dimensions (int, list, None)	None
removeUELs	removes UELs that appear in the symbol dimensions, If uels is None then remove all unused UELs (categories). If dimensions is None then operate on all dimensions.	uels=None (str, list, None) dimensions=None (int, list, None)	bool
rjustUELs	will right justify all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	length (int) fill_character=None - if None, assumes " " dimensions=None (int, list, or None) - if None, assumes all symbols	self
rstripUELs	will right strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self

Method	Description	Arguments/Defaults	Returns
`setRecords`	main convenience method to set standard `pandas.DataFrame` formatted records. If `uels_on_axes=True` `setRecords` will assume that all domain information is contained in the axes of the pandas object – data will be flattened (if necessary).	`records` (many types)	`None`
`setUELs`	set the UELs for symbol `dimensions`. If `dimensions` is `None` then set UELs for all dimensions. If `rename=True`, then the old UEL names will be renamed with the new UEL names. ** All trailing whitespace is trimmed **	`uels` (`str`, `list`) `dimensions=None` (`int`, `list`, `None`) `rename=False` (`bool`)	`None`
`stripUELs`	will strip whitespace from all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`titleUELs`	will title (capitalize all individual words) in all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`toList`	convenience method to return symbol records as a Python list	`include_element_text=False` (bool)	`list`
`upperUELs`	will uppercase all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`

Adding Set Records

Three possibilities exist to assign symbol records to a set (roughly ordered in complexity):

1. Setting the argument `records` in the set constructor/container method (internally calls `setRecords`) - creates a data copy
2. Using the symbol method `setRecords` - creates a data copy

3. Setting the property `records` directly - does not create a data copy

If the data is in a convenient format, a user may want to pass the records directly within the set constructor. This is an optional keyword argument and internally the set constructor will simply call the `setRecords` method. The symbol method `setRecords` is a convenience method that transforms the given data into an approved Pandas DataFrame format (see Standard Data Formats). Many native Python data types can be easily transformed into DataFrames, so the `setRecords` method for `Set` objects will accept a number of different types for input. The `setRecords` method is called internally on any data structure that is passed through the `records` argument. We show a few examples of ways to create differently structured sets:

Example #1 - Create a 1D set from a list

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
# NOTE: the above syntax is equivalent to -
# i = gt.Set(m, "i")
# i.setRecords(["seattle", "san-diego"])
# NOTE: the above syntax is also equivalent to -
# m.addSet("i", records=["seattle", "san-diego"])
# NOTE: the above syntax is also equivalent to -
# i = m.addSet("i")
# i.setRecords(["seattle", "san-diego"])
# NOTE: the above syntax is also equivalent to -
# m.addSet("i")
# m["i"].setRecords(["seattle", "san-diego"])
In [1]: i.records
Out[1]:
       uni   element_text
0    seattle
1  san-diego
```

Example #2 - Create a 1D set from a tuple

```
import gams.transfer as gt
m = gt.Container()
j = gt.Set(m, "j", records=("seattle", "san-diego"))
# NOTE: the above syntax is equivalent to -
# j = gt.Set(m, "j")
# j.setRecords(("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# m.addSet("j", records=("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# j = m.addSet("j")
# j.setRecords(("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# m.addSet("j")
# m["j"].setRecords(("seattle", "san-diego"))
In [1]: j.records
Out[1]:
       uni   element_text
0    seattle
1  san-diego
```

Example #3 - Create a 2D set from a list of tuples

```
import gams.transfer as gt
m = gt.Container()
k = gt.Set(m, "k", ["*", "*"], records=[("seattle", "san-diego")])
# NOTE: the above syntax is equivalent to -
# k = gt.Set(m, "k", ["*", "*"])
# k.setRecords([("seattle", "san-diego")])
# NOTE: the above syntax is also equivalent to -
# m.addSet("k", ["*","*"], records=[("seattle", "san-diego")])
# NOTE: the above syntax is also equivalent to -
# k = m.addSet("k", ["*","*"])
# k.setRecords([("seattle", "san-diego")])
# NOTE: the above syntax is also equivalent to -
# m.addSet("k", ["*","*"])
# m["k"].setRecords([("seattle", "san-diego")])
In [1]: k.records
Out[1]:
     uni_0      uni_1 element_text
0  seattle  san-diego
```

Example #4 - Create a 1D set from a DataFrame slice + .unique()

```
import gams.transfer as gt
m = gt.Container()
# note that the raw data is convenient to hold in a DataFrame
dist = pd.DataFrame(
    [
        ("seattle", "new-york", 2.5),
        ("seattle", "chicago", 1.7),
        ("seattle", "topeka", 1.8),
        ("san-diego", "new-york", 2.5),
        ("san-diego", "chicago", 1.8),
        ("san-diego", "topeka", 1.4),
    ],
    columns=["from", "to", "thousand_miles"],
)
l = gt.Set(m, "l", records=dist["from"].unique())
# NOTE: the above syntax is equivalent to -
# l = gt.Set(m, "l")
# l.setRecords(dist["from"].unique())
# NOTE: the above syntax is also equivalent to -
# m.addSet("l", records=dist["from"].unique())
# NOTE: the above syntax is also equivalent to -
# l = m.addSet("l")
# l.setRecords(dist["from"].unique())
# NOTE: the above syntax is also equivalent to -
# m.addSet("l")
# m["l"].setRecords(dist["from"].unique())
In [1]: l.records
Out[1]:
       uni   element_text
0    seattle
1  san-diego
```

Note

The `.unique()` method preserves the order of appearance, unlike `set()`.

Set element text is very handy when labeling specific set elements within a set. A user can add a set element text directly with a set element. Note that it is not required to label all set elements, as can be seen in the following example.

Example #5 - Add set element text

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(
    m,
    "i",
    records=[
        ("seattle", "home of sub pop records"),
        ("san-diego",),
        ("washington_dc", "former gams hq"),
    ],
)
# NOTE: the above syntax is equivalent to -
#
# i = gt.Set(m, "i")
# i_recs = [
#     ("seattle", "home of sub pop records"),
#     ("san-diego",),
#     ("washington_dc", "former gams hq"),
# ]
#
# i.setRecords(i_recs)
# NOTE: the above syntax is also equivalent to -
# m.addSet("i", records=i_recs)
# NOTE: the above syntax is also equivalent to -
# i = m.addSet("i")
# i.setRecords(i_recs)
# NOTE: the above syntax is also equivalent to -
# m.addSet("i")
# m["i"].setRecords(i_recs)
In [1]: i.records
Out[1]:
           uni               element_text
0        seattle  home of sub pop records
1      san-diego
2  washington_dc           former gams hq
```

Example #6 - Create a 1D set from a Pandas Series

```
import gams.transfer as gt
import pandas as pd
s = pd.Series(index=["a", "b"])
m = gt.Container()
i = gt.Set(m, "i", records=s, uels_on_axes=True)
# NOTE: We pass the uels_on_axes=True argument in order to tell setRecords to reshape the data
In [1]: i.records
Out[1]:
  uni element_text
0   a
1   b
# Now let's add in some element_text
s = pd.Series(index=["a", "b"], data=["node_1", "node_2"])
m = gt.Container()
i = gt.Set(m, "i", records=s, uels_on_axes=True)
In [2]: i.records
Out[2]:
  uni element_text
0   a       node_1
1   b       node_2
# If uels_on_axes=False, setRecords will attempt to create a categorical data structure from s.values -- which cannot be
      done successfully if there are NaN values in s.values.
s = pd.Series(index=["a", "b"])
m = gt.Container()
i = gt.Set(m, "i", records=s)
ValueError: Categorical categories cannot be null
```

Example #7 - Create a 2D set from a Pandas Series

```
s = pd.Series(
   index=pd.MultiIndex.from_tuples([("a", "b"), ("c", "d")]),
   data=["link_1", "link_2"],
)
m = gt.Container()
i = gt.Set(m, "i", ["*", "*"], records=s, uels_on_axes=True)
# Here is the raw pandas.Series object
In [1]: s
Out[1]:
a  b    link_1
c  d    link_2
dtype: object
# When setting records the pandas.Series object will get converted into a pandas.DataFrame
In [2]: i.records
Out[2]:
  uni_0 uni_1 element_text
0     a     b       link_1
1     c     d       link_2
```

Attention

> The order of the set element could be surprising depending on how the pandas MultiIndex is created. Users should take care that the order of the elements returned from the `getUELs` method is correct.

Example #8 - Create a 2D set from a Pandas DataFrame with uels_on_axes=True

```
import gams.transfer as gt
import pandas as pd
# Create some data first
dim1 = ["a", "b"]
dim2 = ["e", "f"]
dim3 = ["z", "x", "y"]
df = pd.DataFrame(
    index=dim1, columns=pd.MultiIndex.from_product([dim2, dim3]), dtype=bool
)
# now remove all set elements that contain the "z" element in the 3rd dimension
df.loc[:, (slice(None), "z")] = False
# now remove all set elements that contain the "a" element in the 1st dimension
df.loc["a", :] = False
In [1]: df
Out[1]:
       e                   f
```

```
       z      x      y      z      x      y
a  False  False  False  False  False  False
b  False   True   True  False   True   True
# create the Container and add records
m = gt.Container()
i = gt.Set(m, "i", ["*"] * 3, records=df, uels_on_axes=True)
In [2]: i.records
Out[2]:
  uni_0 uni_1 uni_2 element_text
0     b     e     x
1     b     e     y
2     b     f     x
3     b     f     y
```

Note

It not possible to set `element_text` when `uels_on_axes=True`

Directly Set Records

The primary advantage of the `setRecords` method is that `transfer` will convert many different (and convenient) data types into the standard data format (a Pandas DataFrame). Users that require higher performance will want to directly pass the `Container` a reference to a valid Pandas DataFrame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process and it avoids making a copy of the data in memory. In this section we walk the user through an example of how to set records directly.

Example #1 - Directly set records (1D set)

```
import gams.transfer as gt
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", description="supply")
# create a standard format dataframe
df_i = pd.DataFrame(
    data=[("seattle", ""), ("san-diego", "")], columns=["uni", "element_text"]
)
# need to create categorical column type, referencing elements already in df_i
df_i["uni"] = df_i["uni"].astype(
    pd.CategoricalDtype(categories=df_i["uni"].unique(), ordered=True)
)
# set the records directly
i.records = df_i
In [1]: i.isValid()
Out[1]: True
```

Stepping through this example we take the following steps:

1. Create an empty `Container`
2. Create a GAMS set `i` in the Container, but do not set the `records`
3. Create a Pandas DataFrame (manually, in this example) taking care to follow the standard format
4. The DataFrame has the right shape and column labels so we can proceed to set the records.
5. We need to cast the `uni` column as a `categorical` data type, so we create a custom ordered categorty type using `pandas.CategoricalDtype`
6. Finally, we set the records directly by passing a reference to `df_i` into the symbol records attribute. The setter function of `.records` checks that a DataFrame is being set, but does not check validity. Thus, as a final step we call the `.isValid()` method to verify that the symbol is valid.

Attention

> Users can debug their DataFrames by running `<symbol_name>.isValid(verbose=True)` to get feedback about their data.

Example #2 - Directly set records (1D subset)

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"], description="supply")
j = gt.Set(m, "j", i, description="supply")
# create a standard format dataframe
df_j = pd.DataFrame(data=[("seattle", "")], columns=["i", "element_text"])
# create the categorical column type
df_j["i"] = df_j["i"].astype(i.records["uni"].dtype)
# set the records
j.records = df_j
In [1]: j.isValid()
Out[1]: True
```

This example is more subtle in that we want to create a set `j` that is a subset of `i`. We create the set `i` using the `setRecords` method but then set the records directly for `j`. There are two important details to note: 1) the column labels in `df_j` now reflect the standard format for a symbol with a domain set (as opposed to the universe) and 2) we create the categorical dtype by referencing the parent set (`i`) for the categories (instead of referencing itself).

Generate Set Records

Generating the initial `pandas.DataFrame` object could be difficult for `Set` symbols that have a large number of records and a small number of UELs – these higher dimensional symbols will benefit from the `generateRecords` convenience function. Internally, `generateRecords` computes the dense Cartesian product of all the domain sets that define a symbol (`generateRecords` will only work on symbols where `<symbol>.domain_type == "regular"`).

Example #1 - Create a large (dense) 4D set

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Set(m, "a", [i, j, k, l])
# generate the records
a.generateRecords()
In [1]: a.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
         i    j    k    l   element_text
0         i0   j0   k0   l0
1         i0   j0   k0   l1
2         i0   j0   k0   l2
3         i0   j0   k0   l3
4         i0   j0   k0   l4
...      ...  ...  ...  ...          ...
6249995  i49  j49  k49  l45
6249996  i49  j49  k49  l46
6249997  i49  j49  k49  l47
6249998  i49  j49  k49  l48
6249999  i49  j49  k49  l49
[6250000 rows x 5 columns]
```

It is also possible to generate a sparse set (randomly selected rows are removed from the dense dataframe) with the `density` argument to `generateRecords`.

Example #2 - Create a large (sparse) 4D set

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Set(m, "a", [i, j, k, l])
# generate the records
a.generateRecords(density=0.05)
In [1]: a.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
        i    j    k    l   element_text
0        i0   j0   k1   l4
1        i0   j0   k1  l13
2        i0   j0   k1  l19
3        i0   j0   k1  l23
4        i0   j0   k2   l1
...     ...  ...  ...  ...          ...
312495  i49  j49  k48  l27
312496  i49  j49  k48  l30
312497  i49  j49  k49   l7
312498  i49  j49  k49  l32
312499  i49  j49  k49  l42
[312500 rows x 5 columns]
```

Example #3 - Create a large 4D set w/ only 1 sparse dimension

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Set(m, "a", [i, j, k, l])
# generate the records
a.generateRecords(density=[1, 0.05, 1, 1])
In [1]: a.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
        i    j    k    l   element_text
0        i0  j22   k0   l0
1        i0  j22   k0   l1
2        i0  j22   k0   l2
3        i0  j22   k0   l3
4        i0  j22   k0   l4
...     ...  ...  ...  ...          ...
249995  i49  j36  k49  l45
249996  i49  j36  k49  l46
249997  i49  j36  k49  l47
249998  i49  j36  k49  l48
249999  i49  j36  k49  l49
[250000 rows x 5 columns]
```

Parameter

There are two different ways to create a GAMS parameter and add it to a `Container`.

1. Use `Parameter` constructor
2. Use the `Container` method `addParameter` (which internally calls the `Parameter` constructor)

Constructor

Constructor Arguments

Argument	Type	Description	Required	Default
container	Container	A reference to the Container object that the symbol is being added to	Yes	-
description	str	Description of symbol	No	""
domain	list, str, or Set/Alias	List of domains given either as string ('*' for universe set) or as reference to a Set/Alias object, an empty domain list will create a scalar parameter	No	[]
domain_forwarding	bool or list	Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of bool to control which domains to forward.	No	False
name	str	Name of symbol	Yes	-
records	many	Symbol records	No	None
uels_on_axes	bool	Instructs setRecords to assume symbol domain information is contained in the axes of the Pandas object	No	False

Note

Parameter records can be updated through the object constructor (a new object will not be created) if a symbol of the same name already exists in the container, has the same domain, and has the same domain_forwarding state. The symbol description will only be updated if new text is provided.

Properties

Property	Description	Type	Special Setter Behavior
container	reference to the Container that the symbol belongs to	Container	-
description	description of symbol	str	-
dimension	dimension of symbol	int	setting is a shorthand notation to create ["*"] * n domains in symbol
domain	list of domains given either as string (* for universe set) or as reference to the Set/Alias object	list, str, or Set/Alias	-

Property	Description	Type	Special Setter Behavior
`domain_forwarding`	flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of `bool` to control which domains to forward.	`bool` or `list`	no effect after records have been set
`domain_labels`	column headings for the `records` DataFrame	`list` of `str`	will add a `_<dimension>` tag to user supplied column names (if not unique)
`domain_names`	string version of domain names	`list` of `str`	-
`domain_type`	`none`, `relaxed` or `regular` depending on state of domain links	`str`	-
`is_scalar`	True if the `len(self.domain) = 0`	`bool`	-
`modified`	Flag that identifies if the `Parameter` has been modified	`bool`	-
`name`	name of symbol	`str`	sets the GAMS name of the symbol
`number_records`	number of symbol records (i.e., returns `len(self.records)` if not `None`)	`int`	-
`records`	the main symbol records	`pandas.DataFrame`	responsive to `domain_forwarding` state
`shape`	a tuple describing the array dimensions if `records` were converted with `.toDense()`	`tuple`	-
`summary`	output a `dict` of only the metadata	`dict`	-

Methods

Method	Description	Arguments/Defaults	Returns
`addUELs`	adds UELs to the symbol `dimensions`. If `dimensions` is `None` then add UELs to all dimensions. ** All trailing whitespace is trimmed **	`uels` (`str`, `list`) `dimensions=None` (`int`, `list`, `None`)	`None`

Method	Description	Arguments/Defaults	Returns
`capitalizeUELs`	will capitalize all UELs in the symbol or a subset of specified `dimensions`, can be chained with other `*UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`casefoldUELs`	will casefold all UELs in the symbol or a subset of specified `dimensions`, can be chained with other `*UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`countDomainViolations`	returns the count of how many records contain at least one domain violation	-	`int`
`countDuplicateRecords`	returns the count of how many (case insensitive) duplicate records exist	-	`int`
`countEps`	total number of `SpecialValues.EPS` in `value` column	-	`int` or `None`
`countNA`	total number of `SpecialValues.NA` in `value` column	-	`int` or `None`
`countNegInf`	total number of `SpecialValues.NEGINF` in `value` column	-	`int` or `None`
`countPosInf`	total number of `SpecialValues.POSINF` in `value` column	-	`int` or `None`
`countUndef`	total number of `SpecialValues.UNDEF` in `value` column	-	`int` or `None`
`dropDefaults`	an alias to `dropZeros`	-	`None`
`dropDomainViolations`	drop records from the symbol that contain a domain violation	-	`None`
`dropDuplicateRecords`	drop records with (case insensitive) duplicate domains from the symbol – `keep` argument can take values of "first" (keeps the first instance of a duplicate record), "last" (keeps the last instance of a record), or `False` (drops all duplicates including the first and last)	`keep="first"` (str, `False`)	`None`
`dropEps`	drop records from the symbol that are `GAMS EPS` (zero `0.0` records will be retained)	-	`None`

Method	Description	Arguments/Defaults	Returns
`dropMissing`	drop records from the symbol that are `NaN` (includes both `NA` and `Undef` special values)	-	`None`
`dropNA`	drop records from the symbol that are `GAMS NA`	-	`None`
`dropUndef`	drop records from the symbol that are `GAMS Undef`	-	`None`
`dropZeros`	drop records from the symbol that are zero (GAMS EPS (`-0.0`) will not be dropped)	-	`None`
`equals`	Used to compare the symbol to another symbol. If `check_uels=True` then check both used and unused UELs and confirm same order, otherwise only check used UELs in data and do not check UEL order. If `check_meta_data=True` then check that symbol name and description are the same, otherwise skip. `rtol` (relative tolerance) and `atol` (absolute tolerance) set equality tolerances. If `verbose=True` will return an exception from the asserter describing the nature of the difference.	`check_uels=True` (`bool`) `check_meta_data=True` (`bool`) `rtol=0.0` (`float`, `None`) `atol=0.0` (`float`, `None`) `verbose=False` (`bool`)	`bool`
`findDomainViolations`	get a view of the records DataFrame that contain any domain violations	-	`pandas.DataFrame`
`findDuplicateRecords`	get a view of the records DataFrame that contain any (case insensitive) duplicate domains – `keep` argument can take values of "first" (finds all duplicates while keeping the first instance as unique), "last" (finds all duplicates while keeping the last instance as unique), or `False` (finds all duplicates)	`keep="first"` (str, `False`)	`pandas.DataFrame`

Method	Description	Arguments/Defaults	Returns
`findEps`	find positions of `SpecialValues.EPS` in `value` column	-	`pandas.DataFrame` or `None`
`findNA`	find positions of `SpecialValues.NA` in `value` column	-	`pandas.DataFrame` or `None`
`findNegInf`	find positions of `SpecialValues.NEGINF` in `value` column	-	`pandas.DataFrame` or `None`
`findPosInf`	find positions of `SpecialValues.POSINF` in `value` column	-	`pandas.DataFrame` or `None`
`findUndef`	find positions of `SpecialValues.Undef` in `value` column	-	`pandas.DataFrame` or `None`
`generateRecords`	convenience method to set standard `pandas.DataFrame` formatted records given domain set information. Will generate records with the Cartesian product of all domain sets. The `density` argument can take any value on the interval `[0,1]`. If `density` is <1 then randomly selected records will be removed. `density` will accept a `list` of length `dimension` -- allows users to specify a density per symbol dimension. Random number state can be set with `seed` argument.	`density=1.0` (`float`, `list`) `func=numpy.random.uniform(0,1)` (callable) `seed=None` (`int`, `None`)	`None`
`getMaxAbsValue`	get the maximum absolute value in `value` column	-	`float` or `None`
`getMaxValue`	get the maximum value in `value` column	-	`float` or `None`
`getMeanValue`	get the mean value in `value` column	-	`float` or `None`
`getMinValue`	get the minimum value in `value` column	-	`float` or `None`
`getSparsity`	get the sparsity of the symbol w.r.t the cardinality	-	`float`

Method	Description	Arguments/Defaults	Returns
getUELs	gets UELs from symbol dimensions. If dimensions is None then get UELs from all dimensions (maintains order). The argument codes accepts a list of str UELs and will return the corresponding int; must specify a single dimension if passing codes. Returns only UELs in the data if ignore_unused=True, otherwise return all UELs.	dimensions=None (int, list, None) codes=None (int, list, None) ignore_unused=False (bool)	list
hasDomainViolations	returns True if there are domain violations in the records, returns False if not.	-	bool
hasDuplicateRecords	returns True if there are (case insensitive) duplicate records in the symbol, returns False if not.	-	bool
isValid	checks if the symbol is in a valid format, throw exceptions if verbose=True, recheck a symbol if force=True	verbose=False force=True	bool
ljustUELs	will left justify all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	length (int) fill_character=None - if None, assumes " " dimensions=None (int, list, or None) - if None, assumes all symbols	self
lowerUELs	will lowercase all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
lstripUELs	will left strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self

Method	Description	Arguments/Defaults	Returns
pivot	Convenience function to pivot records into a new shape (only symbols with >1D can be pivoted). If index is None then it is set to dimensions [0..dimension-1]. If columns is None then it is set to the last dimension. Missing values in the pivot will take the value provided by fill_value	index=None (str, list, None) columns=None (str, list, None) fill_value=None (int, float, str)	pd.DataFrame
removeUELs	removes UELs that appear in the symbol dimensions, If uels is None then remove all unused UELs (categories). If dimensions is None then operate on all dimensions.	uels=None (str, list, None) dimensions=None (int, list, None)	bool
renameUELs	renames UELs (case-sensitive) that appear in the symbol dimensions. If dimensions is None then operate on all dimensions of the symbol. If allow_merge=True, the categorical object will be re-created to offer additional data flexibility. ** All trailing whitespace is trimmed **	uels (str, list, dict) dimensions (int, list, None) allow_merge=False (bool)	None
reorderUELs	reorders the UELs in the symbol dimensions. If uels is None, reorder UELs to data order and append any unused categories. If dimensions is None then reorder UELs in all dimensions of the symbol.	uels (str, list, dict, None) dimensions (int, list, None)	None
rjustUELs	will right justify all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	length (int) fill_character=None - if None, assumes " " dimensions=None (int, list, or None) - if None, assumes all symbols	self

Method	Description	Arguments/Defaults	Returns
`rstripUELs`	will right strip whitespace from all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`setRecords`	main convenience method to set standard `pandas.DataFrame` records. If `uels_on_axes=True` `setRecords` will assume that all domain information is contained in the axes of the pandas object – data will be flattened (if necessary).	`records` (many types).	`None`
`setUELs`	set the UELs for symbol `dimensions`. If `dimensions` is `None` then set UELs for all dimensions. If `rename=True`, then the old UEL names will be renamed with the new UEL names. ** All trailing whitespace is trimmed **	`uels` (`str`, `list`) `dimensions=None` (`int`, `list`, `None`) `rename=False` (`bool`)	`None`
`stripUELs`	will strip whitespace from all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`titleUELs`	will title (capitalize all individual words) in all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`toDense`	convert symbol to a dense `numpy.array` format	-	`numpy.array` or `None`

Method	Description	Arguments/Defaults	Returns
`toDict`	convenience method to return symbol records as a Python dictionary, `orient` can take values `natural` or `columns` and will control the shape of the `dict`. Must use `orient="columns"` if attempting to set symbol records with `setRecords`	`orient="natural"`	`dict`
`toList`	convenience method to return symbol records as a Python list	-	`list`
`toSparseCoo`	convert symbol to a sparse COOrdinate `numpy.array` format	-	sparse matrix format or `None`
`toValue`	convenience method to return symbol records as a Python float. Only possible with scalar symbols.	-	`float`
`upperUELs`	will uppercase all UELs in the symbol or a subset of specified `dimensions`, can be chained with other `*UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`whereMax`	find the domain entry of records with a maximum value (return first instance only)	-	`list` of `str` or `None`
`whereMaxAbs`	find the domain entry of records with a maximum absolute value (return first instance only)	-	`list` of `str` or `None`
`whereMin`	find the domain entry of records with a minimum value (return first instance only)	-	`list` of `str` or `None`

Adding Parameter Records

Three possibilities exist to assign symbol records to a parameter (roughly ordered in complexity):

1. Setting the argument `records` in the set constructor/container method (internally calls `setRecords`) - creates a data copy
2. Using the symbol method `setRecords` - creates a data copy
3. Setting the property `records` directly - does not create a data copy

If the data is in a convenient format, a user may want to pass the records directly within the parameter constructor. This is an optional keyword argument and internally the parameter constructor will simply call the `setRecords` method. The symbol method `setRecords` is a convenience method that transforms the given data into an approved Pandas DataFrame format (see Standard Data Formats). Many native Python data types can be easily transformed into DataFrames, so the `setRecords` method for `Set` objects will accept a number of different types for input. The `setRecords` method is called internally on any data structure that is passed through the `records` argument. We show a few examples of ways to create differently structured parameters:

Example #1 - Create a GAMS scalar

```
import gams.transfer as gt
m = gt.Container()
pi = gt.Parameter(m, "pi", records=3.14159)
# NOTE: the above syntax is equivalent to -
# pi = gt.Parameter(m, "pi")
# pi.setRecords(3.14159)
# NOTE: the above syntax is also equivalent to -
# m.addParameter("pi", records=3.14159)
# NOTE: the above syntax is also equivalent to -
# pi = m.addParameter("pi")
# pi.setRecords(3.14159)
# NOTE: the above syntax is also equivalent to -
# m.addParameter("pi")
# m["pi"].setRecords(3.14159)
In [14]: pi.records
Out[14]:
     value
0  3.14159
```

Note

> `transfer` will still convert scalar values to a standard format (i.e., a Pandas DataFrame with a single row and column).

Example #2 - Create a 1D parameter (defined over $*$) from a list of tuples

```
import gams.transfer as gt
m = gt.Container()
i = gt.Parameter(m, "i", ["*"], records=[("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is equivalent to -
# i = gt.Parameter(m, "i")
# i.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.addParameter("i", records=[("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# i = m.addParameter("i")
# i.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.addParameter("i")
# m["i"].setRecords([("i" + str(i), i) for i in range(5)])
In [1]: i.records
Out[1]:
  uni  value
0   i0    0.0
1   i1    1.0
2   i2    2.0
3   i3    3.0
4   i4    4.0
```

Example #3 - Create a 1D parameter (defined over a set) from a list of tuples

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=["i" + str(i) for i in range(5)])
a = gt.Parameter(m, "a", i, records=[("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is equivalent to -
# i = gt.Set(m, "i")
# i.setRecords(["i" + str(i) for i in range(5)])
# a = gt.Parameter(m, "a", i)
# a.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.addSet("i", records=["i" + str(i) for i in range(5)])
# m.addParameter("a", i, records=[("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# i = m.addSet("i")
# i.setRecords(["i" + str(i) for i in range(5)])
# a = m.addParameter("a", i)
# a.setRecords([("i" + str(i), i) for i in range(5)])
# NOTE: the above syntax is also equivalent to -
# m.addSet("i")
# m["i"].setRecords(["i" + str(i) for i in range(5)])
# m.addParameter("a", i)
# m["a"].setRecords([("i" + str(i), i) for i in range(5)])
In [1]: a.records
Out[1]:
  i  value
0  i0    0.0
1  i1    1.0
2  i2    2.0
3  i3    3.0
4  i4    4.0
```

Example #4 - Create a 2D parameter (defined over a set) from a DataFrame slice

```
import gams.transfer as gt
import pandas as pd
dist = pd.DataFrame(
    [
        ("seattle", "new-york", 2.5),
        ("seattle", "chicago", 1.7),
        ("seattle", "topeka", 1.8),
        ("san-diego", "new-york", 2.5),
        ("san-diego", "chicago", 1.8),
        ("san-diego", "topeka", 1.4),
    ],
    columns=["from", "to", "thousand_miles"],
)
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=dist["from"].unique())
j = gt.Set(m, "j", ["*"], records=dist["to"].unique())
a = gt.Parameter(m, "a", [i, j], records=dist.loc[[0, 3], :])
# NOTE: the above syntax is equivalent to -
# i = gt.Set(m, "i")
# i.setRecords(dist["from"].unique())
# j = gt.Set(m, "j")
# j.setRecords(dist["to"].unique())
# a = gt.Parameter(m, "a", [i, j])
# a.setRecords(dist.loc[[0, 3], :])
# NOTE: the above syntax is also equivalent to -
# m.addSet("i", records=dist["from"].unique())
# m.addSet("j", records=dist["to"].unique())
# m.addParameter("a", i, records=dist.loc[[0, 3], :])
In [1]: a.records
Out[1]:
         i         j    value
0    seattle  new-york    2.5
3  san-diego  new-york    2.5
```

Note

The original indexing is preserved when a user slices rows out of a reference dataframe.

Example #5 - Create a 2D parameter (defined over a set) from a matrix

```
import gams.transfer as gt
import pandas as pd
dist = pd.DataFrame(
    [
        ("seattle", "new-york", 2.5),
        ("seattle", "chicago", 1.7),
        ("seattle", "topeka", 1.8),
        ("san-diego", "new-york", 2.5),
        ("san-diego", "chicago", 1.8),
        ("san-diego", "topeka", 1.4),
    ],
    columns=["from", "to", "thousand_miles"],
)
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=dist["from"].unique())
j = gt.Set(m, "j", ["*"], records=dist["to"].unique())
a = gt.Parameter(m, "a", [i, j], records=dist)
In [1]: a.toDense()
Out[1]:
array([[2.5, 1.7, 1.8],
       [2.5, 1.8, 1.4]])
# use a.toDense() to create a new (and identical) parameter a2
a2 = gt.Parameter(m, "a2", [i, j], records=a.toDense())
# check that a is identical to a2
In [1]: a.equals(a2, check_meta_data=False)
Out[1]: True
```

Example #6 - Create a 2D parameter from an array using setRecords

```
import gams.transfer as gt
import numpy as np
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", records=["j" + str(j) for j in range(5)])
# create the parameter with linked domains (these will control the .shape of the symbol)
a = gt.Parameter(m, "a", [i, j])
# here we use the .shape property to easily generate a dense random array in numpy
a.setRecords(np.random.uniform(low=1, high=10, size=a.shape))
In [1]: a.toDense()
Out[1]:
array([[3.6694495 , 5.17395381, 1.99129484, 3.28315433, 1.44793791],
       [1.06953243, 6.56331121, 5.26162554, 5.98098795, 8.30006   ],
       [3.77213221, 5.82144901, 9.30035479, 9.12534285, 8.51970747],
       [8.47965504, 7.84426304, 5.2442471 , 6.96666622, 6.55194415],
       [5.62682779, 4.92509183, 8.94579609, 2.7724934 , 9.99576081]])
```

Example #7 - Create a 1D parameter from a pandas Series

```
import gams.transfer as gt
import pandas as pd
s = pd.Series(index=["a", "b", "c"], data=[i + 1 for i in range(3)])
m = gt.Container()
i = gt.Parameter(m, "i", ["*"], records=s, uels_on_axes=True)
In [1]: i.records
Out[1]:
  uni  value
0   a    1.0
1   b    2.0
2   c    3.0
```

Example #8 - Create a 2D parameter from a pandas Series

```
import gams.transfer as gt
import pandas as pd
dim1 = ["a", "b", "c"]
dim2 = ["z", "y", "x"]
s = pd.Series(
    index=pd.MultiIndex.from_product([dim1, dim2]),
    data=[i + 1 for i in range(len(dim1) * len(dim2))],
)
m = gt.Container()
i = gt.Parameter(m, "i", ["*", "*"], records=s, uels_on_axes=True)
In [1]: i.records
```

```
Out[1]:
  uni_0 uni_1  value
0     a     z    1.0
1     a     y    2.0
2     a     x    3.0
3     b     z    4.0
4     b     y    5.0
5     b     x    6.0
6     c     z    7.0
7     c     y    8.0
8     c     x    9.0
# NOTE: the order of the second dimension is automatically put into lexicographical order by Pandas
# This can be unexpected and undesirable from a GAMS perspective.
In [2]: s.index.levels[1]
Out[2]: Index(['x', 'y', 'z'], dtype='object')
# gams.transfer goes through extra work to maintain data order in the categorical data structures
# this can be see here where "z" is mapped to integer 0, "y" is mapped to 1 and "x" is mapped to 2
In [3]: i.records["uni_1"]
Out[3]:
0    z
1    y
2    x
3    z
4    y
5    x
6    z
7    y
8    x
Name: uni_1, dtype: category
Categories (3, object): ['z' < 'y' < 'x']
In [4]: i.records["uni_1"].cat.codes
Out[4]:
0    0
1    1
2    2
3    0
4    1
5    2
6    0
7    1
8    2
dtype: int8
```

Example #9 - Create a 2D parameter from a DataFrame (uels_on_axes=True)

```
import gams.transfer as gt
import pandas as pd
import numpy as np
dim1 = [f"d{i}" for i in range(2)]
dim2 = [f"d{i}" for i in range(2)]
dim3 = [f"d{i}" for i in range(2)]
dim4 = [f"d{i}" for i in range(2)]
rng = np.random.default_rng(seed=100)
df = pd.DataFrame(
    data=rng.uniform(size=(2, 8)),
    index=dim1,
    columns=pd.MultiIndex.from_product([dim2, dim3, dim4]),
)
In [1]: df
Out[1]:
          d0                                  d1
          d0                  d1              d0                  d1
          d0        d1        d0        d1    d0        d1        d0        d1
d0  0.834982  0.596554  0.288863  0.042952  0.973654  0.596472  0.790263  0.910339
d1  0.688154  0.189991  0.981479  0.284740  0.629273  0.581036  0.599912  0.535248
m = gt.Container()
i = gt.Parameter(m, "i", ["*"] * 4, records=df, uels_on_axes=True)
In [2]: i.records
Out[2]:
   uni_0 uni_1 uni_2 uni_3     value
0     d0    d0    d0    d0  0.834982
1     d0    d0    d0    d1  0.596554
2     d0    d0    d1    d0  0.288863
3     d0    d0    d1    d1  0.042952
4     d0    d1    d0    d0  0.973654
5     d0    d1    d0    d1  0.596472
6     d0    d1    d1    d0  0.790263
7     d0    d1    d1    d1  0.910339
8     d1    d0    d0    d0  0.688154
9     d1    d0    d0    d1  0.189991
10    d1    d0    d1    d0  0.981479
11    d1    d0    d1    d1  0.284740
12    d1    d1    d0    d0  0.629273
```

```
13    d1    d1    d0    d1  0.581036
14    d1    d1    d1    d0  0.599912
15    d1    d1    d1    d1  0.535248
```

Directly Set Records

As with sets, the primary advantage of the `setRecords` method is that `transfer` will convert many different (and convenient) data types into the standard data format (a Pandas DataFrame). Users that require higher performance will want to directly pass the `Container` a reference to a valid Pandas DataFrame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process and it avoids making a copy of the data in memory. In this section we walk the user through an example of how to set records directly.

Example #1 - Correctly set records (directly)

```
import gams.transfer as gt
import pandas as pd
import numpy as np
df = pd.DataFrame(
    data=[
        ("h" + str(h), "m" + str(m), "s" + str(s))
        for h in range(8760)
        for m in range(60)
        for s in range(60)
    ],
    columns=["h", "m", "s"],
)
df["value"] = np.random.uniform(0, 100, len(df))
m = gt.Container()
hrs = gt.Set(m, "h", records=df["h"].unique())
mins = gt.Set(m, "m", records=df["m"].unique())
secs = gt.Set(m, "s", records=df["s"].unique())
df["h"] = df["h"].astype(hrs.records["uni"].dtype)
df["m"] = df["m"].astype(mins.records["uni"].dtype)
df["s"] = df["s"].astype(secs.records["uni"].dtype)
a = gt.Parameter(m, "a", [hrs, mins, secs])
# set records
a.records = df
In [1]: a.isValid()
Out[1]: True
```

In this example we create a large parameter (31,536,000 records and 8880 unique domain elements – we mimic data that is labeled for every second in one year) and assign it to a parameter with `a.records`. `transfer` requires that all domain columns must be a categorical data type, furthermore, this categorical must be ordered. The `records` setter function does very little work other than checking if the object being set is a DataFrame. This places more responsibility on the user to create a DataFrame that complies with the standard format. In Example #1 we take care to properly reference the categorical data types from the domain sets – and in the end `a.isValid() = True`.

Users will need to use the `.isValid(verbose=True)` method to debug any structural issues. As an example we incorrectly generate categorical data types by passing the DataFrame constructor the generic `dtype="category"` argument. This creates categorical column types but they are not ordered and they do not reference the underlying domain set. These errors result in `a` being invalid.

Example #2 - Incorrectly set records (directly)

```
import gams.transfer as gt
import pandas as pd
import numpy as np
df = pd.DataFrame(
    data=[
        ("h" + str(h), "m" + str(m), "s" + str(s))
        for h in range(8760)
```

```
        for m in range(60)
        for s in range(60)
    ],
    columns=["h", "m", "s"],
    dtype="category"
)
df["value"] = np.random.uniform(0, 100, len(df))
m = gt.Container()
hrs = gt.Set(m, "h", records=df["h"].unique())
mins = gt.Set(m, "m", records=df["m"].unique())
secs = gt.Set(m, "s", records=df["s"].unique())
a = gt.Parameter(m, "a", [hrs, mins, secs])
# set the records directly
a.records = df
In [1]: a.isValid()
Out[1]: False
In [2]: a.isValid(verbose=True)
Out[2]: Exception: Domain information in column 'h' for 'records' must be an ORDERED categorical type (i.e.,
      <symbol_object>.records["h"].dtype.ordered = True)
```

Generate Parameter Records

Generating the initial `pandas.DataFrame` object could be difficult for `Parameter` symbols that have a large number of records and a small number of UELs – these higher dimensional symbols will benefit from the `generateRecords` convenience function. Internally, `generateRecords` computes the dense Cartesian product of all the domain sets that define a symbol (`generateRecords` will only work on symbols where `<symbol>.domain_type == "regular"`).

Example #1 - Create a large (dense) 4D parameter

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Parameter(m, "a", [i, j, k, l])
# generate the records
a.generateRecords()
In [1]: a.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
         i    j    k    l       value
0         i0   j0   k0   l0  0.386390
1         i0   j0   k0   l1  0.671253
2         i0   j0   k0   l2  0.522057
3         i0   j0   k0   l3  0.037694
4         i0   j0   k0   l4  0.564205
...      ...  ...  ...  ...       ...
6249995  i49  j49  k49  l45  0.573354
6249996  i49  j49  k49  l46  0.033717
6249997  i49  j49  k49  l47  0.410322
6249998  i49  j49  k49  l48  0.758310
6249999  i49  j49  k49  l49  0.920708
[6250000 rows x 5 columns]
```

Note

> In Example #1 a large 4D parameter was generated – by default, the value of these records are randomly drawn numbers from the interval `[0,1]` (uniform distribution).

As with `Sets`, it is possible to generate a sparse parameter with the `density` argument to `generateRecords`. We extend this example by passing our own custom `func` argument that will control the behavior of the `value` columns. The `func` argument accepts a `callable` (i.e., a reference to a function).

Example #2 - Create a large (sparse) 4D parameter with normally distributed values

```
import gams.transfer as gt
import numpy as np
# create a custom function to pass to ‘generateRecords‘
def value_dist(size):
    return np.random.normal(loc=10.0, scale=2.3, size=size)
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Parameter(m, "a", [i, j, k, l])
# generate the records
a.generateRecords(density=0.05, func=value_dist)
In [1]: a.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
        i    j    k    l        value
0        i0   j0   k0  133  12.490579
1        i0   j0   k0  143   9.460560
2        i0   j0   k0  144   7.660337
3        i0   j0   k0  147   8.811967
4        i0   j0   k1   l5  11.103291
...     ...  ...  ...  ...        ...
312495  i49  j49  k48  138  10.619791
312496  i49  j49  k48  141  14.208250
312497  i49  j49  k48  147   6.104145
312498  i49  j49  k49   10  10.216812
312499  i49  j49  k49  139   9.739771
[312500 rows x 5 columns]
In [3]: a.records["value"].mean()
Out[3]: 10.004072307451391
In [4]: a.records["value"].std()
Out[4]: 2.292569938350144
```

Note

> The custom `callable` function reference must expose a `size` argument. It might be tedious to know the exact number of the records that will be generated, especially if a fractional density is specified; therefore, the `generateRecords` method will pass in the correct size automatically. Users are encouraged to use the Numpy suite of random distributions when generating samples – custom functions have the potential to be computationally burdensome if a symbol has a large number of records.

Example #3 - Create a large 4D parameter with 1 sparse dimension

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Parameter(m, "a", [i, j, k, l])
# generate the records
a.generateRecords(density=[1, 0.05, 1, 1])
In [1]: a.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
        i    j    k    l       value
0        i0  j30   k0   10  0.473084
1        i0  j30   k0   11  0.192571
2        i0  j30   k0   12  0.060711
3        i0  j30   k0   13  0.655477
4        i0  j30   k0   14  0.629535
...     ...  ...  ...  ...       ...
249995  i49  j32  k49  145  0.442380
249996  i49  j32  k49  146  0.002444
249997  i49  j32  k49  147  0.332731
249998  i49  j32  k49  148  0.983800
249999  i49  j32  k49  149  0.984322
[250000 rows x 5 columns]
```

Example #4 - Create a large 4D parameter with a random number seed

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Parameter(m, "a", [i, j, k, l])
a2 = gt.Parameter(m, "a2", [i, j, k, l])
# generate the records
a.generateRecords(density=0.05, seed=123)
a2.generateRecords(density=0.05)
In [1]: a.equals(a2, check_meta_data=False)
Out[1]: False
In [2]: a2.generateRecords(density=0.05, seed=123)
In [3]: a.equals(a2, check_meta_data=False)
Out[3]: True
```

Note

The `seed` is an `int` that will set the random number generator state (enables reproducible sequences of random numbers).

Variable

There are two different ways to create a GAMS variable and add it to a `Container`.

1. Use `Variable` constructor
2. Use the `Container` method `addVariable` (which internally calls the `Variable` constructor)

Constructor

Constructor Arguments

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the `Container` object that the symbol is being added to	Yes	-
`description`	`str`	Description of symbol	No	""
`domain`	`list`, `str`, or `Set/Alias`	List of domains given either as string (∗ for universe set) or as reference to a Set/Alias object, an empty domain list will create a scalar variable	No	[]
`domain_forwarding`	`bool` or `list`	Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of `bool` to control which domains to forward.	No	False

Argument	Type	Description	Required	Default
`name`	`str`	Name of symbol	Yes	-
`records`	many	Symbol records	No	None
`type`	`str`	Type of variable being created [`binary`, `integer`, `positive`, `negative`, `free`, `sos1`, `sos2`, `semicont`, `semiint`]	No	`free`
`uels_on_axes`	`bool`	Instructs `setRecords` to assume symbol domain information is contained in the axes of the Pandas object	No	False

Note

Variable records can be updated through the object constructor (a new object will not be created) if a symbol of the same name already exists in the container, has the same domain, has the same `type`, and has the same `domain_forwarding` state. The symbol description will only be updated if new text is provided.

Properties

Property	Description	Type	Special Setter Behavior
`container`	reference to the `Container` that the symbol belongs to	`Container`	-
`description`	description of symbol	`str`	-
`dimension`	dimension of symbol	`int`	setting is a shorthand notation to create `["*"] * n` domains in symbol
`domain`	list of domains given either as string (* for universe set) or as reference to the Set/Alias object	`list`, `str`, or `Set/Alias`	-
`domain_forwarding`	flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of `bool` to control which domains to forward.	`bool` or `list`	no effect after records have been set
`domain_labels`	column headings for the `records` DataFrame	`list` of `str`	will add a `_<dimension>` tag to user supplied column names (if not unique)
`domain_names`	string version of domain names	`list` of `str`	-

Property	Description	Type	Special Setter Behavior
domain_type	none, relaxed or regular depending on state of domain links	str	-
is_scalar	True if the len(self.domain) = 0	bool	-
modified	Flag that identifies if the Variable has been modified	bool	-
name	name of symbol	str	sets the GAMS name of the symbol
number_records	number of symbol records (i.e., returns len(self.records) if not None)	int	-
records	the main symbol records	pandas.DataFrame	responsive to domain_forwarding state
shape	a tuple describing the array dimensions if records were converted with .toDense()	tuple	-
summary	output a dict of only the metadata	dict	-
type	str type of variable	str	-

Methods

Method	Description	Arguments/Defaults	Returns
addUELs	adds UELs to the symbol dimensions. If dimensions is None then add UELs to all dimensions. ** All trailing whitespace is trimmed **	uels (str, list) dimensions=None (int, list, None)	None
capitalizeUELs	will capitalize all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
casefoldUELs	will casefold all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
countDomainViolations	returns the count of how many records contain at least one domain violation	-	int

Method	Description	Arguments/Defaults	Returns
countDuplicateRecords	returns the count of how many (case insensitive) duplicate records exist	-	int
countEps	total number of SpecialValues.EPS across all columns	columns="level" (str, list)	int or None
countNA	total number of SpecialValues.NA across all columns	columns="level" (str, list)	int or None
countNegInf	total number of SpecialValues.NEGINF across all columns	columns="level" (str, list)	int or None
countPosInf	total number of SpecialValues.POSINF across all columns	columns="level" (str, list)	int or None
countUndef	total number of SpecialValues.UNDEF across all columns	columns="level" (str, list)	int or None
dropDefaults	drop records that are set to GAMS default records (check .default_records property for values)	-	None
dropDomainViolations	drop records from the symbol that contain a domain violation	-	None
dropDuplicateRecords	drop records with (case insensitive) duplicate domains from the symbol – keep argument can take values of "first" (keeps the first instance of a duplicate record), "last" (keeps the last instance of a record), or False (drops all duplicates including the first and last)	keep="first" (str, False)	None
dropEps	drop records from the symbol that are GAMS EPS (zero 0.0 records will be retained)	-	None
dropMissing	drop records from the symbol that are NaN (includes both NA and Undef special values)	-	None
dropNA	drop records from the symbol that are GAMS NA	-	None
dropUndef	drop records from the symbol that are GAMS Undef	-	None

Method	Description	Arguments/Defaults	Returns
equals	Used to compare the symbol to another symbol. The columns argument allows the user to numerically compare only specified variable attributes (default is to compare all). If check_uels=True then check both used and unused UELs and confirm same order, otherwise only check used UELs in data and do not check UEL order. If check_meta_data=True then check that symbol name, description and variable type are the same, otherwise skip. rtol (relative tolerance) and atol (absolute tolerance) set equality tolerances; can be different tolerances for different variable attributes (if specified as a dict). If verbose=True will return an exception from the asserter describing the nature of the difference.	columns=["level", "marginal", "lower", "upper", "scale"] check_uels=True (bool) check_meta_data=True (bool) rtol=0.0 (int, float, None) atol=0.0 (int, float, None) verbose=False (bool)	bool
findDomainViolations	get a view of the records DataFrame that contain any domain violations	-	pandas.DataFrame
findDuplicateRecords	get a view of the records DataFrame that contain any (case insensitive) duplicate domains – keep argument can take values of "first" (finds all duplicates while keeping the first instance as unique), "last" (finds all duplicates while keeping the last instance as unique), or False (finds all duplicates)	keep="first" (str, False)	pandas.DataFrame
findEps	find positions of SpecialValues.EPS in column	column="level" (str)	pandas.DataFrame or None

Method	Description	Arguments/Defaults	Returns
`findNA`	find positions of `SpecialValues.NA` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`
`findNegInf`	find positions of `SpecialValues.NEGINF` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`
`findPosInf`	find positions of `SpecialValues.POSINF` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`
`findUndef`	find positions of `SpecialValues.Undef` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`
`generateRecords`	convenience method to set standard `pandas.DataFrame` formatted records given domain set information. Will generate records with the Cartesian product of all domain sets. The `density` argument can take any value on the interval `[0,1]`. If `density` is <1 then randomly selected records will be removed. `density` will accept a `list` of length `dimension` -- allows users to specify a density per symbol dimension. Random number state can be set with `seed` argument.	`density=1.0` (`float`, `list`) `func=numpy.random.uniform(0,1)` (`dict` of `callables`) `seed=None` (`int`, `None`)	`None`
`getMaxValue`	get the maximum value across all `columns`	`columns="level"` (`str`, `list`)	`float` or `None`
`getMinValue`	get the minimum value across all `columns`	`columns="level"` (`str`, `list`)	`float` or `None`
`getMeanValue`	get the mean value across all `columns`	`columns="level"` (`str`, `list`)	`float` or `None`
`getMaxAbsValue`	get the maximum absolute value across all `columns`	`columns="level"` (`str`, `list`)	`float`
`getSparsity`	get the sparsity of the symbol w.r.t the cardinality	-	`float`

Method	Description	Arguments/Defaults	Returns
`getUELs`	gets UELs from symbol `dimensions`. If `dimensions` is `None` then get UELs from all dimensions (maintains order). The argument `codes` accepts a list of `str` UELs and will return the corresponding `int`; must specify a single dimension if passing `codes`. Returns only UELs in the data if `ignore_unused=True`, otherwise return all UELs.	`dimensions=None` (`int`, `list`, `None`) `codes=None` (`int`, `list`, `None`) `ignore_unused=False` (`bool`)	`list`
`hasDomainViolations`	returns `True` if there are domain violations in the records, returns `False` if not.	-	`bool`
`hasDuplicateRecords`	returns `True` if there are (case insensitive) duplicate records in the symbol, returns `False` if not.	-	`bool`
`isValid`	checks if the symbol is in a valid format, throw exceptions if `verbose=True`, recheck a symbol if `force=True`	`verbose=False` `force=True`	`bool`
`ljustUELs`	will left justify all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`length` (`int`) `fill_character=None` - if `None`, assumes `" "` `dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`lowerUELs`	will lowercase all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`lstripUELs`	will left strip whitespace from all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`

Method	Description	Arguments/Defaults	Returns
pivot	Convenience function to pivot records into a new shape (only symbols with >1D can be pivoted). If index is None then it is set to dimensions [0..dimension-1]. If columns is None then it is set to the last dimension. If value is None then the level values will be pivoted. Missing values in the pivot will take the value provided by fill_value	index=None (str, list, None) columns=None (str, list, None) value (str) fill_value=None (int, float, str)	pd.DataFrame
removeUELs	removes UELs that appear in the symbol dimensions, If uels is None then remove all unused UELs (categories). If dimensions is None then operate on all dimensions.	uels=None (str, list, None) dimensions=None (int, list, None)	bool
renameUELs	renames UELs (case-sensitive) that appear in the symbol dimensions. If dimensions is None then operate on all dimensions of the symbol. If allow_merge=True, the categorical object will be re-created to offer additional data flexibility. ** All trailing whitespace is trimmed **	uels (str, list, dict) dimensions (int, list, None) allow_merge=False (bool)	None
reorderUELs	reorders the UELs in the symbol dimensions. If uels is None, reorder UELs to data order and append any unused categories. If dimensions is None then reorder UELs in all dimensions of the symbol.	uels (str, list, dict, None) dimensions (int, list, None)	None

Method	Description	Arguments/Defaults	Returns
rjustUELs	will right justify all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	length (int) fill_character=None - if None, assumes " " dimensions=None (int, list, or None) - if None, assumes all symbols	self
rstripUELs	will right strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
setRecords	main convenience method to set standard pandas.DataFrame records. If uels_on_axes=True setRecords will assume that all domain information is contained in the axes of the pandas object – data will be flattened (if necessary).	records (many types)	None
setUELs	set the UELs for symbol dimensions. If dimensions is None then set UELs for all dimensions. If rename=True, then the old UEL names will be renamed with the new UEL names. ** All trailing whitespace is trimmed **	uels (str, list) dimensions=None (int, list, None) rename=False (bool)	None
stripUELs	will strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
titleUELs	will title (capitalize all individual words) in all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
toDense	convert column to a dense numpy.array format	column="level" (str)	numpy.array or None

Method	Description	Arguments/Defaults	Returns
`toDict`	convenience method to return symbol records as a Python dictionary. `columns` will control which attributes to include in the `dict`. `orient` can take values `natural` or `columns` and will control the shape of the `dict`. Must use `orient="columns"` if attempting to set symbol records with `setRecords`.	`columns="level"` (str) `orient="natural"` (str)	`dict`
`toList`	convenience method to return symbol records as a Python list, the `columns` argument will control with attributes to include in the list	`columns="level"` (str)	`list`
`toSparseCoo`	convert `column` to a sparse COOrdinate `numpy.array` format	`column="level"` (`str`)	sparse matrix format or `None`
`toValue`	convenience method to return symbol records as a Python float. Only possible with scalar symbols. Attribute can be specified with `column` argument.	`column="level"` (str)	`float`
`upperUELs`	will uppercase all UELs in the symbol or a subset of specified `dimensions`, can be chained with other `*UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`whereMax`	find the domain entry of records with a maximum value (return first instance only)	`column="level"` (`str`)	`list` of `str` or `None`
`whereMaxAbs`	find the domain entry of records with a maximum absolute value (return first instance only)	`column="level"` (`str`)	`list` of `str` or `None`
`whereMin`	find the domain entry of records with a minimum value (return first instance only)	`column="level"` (`str`)	`list` of `str` or `None`

Adding Variable Records

Three possibilities exist to assign symbol records to a variable (roughly ordered in complexity):

1. Setting the argument `records` in the set constructor/container method (internally calls `setRecords`) - creates a data copy
2. Using the symbol method `setRecords` - creates a data copy
3. Setting the property `records` directly - does not create a data copy

If the data is in a convenient format, a user may want to pass the records directly within the variable constructor. This is an optional keyword argument and internally the variable constructor will simply call the `setRecords` method. In contrast to the `setRecords` methods in in either the Set or Parameter classes the `setRecords` method for variables will only accept Pandas DataFrames and specially structured `dict` for creating records from matrices. This restriction is out of necessity because to properly set a record for a Variable the user must pass data for the `level`, `marginal`, `lower`, `upper` and `scale` attributes. That said, any missing attributes will be filled in with the GAMS default record values (see: Variable Types), default `scale` value is always 1, and the default `level` and `marginal` values are 0 for all variable types). We show a few examples of ways to create differently structured variables:

Example #1 - Create a GAMS scalar variable

```
import gams.transfer as gt
m = gt.Container()
pi = gt.Variable(m, "pi", records=pd.DataFrame(data=[3.14159], columns=["level"]))
# NOTE: the above syntax is equivalent to -
# pi = gt.Variable(m, "pi", "free")
# pi.setRecords(pd.DataFrame(data=[3.14159], columns=["level"]))
# NOTE: the above syntax is also equivalent to -
# m.addVariable("pi", "free", records=pd.DataFrame(data=[3.14159], columns=["level"]))
In [1]: pi.records
Out[1]:
     level  marginal  lower  upper  scale
0  3.14159       0.0   -inf    inf    1.0
```

Example #2 - Create a 1D variable (defined over $*$) from a list of tuples

In this example we only set the `marginal` values.

```
import gams.transfer as gt
m = gt.Container()
v = gt.Variable(
    m,
    "v",
    "free",
    domain=["*"],
    records=pd.DataFrame(
        data=[("i" + str(i), i) for i in range(5)], columns=["domain", "marginal"]
    ),
)
In [1]: v.records
Out[1]:
  uni    level  marginal  lower  upper  scale
0    i0    0.0       0.0   -inf    inf    1.0
1    i1    0.0       1.0   -inf    inf    1.0
2    i2    0.0       2.0   -inf    inf    1.0
3    i3    0.0       3.0   -inf    inf    1.0
4    i4    0.0       4.0   -inf    inf    1.0
```

Example #3 - Create a 1D variable (defined over a set) from a list of tuples

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=["i" + str(i) for i in range(5)])
v = gt.Variable(
    m,
    "v",
    "free",
    domain=i,
```

```
    records=pd.DataFrame(
        data=[("i" + str(i), i) for i in range(5)], columns=["domain", "marginal"]
    ),
)
In [1]: v.records
Out[1]:
  i    level  marginal  lower  upper  scale
0  i0    0.0       0.0   -inf    inf    1.0
1  i1    0.0       1.0   -inf    inf    1.0
2  i2    0.0       2.0   -inf    inf    1.0
3  i3    0.0       3.0   -inf    inf    1.0
4  i4    0.0       4.0   -inf    inf    1.0
```

Example #4 - Create a 2D positive variable, specifying no numerical data

```
import gams.transfer as gt
import pandas as pd
m = gt.Container()
v = gt.Variable(
    m,
    "v",
    "positive",
    ["*", "*"],
    records=pd.DataFrame([("seattle", "san-diego"), ("chicago", "madison")]),
)
In [1]: v.records
Out[1]:
     uni_0      uni_1  level  marginal  lower  upper  scale
0  seattle  san-diego    0.0       0.0    0.0    inf    1.0
1  chicago    madison    0.0       0.0    0.0    inf    1.0
```

Example #5 - Create a 2D variable (defined over a set) from a matrix

```
import gams.transfer as gt
import pandas as pd
import numpy as np
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", ["*"], records=["j" + str(i) for i in range(5)])
a = gt.Parameter(
    m,
    "a",
    [i, j],
    records=[("i" + str(i), "j" + str(j), i + j) for i in range(5) for j in range(5)],
)
# create a free variable and set the level and marginal attributes from matricies
v = gt.Variable(
    m, "v", domain=[i, j], records={"level": a.toDense(), "marginal": a.toDense()}
)
# if not specified, the toDense() method will convert the level values to a matrix
In [1]: v.toDense()
Out[1]:
array([[0., 1., 2., 3., 4.],
       [1., 2., 3., 4., 5.],
       [2., 3., 4., 5., 6.],
       [3., 4., 5., 6., 7.],
       [4., 5., 6., 7., 8.]])
```

Example #6 - Create a 1D variable from a pandas Series

```
import gams.transfer as gt
import pandas as pd
s = pd.Series(index=["a", "b", "c"], data=[i + 1 for i in range(3)])
m = gt.Container()
v = gt.Variable(m, "v", domain=["*"], records=s, uels_on_axes=True)
In [1]: v.records
Out[1]:
  uni  level  marginal  lower  upper  scale
0   a    1.0       0.0   -inf    inf    1.0
1   b    2.0       0.0   -inf    inf    1.0
2   c    3.0       0.0   -inf    inf    1.0
```

Note

`transfer` will assume that the `level` value is being set if attributes cannot be found in the axes

Example #7 - Create a 1D variable from a pandas Series (set only the marginal)

```
import gams.transfer as gt
import pandas as pd
# NOTE: we include the "marginal" label in level 1 of the MultiIndex
s = pd.Series(
    index=pd.MultiIndex.from_product([["a", "b", "c"], ["marginal"]]),
    data=[i + 1 for i in range(3)],
)
m = gt.Container()
v = gt.Variable(m, "v", domain=["*"], records=s, uels_on_axes=True)
In [1]: v.records
Out[1]:
  uni  level  marginal  lower  upper  scale
0   a    0.0       1.0   -inf    inf    1.0
1   b    0.0       2.0   -inf    inf    1.0
2   c    0.0       3.0   -inf    inf    1.0
```

Note

`transfer` will search the axes for any variable attributes and set those attributes provided. It is necessary to include all attributes into the same axes level (if a MultiIndex) without any other UELs – if other UELs are found in the same level then all element will be interpreted as UELs and could result in GAMS domain violations.

Example #8 - Create a 2D variable from a DataFrame (uels_on_axes=True)

```
import gams.transfer as gt
import pandas as pd
import numpy as np
dim1 = [f"d{i}" for i in range(2)]
dim2 = [f"e{i}" for i in range(2)]
dim3 = [f"f{i}" for i in range(2)]
dim4 = [f"g{i}" for i in range(2)]
rng = np.random.default_rng(seed=100)
df = pd.DataFrame(
    data=rng.uniform(size=(4, 4)),
    index=pd.MultiIndex.from_product([dim1, dim2]),
    columns=pd.MultiIndex.from_product([dim3, dim4]),
)
In [1]: df
Out[1]:
             f0                  f1
             g0        g1        g0        g1
d0 e0  0.834982  0.596554  0.288863  0.042952
   e1  0.973654  0.596472  0.790263  0.910339
d1 e0  0.688154  0.189991  0.981479  0.284740
   e1  0.629273  0.581036  0.599912  0.535248
m = gt.Container()
v = gt.Variable(m, "v", domain=["*"] * 4, records=df, uels_on_axes=True)
In [8]: v.records
Out[8]:
   uni_0 uni_1 uni_2 uni_3     level  marginal  lower  upper  scale
0     d0    e0    f0    g0  0.834982       0.0   -inf    inf    1.0
1     d0    e0    f0    g1  0.596554       0.0   -inf    inf    1.0
2     d0    e0    f1    g0  0.288863       0.0   -inf    inf    1.0
3     d0    e0    f1    g1  0.042952       0.0   -inf    inf    1.0
4     d0    e1    f0    g0  0.973654       0.0   -inf    inf    1.0
5     d0    e1    f0    g1  0.596472       0.0   -inf    inf    1.0
6     d0    e1    f1    g0  0.790263       0.0   -inf    inf    1.0
7     d0    e1    f1    g1  0.910339       0.0   -inf    inf    1.0
8     d1    e0    f0    g0  0.688154       0.0   -inf    inf    1.0
9     d1    e0    f0    g1  0.189991       0.0   -inf    inf    1.0
10    d1    e0    f1    g0  0.981479       0.0   -inf    inf    1.0
11    d1    e0    f1    g1  0.284740       0.0   -inf    inf    1.0
12    d1    e1    f0    g0  0.629273       0.0   -inf    inf    1.0
13    d1    e1    f0    g1  0.581036       0.0   -inf    inf    1.0
14    d1    e1    f1    g0  0.599912       0.0   -inf    inf    1.0
15    d1    e1    f1    g1  0.535248       0.0   -inf    inf    1.0
```

Directly Set Records

As with sets, the primary advantage of the `setRecords` method is that `transfer` will convert many different (and convenient) data types into the standard data format (a Pandas DataFrame). Users that require higher performance will want to directly pass the `Container` a reference to a valid Pandas DataFrame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process and it avoids making a copy of the data in memory. In this section we walk the user through an example of how to set records directly.

Example #1 - Correctly set records (directly)

```
import gams.transfer as gt
import pandas as pd
import numpy as np
df = pd.DataFrame(
    data=[
        ("h" + str(h), "m" + str(m), "s" + str(s))
        for h in range(8760)
        for m in range(60)
        for s in range(60)
    ],
    columns=["h", "m", "s"],
)
# it is necessary to specify all variable attributes if setting records directly
# NOTE: all numeric data must be type float
df["level"] = np.random.uniform(0, 100, len(df))
df["marginal"] = 0.0
df["lower"] = gt.SpecialValues.NEGINF
df["upper"] = gt.SpecialValues.POSINF
df["scale"] = 1.0
m = gt.Container()
hrs = gt.Set(m, "h", records=df["h"].unique())
mins = gt.Set(m, "m", records=df["m"].unique())
secs = gt.Set(m, "s", records=df["s"].unique())
df["h"] = df["h"].astype(hrs.records["uni"].dtype)
df["m"] = df["m"].astype(mins.records["uni"].dtype)
df["s"] = df["s"].astype(secs.records["uni"].dtype)
a = gt.Variable(m, "a", domain=[hrs, mins, secs])
# set records
a.records = df
In [1]: a.isValid()
Out[1]: True
```

Attention

All numeric data in the records will need to be type `float` in order to maintain a valid symbol.

In this example we create a large variable (31,536,000 records and 8880 unique domain elements – we mimic data that is labeled for every second in one year) and assign it to a variable with `a.records`. `transfer` requires that all domain columns must be a categorical data type, furthermore this categorical must be ordered. The `records` setter function does very little work other than checking if the object being set is a DataFrame. This places more responsibility on the user to create a DataFrame that complies with the standard format. In Example #1 we take care to properly reference the categorical data types from the domain sets – and in the end `a.isValid() = True`. As with Set and Parameters, users can use the `.isValid(verbose=True)` method to debug any structural issues.

Generate Variable Records

Generating the initial `pandas.DataFrame` object could be difficult for `Variable` symbols that have a large number of records and a small number of UELs – these higher dimensional symbols will benefit from the `generateRecords` convenience function. Internally, `generateRecords` computes the dense Cartesian product of all the domain sets that define a symbol (`generateRecords` will only work on symbols where `<symbol>.domain_type == "regular"`).

Example #1 - Create a large (dense) 4D variable

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Variable(m, "a", "free", [i, j, k, l])
# generate the records
a.generateRecords()
In [1]: a.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
         i    j    k    l       level  marginal  lower  upper  scale
0         i0   j0   k0   l0  0.470248       0.0   -inf    inf    1.0
1         i0   j0   k0   l1  0.924286       0.0   -inf    inf    1.0
2         i0   j0   k0   l2  0.347550       0.0   -inf    inf    1.0
3         i0   j0   k0   l3  0.937009       0.0   -inf    inf    1.0
4         i0   j0   k0   l4  0.050716       0.0   -inf    inf    1.0
...      ...  ...  ...  ...       ...       ...    ...    ...    ...
6249995  i49  j49  k49  l45  0.385032       0.0   -inf    inf    1.0
6249996  i49  j49  k49  l46  0.029305       0.0   -inf    inf    1.0
6249997  i49  j49  k49  l47  0.440716       0.0   -inf    inf    1.0
6249998  i49  j49  k49  l48  0.432931       0.0   -inf    inf    1.0
6249999  i49  j49  k49  l49  0.157107       0.0   -inf    inf    1.0
[6250000 rows x 9 columns]
```

Note

> In Example #1 a large 4D variable was generated – by default, only the `level` value of these records are randomly drawn from the interval `[0,1]` (uniform distribution). Other variable attributes take the default record value.

As with `Parameters`, it is possible to generate a sparse variable with the `density` argument to `generateRecords`. We extend this example by passing our own custom `func` argument that will control the behavior of the `value` columns. The `func` argument accepts a `dict` of `callables` (i.e., a reference to a function).

Example #2 - Create a large (sparse) 4D variable with normally distributed values

```
import gams.transfer as gt
import numpy as np
# create a custom function to pass to ‘generateRecords‘
def level_dist(size):
    return np.random.normal(loc=10.0, scale=2.3, size=size)
def marginal_dist(size):
    return np.random.normal(loc=0.5, scale=0.1, size=size)
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Variable(m, "a", "free", [i, j, k, l])
# generate the records
a.generateRecords(density=0.05, func={"level":level_dist, "marginal":marginal_dist})
In [1]: a.isValid()
Out[1]: True
In [12]: a.records
Out[12]:
        i    j    k    l        level  marginal  lower  upper  scale
0        i0   j0   k0  l36  11.105235  0.468989   -inf    inf    1.0
1        i0   j0   k0  l40   5.697361  0.478019   -inf    inf    1.0
2        i0   j0   k1  l17  11.900784  0.473814   -inf    inf    1.0
3        i0   j0   k1  l24  10.105931  0.456925   -inf    inf    1.0
4        i0   j0   k1  l31   8.444142  0.490966   -inf    inf    1.0
...     ...  ...  ...  ...        ...       ...    ...    ...    ...
312495  i49  j49  k47  l17  11.523186  0.508001   -inf    inf    1.0
312496  i49  j49  k47  l20   9.341183  0.739237   -inf    inf    1.0
312497  i49  j49  k47  l26  10.705808  0.581103   -inf    inf    1.0
312498  i49  j49  k47  l32   7.910963  0.479655   -inf    inf    1.0
```

```
312499  i49  j49  k49   l8  11.800414  0.628040   -inf    inf    1.0
[312500 rows x 9 columns]
In [3]: a.records["level"].mean()
Out[3]: 10.004072307451391
In [4]: a.records["level"].std()
Out[4]: 2.292569938350144
In [5]: a.records["marginal"].mean()
Out[5]: 0.49970172269778
In [6]: a.records["marginal"].std()
Out[6]: 0.09998772109802055
```

Note

The custom `callable` function reference must expose a `size` argument. It might be tedious to know the exact number of the records that will be generated, especially if a fractional density is specified; therefore, the `generateRecords` method will pass in the correct size automatically. Users are encouraged to use the Numpy suite of random distributions when generating samples – custom functions have the potential to be computationally burdensome if a symbol has a large number of records.

Equation

There are two different ways to create a GAMS equation and add it to a `Container`.

1. Use `Equation` constructor
2. Use the `Container` method `addEquation` (which internally calls the `Equation` constructor)

Constructor

Constructor Arguments

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the `Container` object that the symbol is being added to	Yes	-
`description`	`str`	Description of symbol	No	""
`domain`	`list`, `str`, or `Set/Alias`	List of domains given either as string (∗ for universe set) or as reference to a Set/Alias object, an empty domain list will create a scalar equation	No	[]
`domain_forwarding`	`bool` or `list`	Flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of `bool` to control which domains to forward.	No	False
`name`	`str`	Name of symbol	Yes	-
`records`	many	Symbol records	No	None

Argument	Type	Description	Required	Default
type	str	Type of equation being created [eq (or E/e), geq (or G/g), leq (or L/l), nonbinding (or N/n), external (or X/x)]	Yes	-
uels_on_axes	bool	Instructs setRecords to assume symbol domain information is contained in the axes of the Pandas object	No	False

Note

Equation records can be updated through the object constructor (a new object will not be created) if a symbol of the same name already exists in the container, has the same domain, has the same type, and has the same domain_forwarding state. The symbol description will only be updated if new text is provided.

Properties

Property	Description	Type	Special Setter Behavior
container	reference to the Container that the symbol belongs to	Container	-
description	description of symbol	str	-
dimension	dimension of symbol	int	setting is a shorthand notation to create ["*"] * n domains in symbol
domain	list of domains given either as string (* for universe set) or as reference to the Set/Alias object	list, str, or Set/Alias	-
domain_forwarding	flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of bool to control which domains to forward.	bool or list	no effect after records have been set
domain_labels	column headings for the records DataFrame	list of str	will add a _<dimension> tag to user supplied column names (if not unique)
domain_names	string version of domain names	list of str	-
domain_type	none, relaxed or regular depending on state of domain links	str	-

Property	Description	Type	Special Setter Behavior
is_scalar	True if the len(self.domain) = 0	bool	-
modified	Flag that identifies if the Equation has been modified	bool	-
name	name of symbol	str	sets the GAMS name of the symbol
number_records	number of symbol records (i.e., returns len(self.records) if not None)	int	-
records	the main symbol records	pandas.DataFrame	responsive to domain_forwarding state
shape	a tuple describing the array dimensions if records were converted with .toDense()	tuple	-
summary	output a dict of only the metadata	dict	-
type	str type of variable	str	-

Methods

Method	Description	Arguments/Defaults	Returns
addUELs	adds UELs to the symbol dimensions. If dimensions is None then add UELs to all dimensions. ** All trailing whitespace is trimmed **	uels (str, list) dimensions=None (int, list, None)	None
capitalizeUELs	will capitalize all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
casefoldUELs	will casefold all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
countDomainViolations	returns the count of how many records contain at least one domain violation	-	int

Method	Description	Arguments/Defaults	Returns
countDuplicateRecords	returns the count of how many (case insensitive) duplicate records exist	-	int
countEps	total number of SpecialValues.EPS across all columns	columns="level" (str, list)	int or None
countNA	total number of SpecialValues.NA across all columns	columns="level" (str, list)	int or None
countNegInf	total number of SpecialValues.NEGINF across all columns	columns="level" (str, list)	int or None
countPosInf	total number of SpecialValues.POSINF across all columns	columns="level" (str, list)	int or None
countUndef	total number of SpecialValues.UNDEF across all columns	columns="level" (str, list)	int or None
dropDefaults	drop records that are set to GAMS default records (check .default_records property for values)	-	None
dropDomainViolations	drop records from the symbol that contain a domain violation	-	None
dropDuplicateRecords	drop records with (case insensitive) duplicate domains from the symbol – keep argument can take values of "first" (keeps the first instance of a duplicate record), "last" (keeps the last instance of a record), or False (drops all duplicates including the first and last)	keep="first" (str, False)	None
dropEps	drop records from the symbol that are GAMS EPS (zero 0.0 records will be retained)	-	None
dropMissing	drop records from the symbol that are NaN (includes both NA and Undef special values)	-	None
dropNA	drop records from the symbol that are GAMS NA	-	None
dropUndef	drop records from the symbol that are GAMS Undef	-	None

Method	Description	Arguments/Defaults	Returns
`equals`	Used to compare the symbol to another symbol. The `columns` argument allows the user to numerically compare only specified equation attributes (default is to compare all). If `check_uels=True` then check both used and unused UELs and confirm same order, otherwise only check used UELs in data and do not check UEL order. If `check_meta_data=True` then check that symbol name, description and equation type are the same, otherwise skip. `rtol` (relative tolerance) and `atol` (absolute tolerance) set equality tolerances; can be different tolerances for different equation attributes (if specified as a `dict`). If `verbose=True` will return an exception from the asserter describing the nature of the difference.	`columns=["level", "marginal", "lower", "upper", "scale"]` `check_uels=True` (`bool`) `check_meta_data=True` (`bool`) `rtol=0.0` (`int`, `float`, `None`) `atol=0.0` (`int`, `float`, `None`) `verbose=False` (`bool`)	`bool`
`findDomainViolations`	get a view of the records DataFrame that contain any domain violations	-	`pandas.DataFrame`
`findDuplicateRecords`	get a view of the records DataFrame that contain any (case insensitive) duplicate domains – `keep` argument can take values of "first" (finds all duplicates while keeping the first instance as unique), "last" (finds all duplicates while keeping the last instance as unique), or `False` (finds all duplicates)	`keep="first"` (str, `False`)	`pandas.DataFrame`
`findEps`	find positions of `SpecialValues.EPS` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`

Method	Description	Arguments/Defaults	Returns
`findNA`	find positions of `SpecialValues.NA` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`
`findNegInf`	find positions of `SpecialValues.NEGINF` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`
`findPosInf`	find positions of `SpecialValues.POSINF` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`
`findUndef`	find positions of `SpecialValues.Undef` in `column`	`column="level"` (`str`)	`pandas.DataFrame` or `None`
`generateRecords`	convenience method to set standard `pandas.DataFrame` formatted records given domain set information. Will generate records with the Cartesian product of all domain sets. The `density` argument can take any value on the interval `[0,1]`. If `density` is <1 then randomly selected records will be removed. `density` will accept a `list` of length `dimension` -- allows users to specify a density per symbol dimension. Random number state can be set with `seed` argument.	`density=1.0` (`float`, `list`) `func=numpy.random.uniform(0,1)` (`dict` of `callables`) `seed=None` (`int`, `None`)	`None`
`getMaxAbsValue`	get the maximum absolute value across all `columns`	`columns="level"` (`str`, `list`)	`float` or `None`
`getMaxValue`	get the maximum value across all `columns`	`columns="level"` (`str`, `list`)	`float` or `None`
`getMeanValue`	get the mean value across all `columns`	`columns="level"` (`str`, `list`)	`float` or `None`
`getMinValue`	get the minimum value across all `columns`	`columns="level"` (`str`, `list`)	`float` or `None`
`getSparsity`	get the sparsity of the symbol w.r.t the cardinality	-	`float`

Method	Description	Arguments/Defaults	Returns
`getUELs`	gets UELs from symbol `dimensions`. If `dimensions` is `None` then get UELs from all dimensions (maintains order). The argument `codes` accepts a list of `str` UELs and will return the corresponding `int`; must specify a single dimension if passing `codes`. Returns only UELs in the data if `ignore_unused=True`, otherwise return all UELs.	`dimensions=None` (`int`, `list`, `None`) `codes=None` (`int`, `list`, `None`) `ignore_unused=False` (`bool`)	`list`
`hasDomainViolations`	returns `True` if there are domain violations in the records, returns `False` if not.	-	`bool`
`hasDuplicateRecords`	returns `True` if there are (case insensitive) duplicate records in the symbol, returns `False` if not.	-	`bool`
`isValid`	checks if the symbol is in a valid format, throw exceptions if `verbose=True`, recheck a symbol if `force=True`	`verbose=False` `force=True`	`bool`
`ljustUELs`	will left justify all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`length` (`int`) `fill_character=None` - if `None`, assumes `" "` `dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`lowerUELs`	will lowercase all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`lstripUELs`	will left strip whitespace from all UELs in the symbol or a subset of specified `dimensions`, can be chained with other *`UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`

Method	Description	Arguments/Defaults	Returns
pivot	Convenience function to pivot records into a new shape (only symbols with >1D can be pivoted). If index is None then it is set to dimensions [0..dimension-1]. If columns is None then it is set to the last dimension. If value is None then the level values will be pivoted. Missing values in the pivot will take the value provided by fill_value	index=None (str, list, None) columns=None (str, list, None) value (str) fill_value=None (int, float, str)	pd.DataFrame
removeUELs	removes UELs that appear in the symbol dimensions, If uels is None then remove all unused UELs (categories). If dimensions is None then operate on all dimensions.	uels=None (str, list, None) dimensions=None (int, list, None)	bool
renameUELs	renames UELs (case-sensitive) that appear in the symbol dimensions. If dimensions is None then operate on all dimensions of the symbol. If allow_merge=True, the categorical object will be re-created to offer additional data flexibility. ** All trailing whitespace is trimmed **	uels (str, list, dict) dimensions (int, list, None) allow_merge=False (bool)	None
reorderUELs	reorders the UELs in the symbol dimensions. If uels is None, reorder UELs to data order and append any unused categories. If dimensions is None then reorder UELs in all dimensions of the symbol.	uels (str, list, dict, None) dimensions (int, list, None)	None

Method	Description	Arguments/Defaults	Returns
rjustUELs	will right justify all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	length (int) fill_character=None - if None, assumes " " dimensions=None (int, list, or None) - if None, assumes all symbols	self
rstripUELs	will right strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
setRecords	main convenience method to set standard pandas.DataFrame records. If uels_on_axes=True setRecords will assume that all domain information is contained in the axes of the pandas object – data will be flattened (if necessary).	records (many types)	None
setUELs	set the UELs for symbol dimensions. If dimensions is None then set UELs for all dimensions. If rename=True, then the old UEL names will be renamed with the new UEL names. ** All trailing whitespace is trimmed **	uels (str, list) dimensions=None (int, list, None) rename=False (bool)	None
stripUELs	will strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
titleUELs	will title (capitalize all individual words) in all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
toDense	convert column to a dense numpy.array format	column="level" (str)	numpy.array or None

Method	Description	Arguments/Defaults	Returns
`toDict`	convenience method to return symbol records as a Python dictionary. `columns` will control which attributes to include in the `dict`. `orient` can take values `natural` or `columns` and will control the shape of the `dict`. Must use `orient="columns"` if attempting to set symbol records with `setRecords`.	`columns="level"` (str) `orient="natural"` (str)	`dict`
`toList`	convenience method to return symbol records as a Python list, the `columns` argument will control with attributes to include in the list	`columns="level"` (str)	`list`
`toSparseCoo`	convert `column` to a sparse COOrdinate `numpy.array` format	`column="level"` (`str`)	sparse matrix format or `None`
`toValue`	convenience method to return symbol records as a Python float. Only possible with scalar symbols. Attribute can be specified with `column` argument.	`column="level"` (str)	`float`
`upperUELs`	will uppercase all UELs in the symbol or a subset of specified `dimensions`, can be chained with other `*UELs` string operations	`dimensions=None` (`int`, `list`, or `None`) - if `None`, assumes all symbols	`self`
`whereMax`	find the domain entry of records with a maximum value (return first instance only)	`column="level"` (`str`)	`list` of `str` or `None`
`whereMaxAbs`	find the domain entry of records with a maximum absolute value (return first instance only)	`column="level"` (`str`)	`list` of `str` or `None`
`whereMin`	find the domain entry of records with a minimum value (return first instance only)	`column="level"` (`str`)	`list` of `str` or `None`

Adding Equation Records

Adding equation records mimics that of variables – three possibilities exist to assign symbol records to an equation (roughly ordered in complexity):

1. Setting the argument `records` in the set constructor/container method (internally calls `setRecords`) - creates a data copy
2. Using the symbol method `setRecords` - creates a data copy
3. Setting the property `records` directly - does not create a data copy

Setting equation records require the user to be explicit with the type of equation that is being created; in contrast to setting variable records (where the default variable is considered to be `free`).

If the data is in a convenient format, a user may want to pass the records directly within the equation constructor. This is an optional keyword argument and internally the equation constructor will simply call the `setRecords` method. In contrast to the `setRecords` methods in in either the Set or Parameter classes the `setRecords` method for variables will only accept Pandas DataFrames and specially structured `dict` for creating records from matrices. This restriction is out of necessity because to properly set a record for an Equation the user must pass data for the `level`, `marginal`, `lower`, `upper` and `scale` attributes. That said, any missing attributes will be filled in with the GAMS default record values (`level = 0.0`, `marginal = 0.0`, `lower = -inf`, `upper = inf`, `scale = 1.0`). We show a few examples of ways to create differently structured variables:

Example #1 - Create a GAMS scalar equation

```
import gams.transfer as gt
m = gt.Container()
# here we create an equality (=E=) equation
z = gt.Equation(m, "z", "eq", records=pd.DataFrame(data=[3.14159], columns=["level"]))
# NOTE: the above syntax is equivalent to -
# pi = gt.Equation(m, "pi", "eq")
# pi.setRecords(pd.DataFrame(data=[3.14159], columns=["level"]))
# NOTE: the above syntax is also equivalent to -
# m.addEquation("pi", "eq", records=pd.DataFrame(data=[3.14159], columns=["level"]))
In [1]: pi.records
Out[1]:
     level  marginal  lower  upper  scale
0  3.14159       0.0   -inf    inf    1.0
```

Example #2 - Create a 1D Equation (defined over $*$) from a list of tuples

In this example we only set the `marginal` values.

```
import gams.transfer as gt
m = gt.Container()
# here we define a greater than or equal (=G=) equation
i = gt.Equation(
    m,
    "i",
    "geq",
    domain=["*"],
    records=pd.DataFrame(
        data=[("i" + str(i), i) for i in range(5)], columns=["domain", "marginal"]
    ),
)
In [1]: i.type
Out[1]: 'geq'
In [2]: i.records
Out[2]:
  uni    level  marginal  lower  upper  scale
0    i0    0.0       0.0   -inf    inf    1.0
1    i1    0.0       1.0   -inf    inf    1.0
2    i2    0.0       2.0   -inf    inf    1.0
3    i3    0.0       3.0   -inf    inf    1.0
4    i4    0.0       4.0   -inf    inf    1.0
```

Example #3 - Create a 1D Equation (defined over a set) from a list of tuples

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=["i" + str(i) for i in range(5)])
# here we define a less than or equal (=L=) equation
e = gt.Equation(
    m,
    "e",
    "leq",
    domain=i,
    records=pd.DataFrame(
        data=[("i" + str(i), i) for i in range(5)], columns=["domain", "marginal"]
    ),
)
In [1]: i.type
Out[1]: 'leq'
In [5]: e.records
Out[5]:
  i    level  marginal  lower  upper  scale
0  i0    0.0       0.0   -inf    inf    1.0
1  i1    0.0       1.0   -inf    inf    1.0
2  i2    0.0       2.0   -inf    inf    1.0
3  i3    0.0       3.0   -inf    inf    1.0
4  i4    0.0       4.0   -inf    inf    1.0
```

Example #4 - Create a 2D equation, specifying no numerical data

```
import gams.transfer as gt
import pandas as pd
m = gt.Container()
e = gt.Equation(
    m,
    "e",
    "eq",
    ["*", "*"],
    records=pd.DataFrame([("seattle", "san-diego"), ("chicago", "madison")]),
)
In [1]: e.records
Out[1]:
     uni_0      uni_1  level  marginal  lower  upper  scale
0  seattle  san-diego    0.0       0.0   -inf    inf    1.0
1  chicago    madison    0.0       0.0   -inf    inf    1.0
```

Example #5 - Create a 2D equation (defined over a set) from a matrix

```
import gams.transfer as gt
import pandas as pd
import numpy as np
m = gt.Container()
i = gt.Set(m, "i", ["*"], records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", ["*"], records=["j" + str(i) for i in range(5)])
a = gt.Parameter(
    m,
    "a",
    [i, j],
    records=[("i" + str(i), "j" + str(j), i + j) for i in range(5) for j in range(5)],
)
# create a nonbinding (=N=) equation and set the level and marginal attributes from matricies
e = gt.Equation(
    m, "e", "nonbinding", domain=[i, j], records={"level": a.toDense(), "marginal": a.toDense()}
)
In [1]: e.records
Out[1]:
   i   j    level  marginal  lower  upper  scale
0   i0  j1    1.0       1.0   -inf    inf    1.0
1   i0  j2    2.0       2.0   -inf    inf    1.0
2   i0  j3    3.0       3.0   -inf    inf    1.0
3   i0  j4    4.0       4.0   -inf    inf    1.0
4   i1  j0    1.0       1.0   -inf    inf    1.0
5   i1  j1    2.0       2.0   -inf    inf    1.0
6   i1  j2    3.0       3.0   -inf    inf    1.0
7   i1  j3    4.0       4.0   -inf    inf    1.0
8   i1  j4    5.0       5.0   -inf    inf    1.0
9   i2  j0    2.0       2.0   -inf    inf    1.0
10  i2  j1    3.0       3.0   -inf    inf    1.0
```

```
11  i2  j2    4.0       4.0   -inf    inf    1.0
12  i2  j3    5.0       5.0   -inf    inf    1.0
13  i2  j4    6.0       6.0   -inf    inf    1.0
14  i3  j0    3.0       3.0   -inf    inf    1.0
15  i3  j1    4.0       4.0   -inf    inf    1.0
16  i3  j2    5.0       5.0   -inf    inf    1.0
17  i3  j3    6.0       6.0   -inf    inf    1.0
18  i3  j4    7.0       7.0   -inf    inf    1.0
19  i4  j0    4.0       4.0   -inf    inf    1.0
20  i4  j1    5.0       5.0   -inf    inf    1.0
21  i4  j2    6.0       6.0   -inf    inf    1.0
22  i4  j3    7.0       7.0   -inf    inf    1.0
23  i4  j4    8.0       8.0   -inf    inf    1.0
# if not specified, the toDense() method will convert the level values to a matrix
In [2]: e.toDense()
Out[2]:
array([[0., 1., 2., 3., 4.],
       [1., 2., 3., 4., 5.],
       [2., 3., 4., 5., 6.],
       [3., 4., 5., 6., 7.],
       [4., 5., 6., 7., 8.]])
```

Example #6 - Create a 1D equation from a pandas Series

```
import gams.transfer as gt
import pandas as pd
s = pd.Series(index=["a", "b", "c"], data=[i + 1 for i in range(3)])
m = gt.Container()
e = gt.Equation(m, "e", "eq", domain=["*"], records=s, uels_on_axes=True)
In [1]: e.records
Out[1]:
  uni  level  marginal  lower  upper  scale
0   a    1.0       0.0   -inf    inf    1.0
1   b    2.0       0.0   -inf    inf    1.0
2   c    3.0       0.0   -inf    inf    1.0
```

Note

> `transfer` will assume that the `level` value is being set if attributes cannot be found in the axes (same behavior when setting `Variables`)

Example #7 - Create a 1D equation from a pandas Series (set only the marginal)

```
import gams.transfer as gt
import pandas as pd
# NOTE: we include the "marginal" label in level 1 of the MultiIndex
s = pd.Series(
    index=pd.MultiIndex.from_product([["a", "b", "c"], ["marginal"]]),
    data=[i + 1 for i in range(3)],
)
m = gt.Container()
e = gt.Equation(m, "e", "eq", domain=["*"], records=s, uels_on_axes=True)
In [1]: v.records
Out[1]:
  uni  level  marginal  lower  upper  scale
0   a    0.0       1.0   -inf    inf    1.0
1   b    0.0       2.0   -inf    inf    1.0
2   c    0.0       3.0   -inf    inf    1.0
```

Example #8 - Create a 2D equation from a DataFrame (uels_on_axes=True)

```
import gams.transfer as gt
import pandas as pd
import numpy as np
dim1 = [f"d{i}" for i in range(2)]
dim2 = [f"e{i}" for i in range(2)]
dim3 = [f"f{i}" for i in range(2)]
dim4 = [f"g{i}" for i in range(2)]
rng = np.random.default_rng(seed=100)
df = pd.DataFrame(
```

```
    data=rng.uniform(size=(4, 4)),
    index=pd.MultiIndex.from_product([dim1, dim2]),
    columns=pd.MultiIndex.from_product([dim3, dim4]),
)
In [1]: df
Out[1]:
             f0                  f1
             g0        g1        g0        g1
d0 e0  0.834982  0.596554  0.288863  0.042952
   e1  0.973654  0.596472  0.790263  0.910339
d1 e0  0.688154  0.189991  0.981479  0.284740
   e1  0.629273  0.581036  0.599912  0.535248
m = gt.Container()
e = gt.Equation(m, "e", "eq", domain=["*"] * 4, records=df, uels_on_axes=True)
In [8]: e.records
Out[8]:
   uni_0 uni_1 uni_2 uni_3     level  marginal  lower  upper  scale
0     d0    e0    f0    g0  0.834982       0.0   -inf    inf    1.0
1     d0    e0    f0    g1  0.596554       0.0   -inf    inf    1.0
2     d0    e0    f1    g0  0.288863       0.0   -inf    inf    1.0
3     d0    e0    f1    g1  0.042952       0.0   -inf    inf    1.0
4     d0    e1    f0    g0  0.973654       0.0   -inf    inf    1.0
5     d0    e1    f0    g1  0.596472       0.0   -inf    inf    1.0
6     d0    e1    f1    g0  0.790263       0.0   -inf    inf    1.0
7     d0    e1    f1    g1  0.910339       0.0   -inf    inf    1.0
8     d1    e0    f0    g0  0.688154       0.0   -inf    inf    1.0
9     d1    e0    f0    g1  0.189991       0.0   -inf    inf    1.0
10    d1    e0    f1    g0  0.981479       0.0   -inf    inf    1.0
11    d1    e0    f1    g1  0.284740       0.0   -inf    inf    1.0
12    d1    e1    f0    g0  0.629273       0.0   -inf    inf    1.0
13    d1    e1    f0    g1  0.581036       0.0   -inf    inf    1.0
14    d1    e1    f1    g0  0.599912       0.0   -inf    inf    1.0
15    d1    e1    f1    g1  0.535248       0.0   -inf    inf    1.0
```

Directly Set Records

As with set, parameters and variables, the primary advantage of the `setRecords` method is that `transfer` will convert many different (and convenient) data types into the standard data format (a Pandas DataFrame). Users that require higher performance will want to directly pass the `Container` a reference to a valid Pandas DataFrame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process and it avoids making a copy of the data in memory. In this section we walk the user through an example of how to set records directly.

Example #1 - Correctly set records (directly)

```
import gams.transfer as gt
import pandas as pd
import numpy as np
df = pd.DataFrame(
    data=[
        ("h" + str(h), "m" + str(m), "s" + str(s))
        for h in range(8760)
        for m in range(60)
        for s in range(60)
    ],
    columns=["h", "m", "s"],
)
# it is necessary to specify all variable attributes if setting records directly
# NOTE: all numeric data must be type float
df["level"] = np.random.uniform(0, 100, len(df))
df["marginal"] = 0.0
df["lower"] = gt.SpecialValues.NEGINF
df["upper"] = gt.SpecialValues.POSINF
df["scale"] = 1.0
m = gt.Container()
hrs = gt.Set(m, "h", records=df["h"].unique())
mins = gt.Set(m, "m", records=df["m"].unique())
secs = gt.Set(m, "s", records=df["s"].unique())
df["h"] = df["h"].astype(hrs.records["uni"].dtype)
df["m"] = df["m"].astype(mins.records["uni"].dtype)
df["s"] = df["s"].astype(secs.records["uni"].dtype)
a = gt.Equation(m, "a", "eq", domain=[hrs, mins, secs])
# set records
a.records = df
In [1]: e.isValid()
Out[1]: True
```

Attention

All numeric data in the records will need to be type `float` in order to maintain a valid symbol.

In this example we create a large equation (31,536,000 records and 8880 unique domain elements) and assign it to a variable with `a.records`. `transfer` requires that all domain columns must be a categorical data type, furthermore this categorical must be ordered. The `records` setter function does very little work other than checking if the object being set is a DataFrame. This places more responsibility on the user to create a DataFrame that complies with the standard format. In Example #1 we take care to properly reference the categorical data types from the domain sets – and in the end `a.isValid() = True`. As with Set and Parameters, users can use the `.isValid(verbose=True)` method to debug any structural issues.

Generate Equation Records

Generating the initial `pandas.DataFrame` object could be difficult for `Equation` symbols that have a large number of records and a small number of UELs – these higher dimensional symbols will benefit from the `generateRecords` convenience function. Internally, `generateRecords` computes the dense Cartesian product of all the domain sets that define a symbol (`generateRecords` will only work on symbols where `<symbol>.domain_type == "regular"`).

Example #1 - Create a large (dense) 4D equation

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Equation(m, "a", "eq", [i, j, k, l])
# generate the records
a.generateRecords()
In [1]: a.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
         i    j    k    l       level  marginal  lower  upper  scale
0         i0   j0   k0   l0  0.470248       0.0   -inf    inf    1.0
1         i0   j0   k0   l1  0.924286       0.0   -inf    inf    1.0
2         i0   j0   k0   l2  0.347550       0.0   -inf    inf    1.0
3         i0   j0   k0   l3  0.937009       0.0   -inf    inf    1.0
4         i0   j0   k0   l4  0.050716       0.0   -inf    inf    1.0
...      ...  ...  ...  ...       ...       ...    ...    ...    ...
6249995  i49  j49  k49  l45  0.385032       0.0   -inf    inf    1.0
6249996  i49  j49  k49  l46  0.029305       0.0   -inf    inf    1.0
6249997  i49  j49  k49  l47  0.440716       0.0   -inf    inf    1.0
6249998  i49  j49  k49  l48  0.432931       0.0   -inf    inf    1.0
6249999  i49  j49  k49  l49  0.157107       0.0   -inf    inf    1.0
[6250000 rows x 9 columns]
```

Note

In Example #1 a large 4D equation was generated – by default, only the `level` value of these records are randomly drawn from the interval `[0,1]` (uniform distribution). Other variable attributes take the default record value.

As with `Variables`, it is possible to generate a sparse variable with the `density` argument to `generateRecords`. We extend this example by passing our own custom `func` argument that will control the behavior of the `value` columns. The `func` argument accepts a `dict` of `callables` (i.e., a reference to a function).

Example #2 - Create a large (sparse) 4D equation with normally distributed values

```
import gams.transfer as gt
import numpy as np
# create a custom function to pass to ‘generateRecords‘
def level_dist(size):
    return np.random.normal(loc=10.0, scale=2.3, size=size)
def marginal_dist(size):
    return np.random.normal(loc=0.5, scale=0.1, size=size)
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(50)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(50)])
k = gt.Set(m, "k", records=[f"k{i}" for i in range(50)])
l = gt.Set(m, "l", records=[f"l{i}" for i in range(50)])
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = gt.Equation(m, "a", "eq", [i, j, k, l])
# generate the records
a.generateRecords(density=0.05, func={"level":level_dist, "marginal":marginal_dist})
In [1]: a.isValid()
Out[1]: True
In [12]: a.records
Out[12]:
        i    j    k    l        level  marginal  lower  upper  scale
0        i0   j0   k0  136  11.105235  0.468989   -inf    inf    1.0
1        i0   j0   k0  140   5.697361  0.478019   -inf    inf    1.0
2        i0   j0   k1  117  11.900784  0.473814   -inf    inf    1.0
3        i0   j0   k1  124  10.105931  0.456925   -inf    inf    1.0
4        i0   j0   k1  131   8.444142  0.490966   -inf    inf    1.0
...     ...  ...  ...  ...        ...       ...    ...    ...    ...
312495  i49  j49  k47  117  11.523186  0.508001   -inf    inf    1.0
312496  i49  j49  k47  120   9.341183  0.739237   -inf    inf    1.0
312497  i49  j49  k47  126  10.705808  0.581103   -inf    inf    1.0
312498  i49  j49  k47  132   7.910963  0.479655   -inf    inf    1.0
312499  i49  j49  k49   18  11.800414  0.628040   -inf    inf    1.0
[312500 rows x 9 columns]
In [3]: a.records["level"].mean()
Out[3]: 10.004072307451391
In [4]: a.records["level"].std()
Out[4]: 2.292569938350144
In [5]: a.records["marginal"].mean()
Out[5]: 0.49970172269778
In [6]: a.records["marginal"].std()
Out[6]: 0.09998772109802055
```

Note

> The custom `callable` function reference must expose a `size` argument. It might be tedious to know the exact number of the records that will be generated, especially if a fractional density is specified; therefore, the `generateRecords` method will pass in the correct size automatically. Users are encouraged to use the Numpy suite of random distributions when generating samples – custom functions have the potential to be computationally burdensome if a symbol has a large number of records.

Alias

There are two different ways to create a GAMS alias and add it to a `Container`.

1. Use `Alias` constructor
2. Use the `Container` method `addAlias` (which internally calls the `Alias` constructor)

Constructor

Constructor Arguments

Argument	Type	Description	Required	Default
alias_with	Set object	set object from which to create an alias	Yes	-
container	Container	A reference to the Container object that the symbol is being added to	Yes	-
name	str	Name of symbol	Yes	-

Note

The Alias property alias_with can be updated through the object constructor (a new object will not be created) if a symbol of the same name already exists in the container.

Example - Creating an alias from a set

transfer only stores the reference to the parent set as part of the alias structure – most properties that are called from an alias object simply point to the properties of the parent set (with the exception of container, name, and alias_with). It is possible to create an alias from another alias object. In this case a recursive search will be performed to find the root parent set – this is the set that will ultimately be stored as the alias_with property. We can see this behavior in the following example:

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
ip = gt.Alias(m, "ip", i)
ipp = gt.Alias(m, "ipp", ip)
In [1]: ip.alias_with.name
Out[1]: 'i'
In [2]: ipp.alias_with.name
Out[2]: 'i'
```

Properties

Property	Description	Type	Special Setter Behavior
alias_with	aliased object	Set	-
container	reference to the Container that the symbol belongs to	Container	-
description	description of symbol	str	-
dimension	dimension of symbol	int	setting is a shorthand notation to create ["*"] * n domains in symbol
domain	list of domains given either as string (* for universe set) or as reference to the Set/Alias object	list, str, or Set/Alias	-
domain_forwarding	flag that forces set elements to be recursively included in all parent sets (i.e., implicit set growth). Can pass as a list of bool to control which domains to forward.	bool or list	no effect after records have been set

Property	Description	Type	Special Setter Behavior
domain_labels	column headings for the records DataFrame	list of str	will add a _<dimension> tag to user supplied column names (if not unique)
domain_names	string version of domain names	list of str	-
domain_type	none, relaxed or regular depending on state of domain links	str	-
is_singleton	if symbol is a singleton set	bool	-
modified	Flag that identifies if the Alias has been modified	bool	-
name	name of symbol	str	sets the GAMS name of the symbol
number_records	number of symbol records (i.e., returns len(self.records) if not None)	int	-
records	the main symbol records	pandas.DataFrame	responsive to domain_forwarding state
summary	output a dict of only the metadata	dict	-

Methods

Method	Description	Arguments/Defaults	Returns
addUELs	adds UELs to the parent set dimensions. If dimensions is None then add UELs to all dimensions. ** All trailing whitespace is trimmed **	uels (str, list) dimensions=None (int, list, None)	None

Method	Description	Arguments/Defaults	Returns
equals	Used to compare the symbol to another symbol. If check_uels=True then check both used and unused UELs and confirm same order, otherwise only check used UELs in data and do not check UEL order. If check_element_text=True then check that all set elements have the same descriptive element text, otherwise skip. If check_meta_data=True then check that symbol name and description are the same, otherwise skip. If verbose=True will return an exception from the asserter describing the nature of the difference.	check_uels=True (bool) check_element_text=True (bool) check_meta_data=True (bool) verbose=False (bool)	bool
capitalizeUELs	will capitalize all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
casefoldUELs	will casefold all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
countDomainViolations	returns the count of how many records in the parent set contain at least one domain violation	-	int
countDuplicateRecords	returns the count of how many (case insensitive) duplicate records exist in the parent set	-	int
dropDomainViolations	drop records from the parent set that contain a domain violation	-	None

Method	Description	Arguments/Defaults	Returns
`dropDuplicateRecords`	drop records with (case insensitive) duplicate domains from the parent set – `keep` argument can take values of "first" (keeps the first instance of a duplicate record), "last" (keeps the last instance of a record), or `False` (drops all duplicates including the first and last)	`keep="first"` (str, `False`)	`None`
`findDomainViolations`	get a view of the records DataFrame that contain any domain violations	-	`pandas.DataFrame`
`findDuplicateRecords`	get a view of the records DataFrame from the parent set that contain any (case insensitive) duplicate domains – `keep` argument can take values of "first" (finds all duplicates while keeping the first instance as unique), "last" (finds all duplicates while keeping the last instance as unique), or `False` (finds all duplicates)	`keep="first"` (str, `False`)	`pandas.DataFrame`
`getDomainViolations`	returns a `list` of `DomainViolation` objects if any (`None` otherwise)	-	`list` or `None`
`getSparsity`	get the sparsity of the symbol w.r.t the cardinality	-	`float`
`getUELs`	gets UELs from the parent set `dimensions`. If `dimensions` is `None` then get UELs from all dimensions (maintains order). The argument `codes` accepts a list of `str` UELs and will return the corresponding `int`; must specify a single dimension if passing `codes`. Returns only UELs in the data if `ignore_unused=True`, otherwise return all UELs.	`dimensions=None` (`int`, `list`, `None`) `codes=None` (`int`, `list`, `None`) `ignore_unused=False` (`bool`)	`list`
`hasDomainViolations`	returns `True` if there are domain violations in the records of the parent set, returns `False` if not.	-	`bool`

Method	Description	Arguments/Defaults	Returns
hasDuplicateRecords	returns True if there are (case insensitive) duplicate records in the parent set, returns False if not.	-	bool
isValid	checks if the symbol is in a valid format, throw exceptions if verbose=True, re-check a symbol if force=True	verbose=False force=True	bool
ljustUELs	will left justify all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	length (int) fill_character=None - if None, assumes " " dimensions=None (int, list, or None) - if None, assumes all symbols	self
lowerUELs	will lowercase all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
lstripUELs	will left strip whitespace from all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
pivot	Convenience function to pivot records into a new shape (only symbols with >1D can be pivoted). If index is None then it is set to dimensions [0..dimension-1]. If columns is None then it is set to the last dimension. Missing values in the pivot will take the value provided by fill_value	index=None (str, list, None) columns=None (str, list, None) fill_value=None (int, float, str)	pd.DataFrame
removeUELs	removes UELs that appear in the parent set dimensions, If uels is None then remove all unused UELs (categories). If dimensions is None then operate on all dimensions.	uels=None (str, list, None) dimensions=None (int, list, None)	bool

Method	Description	Arguments/Defaults	Returns
renameUELs	renames UELs (case-sensitive) that appear in the parent set dimensions. If dimensions is None then operate on all dimensions of the symbol. If allow_merge=True, the categorical object will be re-created to offer additional data flexibility. ** All trailing whitespace is trimmed **	uels (str, list, dict) dimensions (int, list, None) allow_merge=False (bool)	None
reorderUELs	reorders the UELs in the parent set dimensions. If uels is None, reorder UELs to data order and append any unused categories. If dimensions is None then reorder UELs in all dimensions of the parent set.	uels (str, list, dict, None) dimensions (int, list, None)	None
rjustUELs	will right justify all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	length (int) fill_character=None - if None, assumes " " dimensions=None (int, list, or None) - if None, assumes all symbols	self
rstripUELs	will right strip whitespace from all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
setRecords	main convenience method to set standard pandas.DataFrame formatted records. If uels_on_axes=True setRecords will assume that all domain information is contained in the axes of the pandas object – data will be flattened (if necessary).	records (many types)	None

Method	Description	Arguments/Defaults	Returns
setUELs	set the UELs for parent set dimensions. If dimensions is None then set UELs for all dimensions. If rename=True, then the old UEL names will be renamed with the new UEL names. ** All trailing whitespace is trimmed **	uels (str, list) dimensions=None (int, list, None) rename=False (bool)	None
stripUELs	will strip whitespace from all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
titleUELs	will title (capitalize all individual words) in all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self
toList	convenience method to return symbol records as a Python list	include_element_text=False (bool)	list
upperUELs	will uppercase all UELs in the parent set or a subset of specified dimensions in the parent set, can be chain with other *UELs string operations	dimensions=None (int, list, or None) - if None, assumes all symbols	self

Adding Alias Records

The linked structure of Aliases offers some unique opportunities to access some of the setter functionality of the parent set. Specifically, transfer allows the user to change the domain, description, dimension, and records of the underlying parent set as a shorthand notation. We can see this behavior if we look at a modified Example #1 from Adding Set Records.

Example - Creating set records through an alias link

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i")
ip = gt.Alias(m, "ip",i)
ip.description = "adding new descriptive set text"
ip.domain = ["*", "*"]
ip.setRecords([("i" + str(i), "j" + str(j)) for i in range(3) for j in range(3)])
In [1]: i.description
Out[1]: 'adding new descriptive set text'
In [2]: i.domain
Out[2]: ['*', '*']
In [3]: i.records
```

```
Out[3]:
  uni_0 uni_1 element_text
0    i0    j0
1    i0    j1
2    i0    j2
3    i1    j0
4    i1    j1
5    i1    j2
6    i2    j0
7    i2    j1
8    i2    j2
```

Note

An alias `.isValid()=True` when the underlying parent set is also valid – if the parent set is removed from the Container the alias will no longer be valid.

UniverseAlias

There are two different ways to create a GAMS UniverseAlias (an alias to the universe) and add it to a `Container`.

1. Use `UniverseAlias` constructor
2. Use the `Container` method `addUniverseAlias` (which internally calls the `UniverseAlias` constructor)

Constructor

Constructor Arguments

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the `Container` object that the symbol is being added to	Yes	-
`name`	`str`	Name of symbol	Yes	-

Example - Creating an alias to the universe

In GAMS it is possible to create aliases to the universe (i.e., the entire list of UELs) with the syntax:

```
set i / i1, i2 /;
alias(h,*);
set j / j1, j2 /;
```

In this small example, `h` would be associated with all four UELs (`i1`, `i2`, `j1` and `j2`) even though set `j` was defined after the alias declaration. `transfer` mimics this behavior with the `UniverseAlias` class. Internally, the `records` attribute will always call the <`Container`>`.getUELs()` and build the Pandas `DataFrame` on the fly. The `UniverseAlias` class is fundamentally different from the `Alias` class because it does not point to a parent set at all; it is not possible to perform operations (like `setRecords` or

`findDomainViolations`) on the parent set through a `UniverseAlias` (because there is no parent set). This means that a `UniverseAlias` can be created by only defining the symbol name. We can see this behavior in the following example:

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2"])
h = gt.UniverseAlias(m, "h")
j = gt.Set(m, "j", records=["j1", "j2"])
# -- alternative syntax --
# m = gt.Container()
# m.addSet("i", records=["i1", "i2"])
# m.addUniverseAlias("h")
# m.addSet("j", records=["j1", "j2"])
In [1]: m.data
Out[1]:
{'i': <Set `i` (0x7f9598a3bd90)>,
 'h': <UniverseAlias `h` (0x7f9598a61690)>,
 'j': <Set `j` (0x7f95b9359cc0)>}
In [2]: h.records
Out[2]:
   uni
0  i1
1  i2
2  j1
3  j2
```

Note

Unlike other sets, the universe does not hold on to set `element_text`, thus the returned `DataFrame` for the `UniverseAlias` will only have 1 column.

Properties

Property	Description	Type	Special Setter Behavior
`alias_with`	always $*$	`str`	-
`container`	reference to the `Container` that the symbol belongs to	`Container`	-
`description`	always `Aliased with` $*$	`str`	-
`dimension`	always `1`	`int`	-
`domain`	always `["*"]`	`list` of `str`	-
`domain_labels`	always `["*"]`	`list` of `str`	-
`domain_names`	always `["*"]`	`list` of `str`	-
`domain_type`	always `none`	`str`	-
`is_singleton`	always `False`	`bool`	-
`modified`	flag that identifies if the `UniverseAlias` has been modified	`bool`	-
`name`	name of symbol	`str`	sets the GAMS name of the symbol
`number_records`	number of symbol records (i.e., returns `len(records)` if not `None`)	`int`	-
`records`	the main symbol records	`pandas.DataFrame`	-
`summary`	output a `dict` of only the metadata	`dict`	-

Methods

Method	Description	Arguments/Defaults	Returns
equals	Used to compare the symbol to another symbol. If check_uels=True then check both used and unused UELs and confirm same order, otherwise only check used UELs in data and do not check UEL order. If check_element_text=True then check that all set elements have the same descriptive element text, otherwise skip. If check_meta_data=True then check that symbol name and description are the same, otherwise skip. If verbose=True will return an exception from the asserter describing the nature of the difference.	check_uels=True (bool) check_element_text=True (bool) check_meta_data=True (bool) verbose=False (bool)	bool
getSparsity	always 0.0	-	float
getUELs	gets UELs from the Container. Returns only UELs in the data if ignore_unused=True, otherwise return all UELs.	ignore_unused=False (bool)	list
isValid	checks if the symbol is in a valid format, throw exceptions if verbose=True, re-check a symbol if force=True	verbose=False force=True	bool
pivot	Convenience function to pivot records into a new shape (only symbols with >1D can be pivoted). If index is None then it is set to dimensions [0..dimension-1]. If columns is None then it is set to the last dimension. Missing values in the pivot will take the value provided by fill_value	index=None (str, list, None) columns=None (str, list, None) fill_value=None (int, float, str)	pd.DataFrame
toList	convenience method to return symbol records as a Python list	-	list

DomainViolation

DomainViolation objects are convenient containers that store information about the location of domain violations in a symbol. These objects are computed dynamically with the getDomainViolations method and should not be instantiated by the user (they are read-only, to the extent that this is possible in Python). However, the user may be interested in some of the information that they contain.

Constructor

Constructor Arguments/Properties

Argument	Type	Description	Required	Default
`symbol`	`_Symbol`	A reference to the `_Symbol` object that has a domain violation	Yes	-
`dimension`	`int`	An index to the dimension of the `symbol` where the domain violation exists	Yes	-
`domain`	`Set`, `Alias` or `UniverseAlias`	A reference to the symbol domain that is the source of the domain violation	Yes	-
`violations`	`list`	A list of all the domain elements that are causing violations	Yes	-

7.5.4.8 Additional Topics

Validating Data

`transfer` requires that the records for all symbols exist in a standard format (Standard Data Formats) in order for them to be understood by the `Container`. It is certainly possible that the data could end up in a state that is inconsistent with the standard format (especially if setting symbol attributes directly). `transfer` includes the `.isValid()` method in order to determine if a symbol is structurally valid – this method returns a `bool`. This method does not guarantee that a symbol will be successfully written to either GDX or GMD, other data errors (duplicate records, long UEL names, or domain violations) could exist that are not tested in `.isValid()`.

For example, we create two valid sets and then check them with `.isValid()` to be sure.

Note

It is possible to run `.isValid()` on both the `Container` as well as the symbol object – `.isValid()` will also return a `bool` if there are any invalid symbols in the `Container` object.

Example (valid data)

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego", "washington_dc"])
j = gt.Set(m, "j", i, records=["san-diego", "washington_dc"])
In [1]: i.isValid()
Out[1]: True
In [2]: j.isValid()
Out[2]: True
In [3]: m.isValid()
Out[3]: True
```

The `.isValid()` method checks:

1. If the symbol belongs to a Container
2. If all domain set symbols exist in the Container
3. If all domain set symbols objects are valid
4. If any domain set is also a singleton set (not allowed in GAMS)
5. If records are a DataFrame (or `None`)

6. If the records DataFrame is the right number of columns (based on symbol dimension)
7. If the symbol is a scalar, then ensure there is only one record (row) in the DataFrame
8. If records column headings are unique
9. If any symbol attribute columns are missing or out of order
10. If all domain columns are type `category`
11. If all domain categories are type `str`
12. If all data columns are type `float`

Custom Column Headings

The names of the domain columns are flexible, but `transfer` requires unique column names. Users are encouraged to change the column headings of the underlying dataframe by using the `domain_labels` property. Using this property will ensure that unique column names are generated by adding a `_<dimension>` tag to the end of any user supplied column names. The following examples show this behavior.

Attention

> All $*$ domains are recast as `uni`. This allows users to access the column data with both the Pandas bracket and/or dot notation (i.e., `df["uni"]` or `df.uni`).

Column heading behavior at symbol instantiation

The `setRecords` (which is called internally at symbol instantiation) method will set default `domain_labels` if they were not provided by the user. The only way for a user to provide domain labels with `setRecords` is by passing in a Pandas DataFrame object. The `_<dimension>` tag will be added to all domain labels in order to make all domain names unique – this tag is added to all dimensions if any subset of the domain names are non-unique.

```
import gams.transfer as gt
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
# define a symbol with unique domain names
a = gt.Parameter(
    m, "a", [i, "*"], records=[("i1", "u1", 1), ("i2", "u2", 1), ("i3", "u3", 1)]
)
# define a symbol with NON-unique domain names
b = gt.Parameter(
    m, "b", [i, i], records=[("i1", "i1", 1), ("i2", "i2", 1), ("i3", "i3", 1)]
)
# define a symbol from a dataframe that already has column names
df = pd.DataFrame(
    [("i1", "i1", 1), ("i2", "i2", 1), ("i3", "i3", 1)],
    columns=["from", "to", "distance"],
)
c = gt.Parameter(m, "c", [i, i], records=df)
In [1]: m.isValid()
Out[1]: True
In [2]: i.records
Out[2]:
  uni element_text
0  i1
1  i2
2  i3
In [3]: a.records
Out[3]:
    i uni  value
0  i1  u1    1.0
1  i2  u2    1.0
2  i3  u3    1.0
In [4]: b.records
```

```
Out[4]:
  i_0 i_1  value
0  i1  i1    1.0
1  i2  i2    1.0
2  i3  i3    1.0
In [5]: c.records
Out[5]:
  from  to  value
0   i1  i1    1.0
1   i2  i2    1.0
2   i3  i3    1.0
```

Customizing column headings

Many users may want to output the GAMS DataFrame directly to another format (CSV, etc.) and may wish to create customized DataFrame column headings for readability. User can do this by directly setting the `domain_labels` property, as seen in the following example.

Attention

> Users are encouraged to use the `<symbol>.domain_labels` property instead of setting the `<DataFrame>.columns` directly. The avoids the possibility of out-of-sync symbol validity. The `domain_labels` property does not store anything, calling this property simply returns the exact domain labels from the DataFrame.

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
# define a symbol with unique domain names
a = gt.Parameter(
    m, "a", [i, "*"], records=[("i1", "u1", 1), ("i2", "u2", 1), ("i3", "u3", 1)]
)
# customize the headings to allow more friendly output (to csv, etc.)
a.domain_labels = ["start", "destination"]
In [1]: m.isValid()
Out[1]: True
In [2]: a.records
Out[2]:
  start destination  value
0    i1          u1    1.0
1    i2          u2    1.0
2    i3          u3    1.0
```

Converting Records

All data in `transfer` will be stored as a Pandas DataFrame – however, it is desirable to have easy access to data without the additional infrastructure that comes with the DataFrame object. We include `to*` methods (available for all symbol types) that will return other data structures. The following examples show the behavior of `toValue`, `toList`, and `toDict` (previous examples showed examples of `toDense` and `toSparseCoo`).

Examples of toList

```
import gams.transfer as gt
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", records=["i0", "i1", "i2"])
ii = gt.Set(
    m, "ii", ["*", "*"], records=[(f"i{i}", f"i{i}", f"element_text for i{i}") for i in range(3)]
)
s = gt.Set(m, "s", i, is_singleton=True, records="i1")
u = gt.UniverseAlias(m, "u")
ip = gt.Alias(m, "ip", i)
a0 = gt.Parameter(m, "a0", records=1)
a1 = gt.Parameter(m, "a1", i, records=[("i1", 1), ("i2", 2)])
```

```
a2 = gt.Parameter(m, "a2", [i, i], records=[("i1", "i1", 1), ("i2", "i1", 2)])
v0 = gt.Variable(m, "v0", "free", records=1)
v1 = gt.Variable(
    m,
    "v1",
    "free",
    i,
    records=pd.DataFrame([("i1", 1), ("i2", 2)], columns=["i", "level"]),
)
v2 = gt.Variable(
    m,
    "v2",
    "free",
    [i, i],
    records=(
        pd.DataFrame(
            [("i1", "i1", 1), ("i2", "i1", 2)],
            columns=["i", "i", "level"],
        )
    ),
)
In [1]: i.toList()
Out[1]: ['i0', 'i1', 'i2']
In [2]: ii.toList()
Out[2]: [('i0', 'i0'), ('i1', 'i1'), ('i2', 'i2')]
In [3]: ii.toList(include_element_text=True)
Out[3]:
[('i0', 'i0', 'element_text for i0'),
 ('i1', 'i1', 'element_text for i1'),
 ('i2', 'i2', 'element_text for i2')]
In [4]: s.toList()
Out[4]: ['i1']
In [5]: u.toList()
Out[5]: ['i0', 'i1', 'i2']
In [6]: ip.toList()
Out[6]: ['i0', 'i1', 'i2']
In [7]: a0.toList()
Out[7]: [1.0]
In [8]: a1.toList()
Out[8]: [('i1', 1.0), ('i2', 2.0)]
In [9]: a2.toList()
Out[9]: [('i1', 'i1', 1.0), ('i2', 'i1', 2.0)]
In [10]: v0.toList() # default is to include only the "level"
Out[10]: [1.0]
In [11]: v0.toList("marginal")
Out[11]: [0.0]
In [12]: v1.toList() # default is to include only the "level"
Out[12]: [('i1', 1.0), ('i2', 2.0)]
In [13]: v1.toList(["level", "marginal"])
Out[13]: [('i1', 1.0, 0.0), ('i2', 2.0, 0.0)]
```

Examples of toValue

```
In [1]: a0.toValue()
Out[1]: 1.0
In [2]: a1.toValue()
Out[2]: TypeError: Cannot extract value data for non-scalar symbols (symbol dimension is 1)
In [3]: v0.toValue() # default is to only include the "level"
Out[3]: 1.0
In [4]: v0.toValue("marginal")
Out[4]: 0.0
```

Examples of toDict

```
In [1]: i.toDict()
Out[1]: AttributeError: 'Set' object has no attribute 'toDict'
In [2]: a0.toDict()
Out[2]: TypeError: Symbol `a0` is a scalar and cannot be converted into a dict.
In [3]: a1.toDict()
Out[3]: {'i1': 1.0, 'i2': 2.0}
In [4]: a2.toDict()
Out[4]: {('i1', 'i1'): 1.0, ('i2', 'i1'): 2.0}
In [5]: v1.toDict() # default is to only include the "level"
Out[5]: {'i1': 1.0, 'i2': 2.0}
In [6]: v1.toDict(["level","marginal"])
Out[6]: {'i1': {'level': 1.0, 'marginal': 0.0}, 'i2': {'level': 2.0, 'marginal': 0.0}}
In [7]: v1.toDict(orient="columns") # this format is useful for recreating Pandas DataFrames
Out[7]: {'i': {0: 'i1', 1: 'i2'}, 'level': {0: 1.0, 1: 2.0}}
```

Comparing Symbols

Sparse GAMS data is inherently unordered. The concept of order is GAMS is governed by the order of the UELs in the universe set not the order of the records. This differs from the sparse data structures that we use in `transfer` (Pandas DataFrames) because each record (i.e., DataFrame row) has an index (typically `0..n`) and can be sorted by this index. Said a slightly different way, two GDX files will be equivalent if their universe order is the same and the records are the same, however when creating the GDX file, it is of no consequence what order the records are written in. Therefore, in order to calculate an equality between two symbols in `transfer` we must perform a merge operation on the symbol domain labels – an operation that could be computationally expensive for large symbols.

Attention

> The nature of symbol equality in `transfer` means that a potentially expensive merge operation is performed, we do not recommend that the `equals` method be used inside loops or when speed is critical. It is, however, very useful for data debugging.

A quick example shows the syntax of `equals`:

```
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(5)], description="set i")
j = gt.Set(m, "j", records=[f"i{i}" for i in range(5)], description="set j")
In [1]: i.equals(j)
Out[1]: False
```

By default, `equals` takes the strictest view of symbol "equality" – everything must be the same. In this case, the symbol names and descriptions differ between the two sets `i` and `j`. We can relax the view of equality with a combination of argument flags. Comparing the two symbols again, but ignoring the meta data (i.e., ignoring the symbol name, description and type (if a Variable or Equation)):

```
In [1]: i.equals(j, check_meta_data=False)
Out[1]: True
```

It is also possible to ignore the set element text in `equals`:

```
m = gt.Container()
i = gt.Set(m, "i", records=[(f"i{i}", "arlington") for i in range(5)])
j = gt.Set(m, "j", records=[f"i{i}" for i in range(5)])
In [1]: i.records
Out[1]:
  uni   element_text
0    i0    arlington
1    i1    arlington
2    i2    arlington
3    i3    arlington
4    i4    arlington
In [2]: j.records
Out[2]:
  uni   element_text
0    i0
1    i1
2    i2
3    i3
4    i4
In [3]: i.equals(j, check_meta_data=False, check_element_text=False)
Out[3]: True
```

The `check_uels` argument will ensure that the symbol "universe" is the same (in order and content) between two symbols, as illustrated in the following example:

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
ip = gt.Set(m, "ip", records=["i3", "i2", "i1"])
```

Clearly, the two sets `i` and `ip` have the same records, but the UEL order is different. If `check_uels=True` the resulting symbols will not be considered equal – turning this flag off results in equality.

```
In [1]: i.getUELs()
Out[1]: ['i1', 'i2', 'i3']
In [2]: ip.getUELs()
Out[2]: ['i3', 'i2', 'i1']
In [3]: i.equals(ip, check_meta_data=False)
Out[3]: False
In [4]: i.equals(ip, check_meta_data=False, check_uels=False)
Out[4]: True
```

Numerical comparisons are enabled for `Parameters`, `Variables` and `Equations` – equality can be flexibly defined through the `equals` method arguments. Again, the strictest view of equality is taken as the default behavior of `equals` (no numerical tolerances, some limitations exist – see: `numpy.isclose` for more details).

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
a = gt.Parameter(m, "a", i, records=[("i1", 1), ("i2", 2), ("i3", 3)])
ap = gt.Parameter(m, "ap", i, records=[("i1", 1 + 1e-9), ("i2", 2), ("i3", 3)])
In [1]: a.equals(ap, check_meta_data=False)
Out[1]: False
In [2]: a.equals(ap, check_meta_data=False, atol=1e-8)
Out[2]: True
```

Attention

> The numerical comparison is handled by `numpy.isclose`, more details can be found in the Numpy documentation.

In the case of variables and equations, it is possible for the user to confine the numerical comparison to certain certain attributes (`level`, `marginal`, `lower`, `upper` and `scale`) by specifying the `columns` argument as the following example illustrates:

```
m = gt.Container()
a = gt.Variable(m, "a", "free", records=100)
ap = gt.Variable(m, "ap", "free", records=101)
In [1]: a.records
Out[1]:
   level  marginal  lower  upper  scale
0  100.0       0.0   -inf    inf    1.0
In [2]: ap.records
Out[2]:
   level  marginal  lower  upper  scale
0  101.0       0.0   -inf    inf    1.0
In [3]: a.equals(ap, check_meta_data=False)
Out[3]: False
In [4]: a.equals(ap, check_meta_data=False, columns="level")
Out[4]: False
In [5]: a.equals(ap, check_meta_data=False, columns="marginal")
Out[5]: True
```

Domain Forwarding

GAMS includes the ability to define sets directly from data using the implicit set notation (see: Implicit Set Definition (or: Domain Defining Symbol Declarations)). This notation has an analogue in `transfer` called `domain_forwarding`.

Note

> It is possible to recursively update a subset tree in `transfer`.

Domain forwarding is available as an argument to all symbol object constructors; the user would simply need to pass `domain_forwarding=True`.

In this example we have raw data that in the `dist` DataFrame and we want to send the domain information into the `i` and `j` sets – we take care to pass the set objects as the domain for parameter `c`.

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i")
j = gt.Set(m, "j")
dist = pd.DataFrame(
    [
        ("seattle", "new-york", 2.5),
        ("seattle", "chicago", 1.7),
        ("seattle", "topeka", 1.8),
        ("san-diego", "new-york", 2.5),
        ("san-diego", "chicago", 1.8),
        ("san-diego", "topeka", 1.4),
    ],
    columns=["from", "to", "thousand_miles"],
)
```

```
c = gt.Parameter(m, "c", [i, j], records=dist, domain_forwarding=True)
In [1]: i.records
Out[1]:
       uni   element_text
0    seattle
1  san-diego
In [2]: j.records
Out[2]:
      uni   element_text
0  new-york
1   chicago
2    topeka
In [3]: c.records
Out[3]:
         i         j    value
0    seattle  new-york    2.5
1    seattle   chicago    1.7
2    seattle    topeka    1.8
3  san-diego  new-york    2.5
4  san-diego   chicago    1.8
5  san-diego    topeka    1.4
```

Note

The element order in the sets `i` and `j` mirrors that in the raw data.

In this example we show that domain forwarding will also work recursively to update the entire set lineage – the domain forwarding occurs at the creation of every symbol object. The correct order of elements in set `i` is `[z, a, b, c]` because the records from `j` are forwarded first, and then the records from `k` are propagated through (back to `i`).

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i")
j = gt.Set(m, "j", i, records=["z"], domain_forwarding=True)
k = gt.Set(m, "k", j, records=["a", "b", "c"], domain_forwarding=True)
In [1]: i.records
Out[1]:
  uni   element_text
0     z
1     a
2     b
3     c
In [2]: j.records
Out[2]:
  i   element_text
0   z
1   a
2   b
3   c
In [3]: k.records
Out[3]:
  j   element_text
0   a
1   b
2   c
```

It is also possible to forward to specific domain sets by passing a `list` of `bool` to the `domain_forwarding` property, as seen in the following example:

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i")
j = gt.Set(m, "j")
k = gt.Set(m, "k")
ijk = gt.Parameter(
    m,
    "ijk",
    [i, j, k],
    records=[("i", "j", "k", 1)],
domain_forwarding=[True, False, True],
)
In [1]: i.records
Out[1]:
  uni element_text
0   i
In [2]: j.records is None
Out[2]: True
In [3]: k.records
Out[3]:
  uni element_text
0   k
```

Domain Violations

Domain violations occur when domain labels appear in symbol data but they do not appear in the parent set which the symbol is defined over – attempting to execute a GAMS model when there domain violations will lead to compilation errors. Domain violations are found dynamically with the <Symbol>.findDomainViolations() method.

Note

the findDomainViolations method can be computationally expensive – UELs in GAMS are case preserving (just like symbol names); additionally, GAMS ignores all trailing white space in UELs (leading white space is considered significant). As a result, transfer must lowercase all UELs and then strip any trailing white space before doing the set comparison to locate (and create) any DomainViolation objects. findDomainViolations should not be used in a loop (nor should any of its related methods: hasDomainViolations, countDomainViolations, getDomainViolations, or dropDomainViolations).

In the following example we intentionally create data with domain violations in the a parameter:

```
m = gt.Container()
i = gt.Set(m, "i", records=["a", "b", "c"])
a = gt.Parameter(m, "a", i, records=[("aa", 1), ("c", 2)])
In [1]: a.findDomainViolations()
Out[1]:
  i    value
0  aa    1.0
In [2]: a.hasDomainViolations()
Out[2]: True
In [3]: a.countDomainViolations()
Out[3]: 1
In [4]: a.getDomainViolations()
Out[4]: [<DomainViolation at 0x7fb6b83d9630>]
```

Dynamically locating domain violations allows transfer to return a view of the underlying pandas dataframe with the problematic domain labels still intact – at this point the user is free to correct issues in the UELs with any of the *UELs methods or by simply dropping any domain violations from the dataframe completely (the dropDomainViolations method is a convenience function for this operation).

Attention

It is not possible to create a GDX file if symbols have domain violations.

Unused UELs will not result in domain violations.

Attempting to write this container to a GDX file will result in an exception.

```
m = gt.Container()
i = gt.Set(m, "i", records=["a", "b", "c"])
a = gt.Parameter(m, "a", i, records=[("aa", 1), ("c", 2)])
m.write("out.gdx")
Exception: Encountered data errors with symbol ‘a‘. Possible causes are from duplicate records and/or domain violations.
Use ’hasDuplicateRecords’, ’findDuplicateRecords’, ’dropDuplicateRecords’, and/or ’countDuplicateRecords’ to find/resolve
      duplicate records.
Use ’hasDomainViolations’, ’findDomainViolations’, ’dropDomainViolations’, and/or ’countDomainViolations’ to find/resolve
      domain violations.
GDX file was not created successfully.
```

Duplicate Records

Duplicate records can easily appear in large datasets – locating and fixing these records is straightforward with transfer. transfer includes find*, has*, count* and drop* methods for duplicate records, just as it has for domain violations.

Note

the `findDuplicateRecords` method can be computationally expensive – UELs in GAMS are case preserving (just like symbol names); additionally, GAMS ignores all trailing white space in UELs (leading white space is considered significant). As a result, `transfer` must lowercase all UELs and then strip any trailing white space before doing the set comparison to locate duplicate records. `findDuplicateRecords` should not be used in a loop (nor should any of its related methods: `hasDuplicateRecords`, `countDuplicateRecords`, or `dropDuplicateRecords`).

Dynamically locating duplicate records allows `transfer` to return a view of the underlying pandas dataframe with the problematic domain labels still intact – at this point the user is free to correct issues in the UELs with any of the `*UELs` methods or by simply dropping any duplicate records from the dataframe completely (the `dropDuplicateRecords` method is a convenience function for this operation).

```
m = gt.Container()
a = gt.Parameter(
    m,
    "a",
    ["*"],
    records=[("i" + str(i), float(i)) for i in range(4)]
    + [("j" + str(i), i) for i in range(4)]
    + [("I" + str(i), i) for i in range(4)],
)
```

Note

The user can decide which duplicate records they would like `keep` with `keep="first"` (default), `keep="last"`, or `keep=False` (which returns all duplicate records)

```
In [1]: a.records
Out[1]:
   uni    value
0     i0    0.0
1     i1    1.0
2     i2    2.0
3     i3    3.0
4     j0    0.0
5     j1    1.0
6     j2    2.0
7     j3    3.0
8     I0    0.0
9     I1    1.0
10    I2    2.0
11    I3    3.0
In [2]: a.findDuplicateRecords()
Out[2]:
   uni    value
8     I0    0.0
9     I1    1.0
10    I2    2.0
11    I3    3.0
In [3]: a.findDuplicateRecords(keep="last")
Out[3]:
  uni    value
0    i0    0.0
1    i1    1.0
2    i2    2.0
3    i3    3.0
In [4]: a.findDuplicateRecords(keep=False)
Out[4]:
   uni    value
0     i0    0.0
1     i1    1.0
2     i2    2.0
3     i3    3.0
8     I0    0.0
9     I1    1.0
10    I2    2.0
11    I3    3.0
```

Attention

It is not possible to create a GDX file if symbols have duplicate records.

Attempting to write this container to a GDX file will result in an exception.

```
m = gt.Container()
```

```
a = gt.Parameter(
    m,
    "a",
    ["*"],
    records=[("i" + str(i), float(i)) for i in range(4)]
    + [("j" + str(i), i) for i in range(4)]
    + [("I" + str(i), i) for i in range(4)],
)
m.write("out.gdx")
Exception: Encountered data errors with symbol 'a'. Possible causes are from duplicate records and/or domain violations.
Use 'hasDuplicateRecords', 'findDuplicateRecords', 'dropDuplicateRecords', and/or 'countDuplicateRecords' to find/resolve
      duplicate records.
Use 'hasDomainViolations', 'findDomainViolations', 'dropDomainViolations', and/or 'countDomainViolations' to find/resolve
      domain violations.
GDX file was not created successfully.
```

Pivoting Data

It might be convenient to pivot data into a multi-dimensional data structure rather than maintaining the flat structure in `records`. A convenience method called `pivot` is provided for all symbol classes and will return a pivoted `pandas.DataFrame`. Pivoting is only available for symbols with more than one dimension.

Example #1 - Pivot a 2D Set

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(5)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(5)])
ij = gt.Set(m, "ij", [i, j])
ij.generateRecords(density=0.25, seed=123)
In [1]: ij.pivot()
Out[1]:
       j0     j1     j3     j4
i0   True   True  False  False
i1   True  False  False  False
i2  False  False   True   True
i4  False   True  False  False
```

Example #2 - Pivot a 3D Set

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(5)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(5)])
iji = gt.Set(m, "iji", [i, j, i])
iji.generateRecords(density=0.25, seed=123)
In [1]: iji.pivot()
Out[1]:
          i0     i1     i2     i3     i4
i0 j0  False   True   True  False  False
   j1   True  False  False  False  False
   j3  False  False  False   True  False
   j4  False  False   True  False  False
i1 j0   True   True  False   True  False
   j1   True   True  False  False   True
   j2  False   True  False  False  False
   j4  False  False  False   True  False
i2 j0   True  False  False  False  False
   j1  False  False   True  False   True
   j3   True  False  False  False  False
i3 j2  False   True   True  False   True
   j3  False   True  False  False  False
   j4   True  False   True   True   True
i4 j0  False   True  False   True  False
   j1  False  False  False   True  False
   j3  False  False  False   True  False
   j4  False  False  False   True   True
In [2]: iji.pivot(fill_value="")
Out[2]:
         i0    i1    i2    i3    i4
i0 j0        True  True
   j1  True
   j3                   True
```

```
   j4             True
i1 j0  True  True       True
   j1  True  True             True
   j2       True
   j4                 True
i2 j0  True
   j1             True       True
   j3  True
i3 j2       True  True       True
   j3       True
   j4  True       True  True  True
i4 j0       True       True
   j1                  True
   j3                  True
   j4                  True  True
```

Note

When pivoting symbols with >2 dimensions, the first [0..(dimension-1)] dimensions will be set to the index and the last dimension will be pivoted into the columns. This behavior can be customized with the `index` and `columns` arguments.

Example #3 - Pivot a 3D Parameter w/ a fill_value

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(5)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(5)])
iji = gt.Parameter(m, "iji", [i, j, i])
iji.generateRecords(density=0.05, seed=123)
In [1]: iji.pivot(fill_value="NONE")
Out[1]:
             i1        i2        i3        i4
i0 j1  0.682352      NONE      NONE      NONE
   j2  0.053821      NONE   0.22036      NONE
i1 j1      NONE      NONE      NONE  0.184372
i2 j0      NONE  0.175906      NONE      NONE
i3 j4      NONE      NONE  0.812095      NONE
In [2]: iji.pivot(fill_value=0)
Out[2]:
             i1        i2        i3        i4
i0 j1  0.682352  0.000000  0.000000  0.000000
   j2  0.053821  0.000000  0.220360  0.000000
i1 j1  0.000000  0.000000  0.000000  0.184372
i2 j0  0.000000  0.175906  0.000000  0.000000
i3 j4  0.000000  0.000000  0.812095  0.000000
In [3]: iji.pivot(fill_value=gt.SpecialValues.EPS)
Out[3]:
             i1        i2        i3        i4
i0 j1  0.682352 -0.000000 -0.000000 -0.000000
   j2  0.053821 -0.000000  0.220360 -0.000000
i1 j1 -0.000000 -0.000000 -0.000000  0.184372
i2 j0 -0.000000  0.175906 -0.000000 -0.000000
i3 j4 -0.000000 -0.000000  0.812095 -0.000000
```

Example #4 - Pivot (only the marginal values) of a 3D Variable

```
import gams.transfer as gt
# NOTE: custom functions should expose a 'seed' argument
def marginal_values(seed, size):
    rng = np.random.default_rng(seed)
    return rng.normal(5, 1.2, size=size)
m = gt.Container()
i = gt.Set(m, "i", records=[f"i{i}" for i in range(5)])
j = gt.Set(m, "j", records=[f"j{i}" for i in range(5)])
iji = gt.Variable(m, "iji", "free", [i, j, i])
iji.generateRecords(density=0.05, func={"marginal": marginal_values}, seed=123)
In [1]: iji.records
Out[1]:
  i_0 j_1 i_2  level  marginal  lower  upper  scale
0  i0  j1  i1    0.0  3.813054   -inf    inf    1.0
1  i0  j2  i1    0.0  4.558656   -inf    inf    1.0
2  i0  j2  i3    0.0  6.545510   -inf    inf    1.0
3  i1  j1  i4    0.0  5.232769   -inf    inf    1.0
```

```
4  i2  j0  i2    0.0  6.104277   -inf    inf    1.0
5  i3  j4  i3    0.0  5.692525   -inf    inf    1.0
In [2]: iji.pivot(value="marginal")
Out[2]:
             i1        i3        i4        i2
i0 j1  3.813054  0.000000  0.000000  0.000000
   j2  4.558656  6.545510  0.000000  0.000000
i1 j1  0.000000  0.000000  5.232769  0.000000
i2 j0  0.000000  0.000000  0.000000  6.104277
i3 j4  0.000000  5.692525  0.000000  0.000000
```

Describing Data

The methods `describeSets`, `describeParameters`, `describeVariables`, and `describeEquations` allow the user to get a summary view of key data statistics. The returned DataFrame aggregates the output for a number of other methods (depending on symbol type). A description of each `Container` method is provided in the following subsections:

describeSets

Argument	Type	Description	Required	Default
`symbols`	`list`, `str`, NoneType	A list of sets in the `Container` to include in the output. describeSets will include aliases if they are explicitly passed by the user.	No	`None` (if `None` specified, will assume all sets – not aliases)

Returns: `pandas.DataFrame`

The following table includes a short description of the column headings in the return.

Property / Statistic	Description
`name`	name of the symbol
`is_singleton`	`bool` if the set/alias is a singleton set (or an alias of a singleton set)
`alias_with`	[OPTIONAL if users passes an alias name as part of `symbols`] name of the parent set (for alias only), None otherwise
`domain`	domain labels for the symbol
`domain_type`	`none`, `relaxed` or `regular` depending on the symbol state
`dimension`	dimension
`number_records`	number of records in the symbol
`sparsity`	`1 - number_records/cardinality`

Example #1

```
import gams.transfer as gt
m = gt.Container("trnsport.gdx")
In [1]: m.describeSets()
Out[1]:
  name  is_singleton domain domain_type  dimension  number_records sparsity
0    i         False    [*]        none          1               2     None
1    j         False    [*]        none          1               3     None
```

Example #2 – with aliases

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i" + str(i) for i in range(1, 10)])
j = gt.Set(m, "j", records=["j" + str(i) for i in range(1, 10)])
ip = gt.Alias(m, "ip", i)
jp = gt.Alias(m, "jp", j)
In [1]: m.describeSets()
Out[1]:
  name  is_singleton domain domain_type  dimension  number_records sparsity
0    i         False    [*]        none          1               9     None
1    j         False    [*]        none          1               9     None
In [2]: m.describeSets(m.listSets() + m.listAliases())
Out[2]:
  name  is_singleton  is_alias alias_with domain domain_type  dimension  number_records sparsity
0    i         False     False       None    [*]        none          1               9     None
1   ip         False      True          i    [*]        none          1               9     None
2    j         False     False       None    [*]        none          1               9     None
3   jp         False      True          j    [*]        none          1               9     None
```

describeParameters

Argument	Type	Description	Required	Default
`symbols`	`list`, `str`, NoneType	A list of parameters in the `Container` to include in the output	No	`None` (if `None` specified, will assume all parameters)

Returns: `pandas.DataFrame`

The following table includes a short description of the column headings in the return.

Property / Statistic	Description
`name`	name of the symbol
`domain`	domain labels for the symbol
`domain_type`	`none`, `relaxed` or `regular` depending on the symbol state
`dimension`	dimension
`number_records`	number of records in the symbol
`min`	min value in data
`mean`	mean value in data
`max`	max value in data
`where_min`	domain of min value (if multiple, returns only first occurrence)
`where_max`	domain of max value (if multiple, returns only first occurrence)
`sparsity`	`1 - number_records/cardinality`

Example

```
import gams.transfer as gt
m = gt.Container("trnsport.gdx")
In [1]: m.describeParameters()
Out[1]:
  name  domain domain_type  dimension  number_records      min     mean      max           where_min           where_max
     sparsity
0    a     [i]     regular          1               2  350.000  475.000  600.000           [seattle]          [san-diego]
      0.0
1    b     [j]     regular          1               3  275.000  300.000  325.000            [topeka]           [new-york]
      0.0
```

```
2    c  [i, j]     regular          2              6    0.126    0.176    0.225  [san-diego, topeka]  [seattle, new-york]
      0.0
3    d  [i, j]     regular          2              6    1.400    1.950    2.500  [san-diego, topeka]  [seattle, new-york]
      0.0
4    f      []        none          0              1   90.000   90.000   90.000                 None                 None
      None
```

describeVariables

Argument	Type	Description	Required	Default
`symbols`	`list`, `str`, NoneType	A list of variables in the `Container` to include in the output	No	`None` (if `None` specified, will assume all variables)

Returns: `pandas.DataFrame`

The following table includes a short description of the column headings in the return.

Property / Statistic	Description
`name`	name of the symbol
`type`	type of variable (i.e., `binary`, `integer`, `positive`, `negative`, `free`, `sos1`, `sos2`, `semicont`, `semiint`)
`domain`	domain labels for the symbol
`domain_type`	`none`, `relaxed` or `regular` depending on the symbol state
`dimension`	dimension
`number_records`	number of records in the symbol
`sparsity`	`1 - number_records/cardinality`
`min_level`	min value in the `level`
`mean_level`	mean value in the `level`
`max_level`	max value in the `level`
`where_max_abs_level`	domain of max(abs(`level`)) in data

Example

```
import gams.transfer as gt
m = gt.Container("trnsport.gdx")
In [1]: m.describeVariables()
Out[1]:
  name      type  domain domain_type  dimension  number_records sparsity  min_level  mean_level  max_level
      where_max_abs_level
0    x  positive  [i, j]     regular          2              6      0.0      0.000     150.000    300.000  [seattle,
      chicago]
1    z      free      []        none          0              1     None    153.675     153.675    153.675
      None
```

describeEquations

Argument	Type	Description	Required	Default
`symbols`	`list`, `str`, NoneType	A list of equations in the `Container` to include in the output	No	`None` (if `None` specified, will assume all equations)

Returns: `pandas.DataFrame`

The following table includes a short description of the column headings in the return.

Property / Statistic	Description
`name`	name of the symbol
`type`	type of variable (i.e., `binary`, `integer`, `positive`, `negative`, `free`, `sos1`, `sos2`, `semicont`, `semiint`)
`domain`	domain labels for the symbol
`domain_type`	`none`, `relaxed` or `regular` depending on the symbol state
`dimension`	dimension
`number_records`	number of records in the symbol
`sparsity`	`1 - number_records/cardinality`
`min_level`	min value in the `level`
`mean_level`	mean value in the `level`
`max_level`	max value in the `level`
`where_max_abs_level`	domain of max(abs(`level`)) in data

Example

```
import gams.transfer as gt
m = gt.Container("trnsport.gdx")
In [1]: m.describeEquations()
Out[1]:
     name type domain domain_type  dimension  number_records sparsity  min_level  mean_level  max_level where_max_abs_level
0    cost   eq     []         none          0               1     None       -0.0         0.0       -0.0
      None
1  demand  geq    [j]      regular          1               3      0.0      275.0       300.0      325.0
      [new-york]
2  supply  leq    [i]      regular          1               2      0.0      350.0       450.0      550.0
      [san-diego]
```

describeAliases

Argument	Type	Description	Required	Default
`symbols`	`list`, `str`, NoneType	A list of alias (only) symbols in the `Container` to include in the output	No	`None` (if `None` specified, will assume all aliases – not sets)

Returns: `pandas.DataFrame`

The following table includes a short description of the column headings in the return. All data is referenced from the parent set that the alias is created from.

Property / Statistic	Description
`name`	name of the symbol
`alias_with`	name of the parent set (for alias only), None otherwise
`is_singleton`	`bool` if the set/alias is a singleton set (or an alias of a singleton set)
`domain`	domain labels for the symbol
`domain_type`	`none`, `relaxed` or `regular` depending on the symbol state

Property / Statistic	Description
`dimension`	dimension
`number_records`	number of records in the symbol
`sparsity`	`1 - number_records/cardinality`

Example

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", records=["j" + str(j) for j in range(10)])
ip = gt.Alias(m, "ip", i)
ipp = gt.Alias(m, "ipp", ip)
jp = gt.Alias(m, "jp", j)
In [1]: m.describeAliases()
Out[1]:
  name alias_with  is_singleton domain domain_type  dimension  number_records sparsity
0   ip          i         False    [*]        none          1               5     None
1  ipp          i         False    [*]        none          1               5     None
2   jp          j         False    [*]        none          1              10     None
```

Matrix Generation

`transfer` stores data in a "flat" format, that is, one record entry per DataFrame row. However, it is often necessary to convert this data format into a matrix format – `transfer` enables users to do this with relative ease using the `toDense` and the `toSparseCoo` symbol methods. The `toDense` method will return a dense N-dimensional numpy array with each dimension corresponding to the GAMS symbol dimension; it is possible to output an array up to 20 dimensions (a GAMS limit). The `toSparseCoo` method will return the data in a sparse scipy COOrdinate format, which can then be efficiently converted into other sparse matrix formats.

Attention

Both the `toDense` and `toSparseCoo` methods do not transform the underlying DataFrame in any way, they only return the transformed data.

Note

`toSparseCoo` will only convert 2-dimensional data to the scipy COOrdinate format. A user interested in sparse data for an N-dimensional symbol will need to decide how to reshape the dense array in order to generate the 2D sparse format.

Attention

In order to use the `toSparseCoo` method the user will need to install the scipy package. Scipy is not provided with GMSPython.

Both the `toDense` and `toSparseCoo` method leverage the indexing that comes along with using `categorical` data types to store domain information. This means that linking symbols together (by passing symbol objects as domain information) impacts the size of the matrix. This is best demonstrated by a few examples.

Example (1D data w/o domain linking (i.e., a relaxed domain))

```
import gams.transfer as gt
m = gt.Container()
a = gt.Parameter(m, "a", "i", records=[("a", 1), ("c", 3)])
In [1]: a.records
Out[1]:
  i    value
0   a    1.0
1   c    3.0
In [2]: a.toDense()
Out[2]: array([1., 3.])
In [3]: a.toSparseCoo()
Out[3]:
<1x2 sparse matrix of type '<class 'numpy.float64'>'
    with 2 stored elements in COOrdinate format>
```

Note that the parameter `a` is not linked to another symbol, so when converting to a matrix, the indexing is referenced to the data structure in `a.records`. Defining a sparse parameter `a` over a set `i` allows us to extract information from the `i` domain and construct a very different dense matrix, as the following example shows:

Example (1D data w/ domain linking (i.e., a regular domain))

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["a", "b", "c", "d"])
a = gt.Parameter(m, "a", i, records=[("a", 1), ("c", 3)])
In [1]: i.records
Out[1]:
  uni   element_text
0     a
1     b
2     c
3     d
In [2]: a.records
Out[2]:
  i    value
0   a    1.0
1   c    3.0
In [3]: a.toDense()
Out[3]: array([1., 0., 3., 0.])
In [4]: a.toSparseCoo()
Out[4]:
<1x4 sparse matrix of type '<class 'numpy.float64'>'
    with 2 stored elements in COOrdinate format>
```

Example (2D data w/ domain linking)

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["a", "b", "c", "d"])
a = gt.Parameter(m, "a", [i, i], records=[("a", "a", 1), ("c", "c", 3)])
In [1]: i.records
Out[1]:
  uni   element_text
0     a
1     b
2     c
3     d
In [2]: a.records
Out[2]:
  i_0 i_1  value
0   a   a    1.0
1   c   c    3.0
In [3]: a.toDense()
Out[3]:
array([[1., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 3., 0.],
       [0., 0., 0., 0.]])
In [4]: a.toSparseCoo()
Out[4]:
<4x4 sparse matrix of type '<class 'numpy.float64'>'
    with 2 stored elements in COOrdinate format>
```

The Universe Set

A Unique Element (UEL) is an `(i,s)` pair where `i` (or index) is an identification number for a (string) label `s`. GAMS uses UELs to efficiently store domain entries of a record by storing the UEL ID `i` of a domain entry instead of the actual string `s`. This avoids storing the same string multiple times. The concept of UELs also exists in Python/Pandas and is called a "categorical series". `transfer` leverages these types in order to efficiently store strings and enable domain checking within the Python environment.

Each domain column in a DataFrame can be assigned a unique categorical type, the effect is that each symbol maintains its own list of UELs per dimension. It is possible to convert a categorical column to its ID number representation by using the categorical accessor `x.records[<domain_column_label>].cat.codes`; however, this type of data manipulation is not necessary within `transfer`, but could be handy when debugging data.

Pandas offers the possibility to create categorical column types that are `ordered` or not; `transfer` relies exclusively on `ordered` categorical data types (in order for a symbol to be valid it must have only `ordered` categories). By using ordered categories, `transfer` will order the UEL such that elements appear in the order in which they appeared in the data (which is how GAMS defines the UEL). `transfer` allows the user to reorder the UELs with the `uel_priority` argument in the `.write()` method.

`transfer` does not actually keep track of the UEL separately from other symbols in the `Container`, it will be created internal to the `.write()` method and is based on the order in which data is added to the container. The user can access the current state of the UEL with the `.getUELs()` container method. For example, we set a two dimensional set:

```
import gams.transfer as gt
m = gt.Container()
j = gt.Set(m, "j", ["*", "*"], records=[("i" + str(n), "j" + str(n)) for n in range(2)])
In [1]: j.records
Out[1]:
  uni_0 uni_1 element_text
0    i0    j0
1    i1    j1
In [2]: m.getUELs()
Out[2]: ['i0', 'i1', 'j0', 'j1']
```

Pandas also includes a number of methods that allow categories to be renamed, appended, etc. These methods may be useful for advanced users, but most users will probably find that modifying the original data structures and resetting the symbol records provides a simpler solution. The design of `transfer` should enable the user to quickly move data back and forth, without worrying about the deeper mechanics of categorical data.

Customize the Universe Set

The concept of a universe set is fundamental to GAMS and has consequences in many areas of GAMS programming including the order of loop execution. For example:

```
set final_model_year / 2030 /;
set t "all model years" / 2022*2030 /;

singleton set my(t) "model solve year";


loop(t,
  my(t) = yes;
  display my;
  );
```

The loop will execute model solve year `2030` first because the UEL `2030` was defined in the set `final_model_year` before it was used again in the definition of set `t`. This could lead to some surprising behavior if model time periods are linked together. Many GAMS users would create a dummy set (perhaps the first line of their model file) that contained all the UELs that had a significant order tom combat this behavior. `transfer` allows for full control (renaming as well as ordering) over the universe set through the `*UELS` methods, briefly described here:

Quick summary table of UELs functions

Method	Brief Description
addUELs	Adds UELS to a symbol dimension(s). This function does not have a container level implementation.
capitalizeUELs	Capitalize all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
casefoldUELs	Casefold all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
getUELs	Gets the UELs in a over either a symbol dimension, the entire symbol or the entire container. Unused UELs do not show up in symbol data but will show up in the GAMS UEL list.
ljustUELs	Left justify all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
lowerUELs	Lowercase all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
lstripUELs	Left strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
removeUELs	Removes UELs from a symbol dimension, the entire symbol, the entire container (or just a subset of symbols). If a used UEL is removed the DataFrame record will show a NaN.
renameUELs	Renames UELs in a symbol dimension, the entire symbol, the entire container (or just a subset of symbols). Very handy for harmonizing UEL labeling of data that might have originated from different sources.
reorderUELs	Reorders UELs in a symbol dimension(s). This function does not have a container level implementation.
rjustUELs	Right justify all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
rstripUELs	Right strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
setUELs	Sets UELs for a symbol dimension(s). Equivalent results could be obtained with a combination of renameUELs and reorderUELs, but this one call may have some performance advantage.
stripUELs	Strip whitespace from all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
titleUELs	Title (capitalize all individual words) in all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations
upperUELs	Uppercase all UELs in the symbol or a subset of specified dimensions, can be chained with other *UELs string operations

These tools are extremely useful when data is arriving at a model from a variety of data sources. We will describe each of these functions in detail and provide examples in the following sections.

Attention

GAMS is insensitive to trailing whitespaces, the *UELs methods will automatically trim any trailing whitespace when creating the new UELs.

getUELs Examples

getUELs is a method of all GAMS symbol classes as well as the Container class. This allows the user to retrieve (ordered) UELs from the entire container or just a specific symbol dimension. For example:

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
j = gt.Set(m, "j", i, records=["j1", "j2", "j3"])
```

```
a = gt.Parameter(m, "a", [i, j], records=[(f"i{i}", f"j{i}", i) for i in range(4)])
In [1]: i.getUELs()
Out[1]: ['i1', 'i2', 'i3']
In [2]: m.getUELs()
Out[2]: ['i1', 'i2', 'i3', 'j1', 'j2', 'j3', 'i0', 'j0']
In [3]: m.getUELs("j")
Out[3]: ['j1', 'j2', 'j3']
```

addUELs Examples

`addUELs` is a method of all GAMS symbol classes. This method allows the user to add in new UELs labels to a specific dimension of a symbol – the user can add UELs that do not exist in the symbol `records`. For example:

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
j = gt.Set(m, "j", i, records=["j1", "j2", "j3"])
a = gt.Parameter(m, "a", [i, j], records=[(f"i{i}", f"j{i}", i) for i in range(1,4)])
i.addUELs("ham")
a.addUELs("and", 0)
a.addUELs("cheese", 1)
In [1]: i.getUELs()
Out[1]: ['i1', 'i2', 'i3', 'ham']
In [2]: a.getUELs()
Out[2]: ['i1', 'i2', 'i3', 'and', 'j1', 'j2', 'j3', 'cheese']
```

In this example we have added three new (unused) UELs: `ham`, `and`, `cheese`. These three UELs will now appear in the GAMS universe set (accessible with `m.getUELs()`). The addition of unused UELs does not impact the validity of the symbols (i.e., unused UELs will not trigger domain violations).

removeUELs Examples

`removeUELs` is a method of all GAMS symbol classes as well as the `Container` class. As a result, this method allows the user to clean up unwanted or simply unused UELs in a symbol dimension(s), over several symbols, or over the entire container. The previous example added three unused UELs (`ham`, `and`, `cheese`), but now we want to remove these UELs in order to clean up the GAMS universe set. We can accomplish this several ways:

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
j = gt.Set(m, "j", i, records=["j1", "j2", "j3"])
a = gt.Parameter(m, "a", [i, j], records=[(f"i{i}", f"j{i}", i) for i in range(1,4)])
i.addUELs("ham")
a.addUELs("and", 0)
a.addUELs("cheese", 1)
# remove symbol UELs explicitly by dimension
i.removeUELs("ham", 0)
a.removeUELs("and", 0)
a.removeUELs("cheese", 1)
# remove symbol UELs for the entire symbol
i.removeUELs("ham")
a.removeUELs(["and", "cheese"])
# remove ONLY unused UELs from each symbol, independently
i.removeUELs()
a.removeUELs()
# remove ONLY unused UELs from the entire container (all symbols)
m.removeUELs()
```

In all cases the resulting universe set will be:

```
In [1]: m.getUELs()
Out[1]: ['i1', 'i2', 'i3', 'j1', 'j2', 'j3']
```

If a user removes a UEL that appears in data, that data will be lost permanently. The domain label will be transformed into an `NaN` as seen in this example:

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
j = gt.Set(m, "j", i, records=["j1", "j2", "j3"])
a = gt.Parameter(m, "a", [i, j], records=[(f"i{i}", f"j{i}", i) for i in range(1,4)])
m.removeUELs("i1")
In [1]: i.records
Out[1]:
  uni   element_text
0   NaN
```

```
1    i2
2    i3
In [2]: a.records
Out[2]:
   i   j    value
0  NaN  j1    1.0
1   i2  j2    2.0
2   i3  j3    3.0
```

Attention

A container cannot be written if there are `NaN` entries in any of the domain columns (in any symbol) – an Exception is raised if there are missing domain labels.

renameUELs Examples

`renameUELs` is a method of all GAMS symbol classes as well as the `Container` class. This method allows the user to rename UELs in a symbol dimension(s), over several symbols, or over the entire container. This particular method is very handy when attempting to harmonize labeling schemes between data structures that originated from different sources. For example:

```
m = gt.Container()
a = gt.Parameter(
    m,
    "a",
    ["*", "*"],
    records=[("WI", "IL", 10), ("IL", "IN", 12.5), ("WI", "IN", 8.7)],
    description="shipment quantities",
)
b = gt.Parameter(
    m,
    "b",
    ["*"],
    records=[("wisconsin", 1.2), ("illinois", 1.7), ("indiana", 1.2)],
    description="multipliers",
)
```

...results in the following records:

```
In [1]: a.records
Out[1]:
  uni_0 uni_1  value
0    WI    IL   10.0
1    IL    IN   12.5
2    WI    IN    8.7
In [2]: b.records
Out[2]:
       uni    value
0  wisconsin    1.2
1   illinois    1.7
2    indiana    1.2
```

However, two different data sources were used to generate the parameters `a` and `b` – one data source used the uppercase postal abbreviation of the state name and the other source used a lowercase full state name as the unique identifier. With the following syntax the user would be able to harmonize to a mixed case postal code labeling scheme (without losing any of the original UEL ordering).

```
m.renameUELs(
    {
        "WI": "Wi",
        "IL": "Il",
        "IN": "In",
        "wisconsin": "Wi",
        "illinois": "Il",
        "indiana": "In",
    }
)
```

...results in the following records (and the universe set):

```
In [1]: a.records
Out[1]:
  uni_0 uni_1  value
0    Wi    Il   10.0
1    Il    In   12.5
2    Wi    In    8.7
In [2]: b.records
Out[2]:
```

```
  uni    value
0    Wi    1.2
1    Il    1.7
2    In    1.2
```

The universe set will now be:

```
In [1]: m.getUELs()
Out[1]: ['Wi', 'Il', 'In']
```

It is possible that some data needs to be cleaned and multiple UELs need to be mapped to a single label (within a single dimension). This is not allowed under default behavior because `transfer` assumes that the provided UELs are truly unique (logically and lexicographically) – however, it might be necessary recreate the underlying categorical object to combine `n` (previously unique) UELs into one to establish the necessary logical set links. For example:

```
m = gt.Container()
a = gt.Parameter(
    m,
    "a",
    ["*", "*"],
    records=[("WISCONSIN", "iowa", 10), ("WI", "illinois", 12)],
)
In [1]: a.records
Out[1]:
      uni_0     uni_1  value
0  WISCONSIN      iowa   10.0
1         WI  illinois   12.0
```

The records are unique for `a`, but logically, there might be a need to rename `WI` to `WISCONSIN`.

```
In [1]: a.renameUELs({"WI": "WISCONSIN"})
Out[1]: Exception: Could not rename UELs (categories) in ‘a‘ dimension ‘0‘. Reason: Categorical categories must be unique
```

In order achieve the desired behavior it is necessary to pass `allow_merge=True` to `renameUELs`:

```
In [1]: a.renameUELs({"WI": "WISCONSIN"}, allow_merge=True)
In [2]: a.records
Out[2]:
      uni_0     uni_1  value
0  WISCONSIN      iowa   10.0
1  WISCONSIN  illinois   12.0
In [3]: a.getUELs()
Out[3]: ['WISCONSIN', 'iowa', 'illinois']
```

reorderUELs Examples

`reorderUELs` is a method of all GAMS symbol classes. This method allows the user to reorder UELs of a specific symbol dimension – `reorderUELs` will not all any new UELs to be create nor can they be removed. For example:

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
j = gt.Set(m, "j", i, records=["j1", "j2", "j3"])
a = gt.Parameter(m, "a", [i, j], records=[(f"i{i}", f"j{i}", i) for i in range(1,4)])
In [1]: i.getUELs()
Out[1]: ['i1', 'i2', 'i3']
In [2]: m.getUELs()
Out[2]: ['i1', 'i2', 'i3', 'j1', 'j2', 'j3']
```

But perhaps we want to reorder the UELs `i1`, `i2`, `i3` to `i3`, `i2`, `i1`.

```
In [1]: i.reorderUELs(['i3', 'i2', 'i1'])
In [2]: i.getUELs()
Out[2]: ['i3', 'i2', 'i1']
In [3]: i.records
Out[3]:
  uni   element_text
0    i1
1    i2
2    i3
```

Note

> This example does not change the indexing scheme of the Pandas DataFrame at all, it only changes the underlying integer numbering scheme for the categories. We can see this by looking at the Pandas `codes`:

```
In [1]: i.records["uni"].cat.codes
Out[1]:
0    2
1    1
2    0
dtype: int8
```

setUELs Examples

`reorderUELs` is a method of all GAMS symbol classes. This method allows the user to create new UELs, rename UELs, and reorder UELs all in one method. For example:

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
```

A user could accomplish a UEL reorder operation with `setUELs`:

```
In [1]: i.setUELs(["i3", "i2", "i1"])
In [2]: i.getUELs()
Out[2]: ['i3', 'i2', 'i1']
In [3]: i.records
Out[3]:
  uni   element_text
0    i1
1    i2
2    i3
```

A user could accomplish a UEL reorder + add UELs operation with `setUELs`:

```
In [1]: i.setUELs(["i3", "i2", "i1", "j1", "j2"])
In [2]: i.getUELs()
Out[2]: ['i3', 'i2', 'i1', 'j1', 'j2']
In [3]: i.records
Out[3]:
  uni   element_text
0    i1
1    i2
2    i3
In [4]: i.records["uni"].cat.codes
Out[4]:
0    2
1    1
2    0
dtype: int8
```

A user could accomplish a UEL reorder + add + rename with `setUELs`:

```
In [1]: i.setUELs(["j3", "j2", "j1", "ham", "cheese"], rename=True)
In [2]: i.getUELs()
Out[2]: ['j3', 'j2', 'j1', 'ham', 'cheese']
In [3]: i.records
Out[3]:
  uni   element_text
0    j3
1    j2
2    j1
In [4]: i.records["uni"].cat.codes
Out[4]:
0    0
1    1
2    2
dtype: int8
```

Note

> This example does not change the indexing scheme of the Pandas DataFrame at all, but the `rename=True` flag means that the records will get updated just as if a `renameUELs` call had been made.

If a user wanted to set new UELs on top of this data, without renaming, they would need to be careful to include the current UELs in the UELs being set. It is possible to loose these labels if they are not included (which will prevent the data from being written to GDX/GMD).

```
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2", "i3"])
i.setUELs(["j1", "i2", "j3", "ham", "cheese"])
In [1]: i.getUELs()
Out[1]: ['j1', 'i2', 'j3', 'ham', 'cheese']
In [2]: i.records
Out[2]:
  uni   element_text
0   NaN
1    i2
2   NaN
```

String Manipulation on UELs

It is easy to perform common string manipulations on UELs at the dimension, symbol and container levels with a series of convenience functions: `lowerUELs`, `upperUELs`, `lstripUELs`, `rstripUELs`, `stripUELs`, `capitalizeUELs`, `casefoldUELs`, `titleUELs`, `ljustUELs`, `rjustUELs`. These methods are wrappers around Python's built in string methods and are designed to efficiently perform bulk UEL transformations on your GAMS data.

The following example shows operations on the entire container:

```
m = gt.Container()
i = gt.Set(m, "i", ["*", "*"], records=[(f"i{i}", f"j{i}") for i in range(3)])
k = gt.Set(m, "k", records=[(f"aaa{i}") for i in range(3)])
In [1]: m.getUELs()
Out[1]: ['i0', 'i1', 'i2', 'j0', 'j1', 'j2', 'aaa0', 'aaa1', 'aaa2']
In [2]: m.upperUELs()
Out[2]: <GAMS Transfer Container (0x7f8110719e10)>
In [3]: m.getUELs()
Out[3]: ['I0', 'I1', 'I2', 'J0', 'J1', 'J2', 'AAA0', 'AAA1', 'AAA2']
In [4]: m.lowerUELs("i")
Out[4]: <GAMS Transfer Container (0x7f8110719e10)>
In [5]: m.getUELs()
Out[5]: ['i0', 'i1', 'i2', 'j0', 'j1', 'j2', 'AAA0', 'AAA1', 'AAA2']
In [6]: m.upperUELs().rjustUELs(4, "_")
Out[6]: <GAMS Transfer Container (0x7f8110719e10)>
In [7]: m.getUELs()
Out[7]: ['__I0', '__I1', '__I2', '__J0', '__J1', '__J2', 'AAA0', 'AAA1', 'AAA2']
```

Note

> The `ljustUELs` and `rjustUELs` methods require the user to specify the final string length and the `fill_character` used to pad the string to achieve the final length.

Similar operations can be performed at the dimension and symbol levels as can be seen in the following examples:

```
In [1]: i.upperUELs(0)
Out[1]: <Set 'i' (0x7f8121661930)>
In [2]: i.getUELs()
Out[2]: ['I0', 'I1', 'I2', 'j0', 'j1', 'j2']
In [3]: i.casefoldUELs()
Out[3]: <Set 'i' (0x7f8121661930)>
In [4]: i.getUELs()
Out[4]: ['i0', 'i1', 'i2', 'j0', 'j1', 'j2']
```

Note

> Symbol dimension is indexed from zero (per Python convention)

Reordering Symbols

The order of the Container file requires the symbols to be sorted such that, for example, a Set used as domain of another symbol appears before that symbol. The Container will try to establish a valid ordering when writing the data. This type of situation could be encountered if the user is adding and removing many symbols (and perhaps rewriting symbols with the same name) – users should attempt to only add symbols to a `Container` once, and care must be taken when creating symbol names. The method `reorderSymbols` attempts to fix symbol ordering problems. The following example shows how this can occur:

Example Symbol reordering

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
j = gt.Set(m, "j", i, records=["i" + str(i) for i in range(3)])
In [1]: m.data
Out[1]: {'i': <Set `i` (0x7f7e98907e50)>, 'j': <Set `j` (0x7f7e987fb580)>}
# now we remove the set i and recreate the data
m.removeSymbols("i")
i = gt.Set(m, "i", records=["i" + str(i) for i in range(5)])
```

The symbols are now out of order in `.data` and must be reordered:

```
In [1]: m.data
Out[1]: {'j': <Set `j` (0x7f7e987fb580)>, 'i': <Set `i` (0x7f7e9885a140)>}
# calling reorderSymbols() will order the dictionary properly, but the domain reference in j is now broken
m.reorderSymbols()
# fix the domain reference in the set j
j.domain = i
In [1]: m.isValid()
Out[1]: True
```

Rename Symbols

It is possible to rename a symbol even after it has been added to a `Container`. There are two methods that can be used to achieve the desired outcome:

- using the container method `renameSymbol`
- directly changing the `name` symbol property

We create a `Container` with two sets:

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
j = gt.Set(m, "j", records=["new-york", "chicago", "topeka"])
```

Example #1 - Change the name of a symbol with the container method

```
In [1]: m.renameSymbol("i","h")
In [2]: m.data
Out[2]: {'h': <Set `h` (0x7f7e988582e0)>, 'j': <Set `j` (0x7f7e801240d0)>}
```

Example #2 - Change the name of a symbol with the .name attribute

```
In [1]: i.name = "h"
In [2]: m.data
Out[2]: {'h': <Set `h` (0x7f7e98907520)>, 'j': <Set `j` (0x7f7ea84bb0d0)>}
```

Note

> Note that the renamed symbols maintain the original symbol order, this will prevent unnecessary reordering operations later in the workflow.

Removing Symbols

Removing symbols from a container is easy when using the `removeSymbols` container method; this method accepts either a `str` or a `list` of `str`.

Attention

> Once a symbol has been removed, it is possible to have hanging references as domain links in other symbols. The user will need to repair these other symbols with the proper domain links in order to avoid validity errors.

GAMS Special Values

The GAMS system contains five special values: `UNDEF` (undefined), `NA` (not available), `EPS` (epsilon), `+INF` (positive infinity), `-INF` (negative infinity). These special values must be mapped to their Python equivalents. `transfer` follows the following convention to generate the `1:1` mapping:

- `+INF` is mapped to `float("inf")`
- `-INF` is mapped to `float("-inf")`
- `EPS` is mapped to `-0.0` (mathematically identical to zero)
- `NA` is mapped to a special `NaN`
- `UNDEF` is mapped to `float("nan")`

`transfer` syntax is designed to quickly get data into a form that is usable in further analyses or visualization; this mapping also highlights the preference for data that is of type `float`, which offers performance benefits within Pandas/NumPy. The user does not need to remember these constants as they are provided within the class `SpecialValues` as `SpecialValues.POSINF`, `SpecialValues.NEGINF`, `SpecialValues.EPS`, `SpecialValues.NA`, and `SpecialValues.UNDEF`. The `SpecialValues` class also contains methods to test for these special values. Some examples are shown below; already, we, begin to introduce some of the `transfer` syntax.

Example (special values in a parameter)

```
import gams.transfer as gt
m = gt.Container()
x = gt.Parameter(
    m,
    "x",
    ["*"],
    records=[
        ("i1", 1),
        ("i2", gt.SpecialValues.POSINF),
        ("i3", gt.SpecialValues.NEGINF),
        ("i4", gt.SpecialValues.EPS),
        ("i5", gt.SpecialValues.NA),
        ("i6", gt.SpecialValues.UNDEF),
    ],
    description="special values",
)
```

The following DataFrame for `x` would look like:

```
In [1]: x.records
Out[1]:
  uni    value
0    i1    1.0
1    i2    inf
2    i3   -inf
3    i4   -0.0
4    i5    NaN
5    i6    NaN
```

The user can now easily test for specific special values in the `value` column of the DataFrame (returns a boolean array):

```
In [1]: gt.SpecialValues.isNA(x.records["value"])
Out[1]: array([False, False, False, False,  True, False])
```

Other data structures can be passed into these methods as long as these structures can be converted into a numpy array with `dtype=float`. It follows that:

```
In [1]: gt.SpecialValues.isEps(gt.SpecialValues.EPS)
Out[1]: True
In [2]: gt.SpecialValues.isPosInf(gt.SpecialValues.POSINF)
Out[2]: True
In [3]: gt.SpecialValues.isNegInf(gt.SpecialValues.NEGINF)
Out[3]: True
```

```
In [4]: gt.SpecialValues.isNA(gt.SpecialValues.NA)
Out[4]: True
In [5]: gt.SpecialValues.isUndef(gt.SpecialValues.UNDEF)
Out[5]: True
In [6]: gt.SpecialValues.isUndef(gt.SpecialValues.NA)
Out[6]: False
In [6]: gt.SpecialValues.isNA(gt.SpecialValues.UNDEF)
Out[6]: False
```

Pandas DataFrames allow data columns to exist with mixed type (`dtype=object`) – `transfer` leverages this convenience feature to enable users to import string representations of `EPS`, `NA`, and `UNDEF`. `transfer` is tolerant of any mixed-case special value string representation. Python offers additional flexibility when representing negative/positive infinity. Any string `x` where `float(x) == float("inf")` evaluates to True can be used to represent positive infinity. Similarly, any string `x` where `float(x) == float("-inf")` evaluates to True can be used to represent negative infinity. Allowed values include `inf`, `+inf`, `INFINITY`, `+INFINITY`, `-inf`, `-INFINITY` and all mixed-case equivalents.

Example (special values defined by strings)

```
import gams.transfer as gt
m = gt.Container()
x = gt.Parameter(
    m,
    "x",
    ["*"],
    records=[
        ("i1", 1),
        ("i2", "+inf"),
        ("i3", "-infinity"),
        ("i4", "eps"),
        ("i5", "na"),
        ("i6", "undef"),
    ],
    description="special values",
)
```

These special strings will be immediately mapped to their `float` equivalents from the `SpecialValues` class in order to ensure that all data entries are float types.

Standard Data Formats

This section is meant to introduce the standard format that `transfer` expects for symbol records. It has already been mentioned that we store data as a Pandas DataFrame, but there is an assumed structure to the column headings and column types that will be important to understand. `transfer` includes convenience functions in order to ease the burden of converting data from a user-centric format to one that is understood by `transfer`. However, advanced users will want to convert their data first and add it directly to the Container to avoid making extra copies of (potentially large) data sets.

Set Records Standard Format

All set records (including singleton sets) are stored as a Pandas DataFrame with `n` number of columns, where `n` is the dimensionality of the symbol + 1. The first `n-1` columns include the domain elements while the last column includes the set element explanatory text. Records are organized such that there is one record per row.

The names of the domain columns are flexible, but `transfer` requires unique column names. Users are encouraged to change the column headings of the underlying dataframe by using the `domain_labels` property. Using this property will ensure that unique column names are generated by adding a `_<dimension>` tag to the end of any user supplied column names. The explanatory text column is called `element_text` and must take the last position in the DataFrame.

All domain columns must be a categorical data type and the `element_text` column must be a `object` type. Pandas allows the categories (basically the unique elements of a column) to be various data types as well, however `transfer` requires that all these are type `str`. All rows in the `element_text` column must be type `str`.

Some examples:

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
j = gt.Set(m, "j", [i, "*"], records=[("seattle", "new-york"), ("san-diego", "st-louis")])
k = gt.Set(m, "k", [i], is_singleton=True, records=["seattle"])
In [1]: i.records
Out[1]:
         uni element_text
0    seattle
1  san-diego
In [2]: j.records
Out[2]:
           i       uni element_text
0    seattle  new-york
1  san-diego  st-louis
In [3]: k.records
Out[3]:
         i element_text
0  seattle
```

Parameter Records Standard Format

All parameter records (including scalars) are stored as a Pandas DataFrame with `n` number of columns, where `n` is the dimensionality of the symbol + 1. The first `n-1` columns include the domain elements while the last column includes the numerical value of the records. Records are organized such that there is one record per row. Scalar parameters have zero dimension, therefore they only have one column and one row.

By default, the names of the domain columns follow a pattern of <`set_name`>; a symbol dimension that is referenced to the universe is labeled `uni`. The domain labels can be customized. Users are encouraged to change the column headings of the underlying dataframe by using the `domain_labels` property. Using this property will ensure that unique column names are generated (if not currently unique) by adding a _<`dimension`> tag to the end of any user supplied column names. The value column is called `value` and must take the last position in the DataFrame.

All domain columns must be a categorical data type and the `value` column must be a `float` type. Pandas allows the categories (basically the unique elements of a column) to be various data types as well, however `transfer` requires that all these are type `str`.

Some examples:

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
a = gt.Parameter(m, "a", ["*"], records=[("seattle", 50), ("san-diego", 100)])
b = gt.Parameter(
    m,
    "b",
    [i, "*"],
    records=[("seattle", "new-york", 32.2), ("san-diego", "st-louis", 123)],
)
c = gt.Parameter(m, "c", records=90)
In [1]: a.records
Out[1]:
       uni    value
0    seattle   50.0
1  san-diego  100.0
In [2]: b.records
Out[2]:
         i       uni    value
0    seattle  new-york   32.2
1  san-diego  st-louis  123.0
In [3]: c.records
Out[3]:
   value
0   90.0
```

Variable/Equation Records Standard Format

Variables and equations share the same standard data format. All records (including scalar variables/equations) are stored as a Pandas DataFrame with `n` number of columns, where `n` is the dimensionality of the symbol + 5. The first `n-5` columns include the domain elements while the last five columns include the numerical values for different attributes of the records. Records are organized such that there is one record per row. Scalar variables/equations have zero dimension, therefore they have five columns and one row.

By default, the names of the domain columns follow a pattern of <`set_name`>; a symbol dimension that is referenced to the universe is labeled `uni`. The domain labels can be customized. Users are encouraged to change the column headings of the underlying dataframe by using the `domain_labels` property. Using this property will ensure that unique column names are generated (if not currently unique) by adding a `_`<`dimension`> tag to the end of any user supplied column names. The attribute columns are called `level`, `marginal`, `lower`, `upper`, and `scale`. These attribute columns must appear in this order. Attributes that are not supplied by the user will be assigned the default GAMS values for that variable/equation type; it is possible to not pass any attributes, `transfer` would then simply assign default values to all attributes.

All domain columns must be a categorical data type and all the attribute columns must be a `float` type. Pandas allows the categories (basically the unique elements of a column) to be various data types as well, however `transfer` requires that all these are type `str`.

Some examples:

```
import gams.transfer as gt
import pandas as pd
m = gt.Container()
i = gt.Set(m, "i", records=["seattle", "san-diego"])
a = gt.Variable(
    m,
    "a",
    "free",
    domain=[i],
    records=pd.DataFrame(
        [("seattle", 50), ("san-diego", 100)], columns=["city", "level"]
    ),
)
In [1]: a.records
Out[1]:
         i    level  marginal  lower  upper  scale
0    seattle   50.0       0.0   -inf    inf    1.0
1  san-diego  100.0       0.0   -inf    inf    1.0
```

GDX Read/Write

Up until now, we have been focused on using `transfer` to create symbols in an empty `Container` using the symbol constructors (or their corresponding container methods). These tools will enable users to ingest data from many different formats and add them to a `Container` – however, it is also possible to read in symbol data directly from GDX files using the `read` container method. In the following sections, we will discuss this method in detail as well as the `write` method, which allows users to write out to new GDX files.

Read GDX

There are two main ways to read in GDX based data.

- Pass the file path directly to the Container constructor (will read all symbols and records)
- Pass the file path directly to the `read` method (default read all symbols, but can read partial files)

The first option here is provided for convenience and will, internally, call the `read` method. This method will read in all symbols as well as their records. This is the easiest and fastest way to get data out of a GDX file and into your Python environment. For the following examples we leverage the GDX output generated from the **`trnsport.gms`** model file.

Example (reading full data w/ Container constructor)

```
import gams.transfer as gt
m = gt.Container("trnsport.gdx")
In [1]: m.data
Out[1]:
{'i': <Set `i` (0x7f95b8d63e80)>,
 'j': <Set `j` (0x7f95b8d63a60)>,
 'a': <Parameter `a` (0x7f95b8d63ee0)>,
 'b': <Parameter `b` (0x7f95b8d63d00)>,
 'd': <Parameter `d` (0x7f95b8da86a0)>,
 'f': <Parameter `f` (0x7f95b8da8670)>,
 'c': <Parameter `c` (0x7f95b8da83d0)>,
 'x': <Positive Variable `x` (0x7f95b8da83a0)>,
 'z': <Free Variable `z` (0x7f95b8da8400)>,
 'cost': <Eq Equation `cost` (0x7f95b8da82b0)>,
 'supply': <Leq Equation `supply` (0x7f95b8da8280)>,
 'demand': <Geq Equation `demand` (0x7f95b8da8580)>}
 In [1]: m.describeParameters()
 Out[1]:
  name  domain domain_type  dimension  number_records      min     mean      max            where_min            where_max
      sparsity
0    a     [i]     regular          1              2  350.000  475.000  600.000            [seattle]          [san-diego]
      0.0
1    b     [j]     regular          1              3  275.000  300.000  325.000             [topeka]            [new-york]
      0.0
2    c  [i, j]     regular          2              6    0.126    0.176    0.225  [san-diego, topeka]  [seattle, new-york]
      0.0
3    d  [i, j]     regular          2              6    1.400    1.950    2.500  [san-diego, topeka]  [seattle, new-york]
      0.0
4    f      []        none          0              1   90.000   90.000   90.000                 None                 None
      None
```

A user could also read in data with the `read` method as shown in the following example.

Example (reading full data w/ `read` method)

```
import gams.transfer as gt
m = gt.Container()
m.read("trnsport.gdx")
In [1]: m.data
Out[1]:
{'i': <Set `i` (0x7f95b8d63e80)>,
 'j': <Set `j` (0x7f95b8d63a60)>,
 'a': <Parameter `a` (0x7f95b8d63ee0)>,
 'b': <Parameter `b` (0x7f95b8d63d00)>,
 'd': <Parameter `d` (0x7f95b8da86a0)>,
 'f': <Parameter `f` (0x7f95b8da8670)>,
 'c': <Parameter `c` (0x7f95b8da83d0)>,
 'x': <Positive Variable `x` (0x7f95b8da83a0)>,
 'z': <Free Variable `z` (0x7f95b8da8400)>,
 'cost': <Eq Equation `cost` (0x7f95b8da82b0)>,
 'supply': <Leq Equation `supply` (0x7f95b8da8280)>,
 'demand': <Geq Equation `demand` (0x7f95b8da8580)>}
```

It is also possible to read in a partial GDX file with the `read` method, as shown in the following example:

```
m = gt.Container()
m.read("trnsport.gdx", "x")
In [1]: m.data
Out[1]: {'x': <Positive Variable `x` (0x7f9598a38dc0)>}
In [2]: m.data["x"].records
Out[2]:
         i         j    level  marginal  lower  upper  scale
0    seattle  new-york   50.0     0.000    0.0    inf    1.0
1    seattle   chicago  300.0     0.000    0.0    inf    1.0
2    seattle    topeka    0.0     0.036    0.0    inf    1.0
3  san-diego  new-york  275.0     0.000    0.0    inf    1.0
4  san-diego   chicago    0.0     0.009    0.0    inf    1.0
5  san-diego    topeka  275.0     0.000    0.0    inf    1.0
```

This syntax assumes that the user will always want to read in both the metadata as well as the actual data records, but it is possible to skip the reading of the records by passing the argument `records=False`.

```
m = gt.Container()
m.read("trnsport.gdx", "x", records=False)
In [1]: m.data
Out[1]: {'x': <Positive Variable `x` (0x7f9598a3a200)>}
In [2]: m["x"].summary
```

```
Out[2]:
{'name': 'x',
 'description': 'shipment quantities in cases',
 'type': 'positive',
 'domain': ['i', 'j'],
 'domain_type': 'regular',
 'dimension': 2,
 'number_records': 6}
In [3]: type(m["x"].records)
Out[3]: <class 'NoneType'>
```

Attention

> The `read` method attempts to link the domain objects together (in order to have a "regular" `domain_type`) but if domain sets are not part of the read operation there is no choice but to default to a "relaxed" `domain_type`. This can be seen in the last example where we only read in the variable `x` and not the domain sets (`i` and `j`) that the variable is defined over. All the data will be available to the user, but domain checking is no longer possible. The symbol `x` will remain with "relaxed" domain type even if the user were to read in sets `i` and `j` in a second `read` call.

Write GDX

A user can write data to a GDX file by simply passing a file path (as a string). The `write` method will then create the GDX and write all data in the `Container`.

Example

```
m.write("path/to/file.gdx")
```

Example (write a compressed GDX file)

```
m.write("path/to/file.gdx", compress=True)
```

By default, all symbols in the Container will be written, however it is possible to write a subset of the symbols to a GDX file with the `symbols` argument. If a domain set is not included in the `symbols` list then the symbol will automatically be relaxed (but will retain the domain set's name as a string label – it does not get relaxed to $*$). This behavior can be seen in the following example.

```
import gams.transfer as gt
m = gt.Container()
i = gt.Set(m, "i", records=["i1", "i2"])
a = gt.Parameter(
    m,
    "a",
    [i, i],
    records=[("i1", "i1", 10), ("i2", "i2", 12)],
)
m.write("out.gdx", "a")
# create a new container and read in the GDX
m2 = gt.Container("out.gdx")
# look at all the data
In [1]: m2.data
Out[1]: {'a': <Parameter `a` (0x7f9598a61510)>}
# notice that `a` has a relaxed domain type now
In [2]: m2["a"].domain_type
Out[2]: 'relaxed'
# `a` retains the labels from the original domain sets
In [3]: m2["a"].domain
Out[3]: ['i', 'i']
# The original container `m` retains its original state before writing
In [4]: m["a"].domain
Out[4]: [<Set `i` (0x7f9598a39a80)>, <Set `i` (0x7f9598a39a80)>]
```

In line 4 we can see that the auto-relaxation of the domain for `a` is only temporary for writing (in this case, from Container object `m`) and will be restored so as not to disturb the Container state.

Advanced users might want to specify an order to their UEL list (i.e., the universe set); recall that the UEL ordering follows that dictated by the data. As a convenience, it is possible to prepend the UEL list with a user specified order using the `uel_priority` argument.

Example (change the order of the UEL)

```
m = gt.Container()
i = gt.Set(m, "i", records=["a", "b", "c"])
m.write("foo.gdx", uel_priority=["a", "c"])
```

The original UEL order for this GDX file would have been `["a", "b", "c"]`, but this example reorders the UEL with `uel_priority` – the positions of `b` and `c` have been swapped. This can be verified with the `gdxdump` utility (using the `uelTable` argument):

```
gdxdump foo.gdx ueltable=foo

Set foo /
  'a' ,
  'c' ,
  'b' /;
$onEmpty

Set i(*) /
'a',
'c',
'b' /;

$offEmpty
```

GamsDatabase Read/Write

We have discussed how to create symbols in an empty `Container` and we have discussed how to exchange data with GDX files, however it is also possible to read and write data directly in memory by interacting with a GamsDatabase/GMD object – this allows `transfer` to be used to read/write data within an Embedded Python Code environment or in combination with the Python OO API. There are some important differences when compared to data exchange with GDX since we are working with data representations in memory.

Read GamsDatabases

Just as with a GDX, there are two main ways to read in data that is in a GamsDatabase/GMD object.

- Pass the GamsDatabase/GMD object directly to the Container constructor (will read all symbols and records)
- Pass the GamsDatabase/GMD object directly to the `read` method (default read all symbols, but can read partial files)

The first option here is provided for convenience and will, internally, call the `read` method. This method will read in all symbols as well as their records. This is the easiest and fastest way to get data out of a GamsDatabase/GMD object and into your Python environment. While it is possible to generate a custom GamsDatabase/GMD object from scratch (using the `gmdcc` API), most users will be interacting with a GamsDatabase/GMD object that has already been instantiated internally when he/she is using Embedded Python Code or the GamsDatabase class in the Python OO API. Our examples will show how to access the GamsDatabase/GMD object – we leverage the some of the data from the **`trnsport.gms`** model file.

Example (reading full data w/ Container constructor)

```
m = gt.Container(gams.db)
```

Note

Embedded Python Code users will want pass the GamsDatabase object that is part of the GAMS Database object – this will always be referenced as `gams.db` regardless of the model file.

The following example uses embedded Python code to create a new Container, read in all symbols, and display some summary statistics as part of the gams log output.

```
Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

Parameter
   a(i) 'capacity of plant i in cases'
        / seattle    350
          san-diego  600 /

   b(j) 'demand at market j in cases'
        / new-york   325
          chicago    300
          topeka     275 /;

Table d(i,j) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;

$onembeddedCode Python:
import gams.transfer as gt

m = gt.Container(gams.db)
print(m.describeSets())

print(m.describeParameters())

$offEmbeddedCode
```

The gams log output will then look as such (the extra `print` calls are just providing nice spacing for this example):

```
GAMS 43.1.0   Copyright (C) 1987-2023 GAMS Development. All rights reserved
--- Starting compilation
--- matrix.gms(29) 3 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
  name  is_singleton domain domain_type  dimension  number_records sparsity
0    i         False    [*]        none          1               2     None
1    j         False    [*]        none          1               3     None
  name  domain domain_type  dimension  number_records      min     mean      max           where_mi
0    a     [i]     regular          1               2  350.000  475.000  600.000           [seattle
```

```
1    b     [j]     regular          1               3  275.000  300.000  325.000             [topeka]
2    d  [i, j]     regular          2               6    1.400    1.950    2.500  [san-diego, topeka]

--- Starting execution - empty program
*** Status: Normal completion

[3 rows x 16 columns]

--- Starting execution - empty program
*** Status: Normal completion
```

A user could also read in a subset of the data located in the GamsDatabase object with the `read` method as shown in the following example. Here we only read in the sets `i` and `j`, as a result the `.describeParameters()` method will return `None`.

Example (reading subset of full data w/ `read` method)

```
Set
   i 'canning plants' / seattle,  san-diego /
   j 'markets'        / new-york, chicago, topeka /;

Parameter
   a(i) 'capacity of plant i in cases'
        / seattle    350
          san-diego  600 /

   b(j) 'demand at market j in cases'
        / new-york   325
          chicago    300
          topeka     275 /;

Table d(i,j) 'distance in thousands of miles'
              new-york  chicago  topeka
   seattle         2.5      1.7     1.8
   san-diego       2.5      1.8     1.4;

$onembeddedCode Python:
import gams.transfer as gt

m = gt.Container()
m.read(gams.db, symbols=["i","j"])
gams.printLog("")
print(m.describeSets())
print(m.describeParameters())

$offEmbeddedCode
```

```
GAMS 43.1.0   Copyright (C) 1987-2023 GAMS Development. All rights reserved
--- Starting compilation
--- matrix.gms(29) 3 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
--- name  is_singleton domain domain_type  dimension  number_records sparsity
0    i         False    [*]        none          1               2     None
```

```
1    j         False    [*]        none          1              3     None
None

--- Starting execution - empty program
*** Status: Normal completion
```

All the typical functionality of the Container exists when working with GamsDatabase/GMD objects. This means that domain linking, matrix conversion, and other more advanced options are available to the user at either compilation time or execution time (depending on the Embedded Code syntax being used, see: Syntax). The next example generates a 1000x1000 matrix and then takes its inverse using the Numpy `linalg` package.

Example (Matrix Generation and Inversion)

```
set i / i1*i1000 /;
alias(i,j);

parameter a(i,j);
a(i,j) = 1 / (ord(i)+ord(j) - 1);
a(i,i) = 1;


embeddedCode Python:
import gams.transfer as gt
import numpy as np
import time

gams.printLog("")
s = time.time()
m = gt.Container(gams.db)
gams.printLog(f"read data: {round(time.time() - s, 3)} sec")

s = time.time()
A = m["a"].toDense()
gams.printLog(f"create matrix A: {round(time.time() - s, 3)} sec")

s = time.time()
invA = np.linalg.inv(A)
gams.printLog(f"generate inv(A): {round(time.time() - s, 3)} sec")

endEmbeddedCode
```

Note

> In this example, the assignment of the `a` parameter is done during execution time so we must use the execution time syntax for embedded code in order to get the numerical records properly.

```
GAMS 43.1.0   Copyright (C) 1987-2023 GAMS Development. All rights reserved
--- Starting compilation
--- test.gms(27) 3 Mb
--- Starting execution: elapsed 0:00:00.003
--- test.gms(9) 36 Mb
--- Initialize embedded library libembpycclib64.dylib
```

```
--- Execute embedded library libembpycclib64.dylib
---
--- read data: 1.1 sec
--- create matrix A: 0.02 sec
--- generate inv(A): 0.031 sec
*** Status: Normal completion
```

We will extend this example in the next section to write the inverse matrix `A` back into a GAMS parameter.

Write to GamsDatabases

A user can write to a GamsDatabase/GMD object with the `.write()` method just as he/she would write a GDX file – however there are some important differences. When a user writes a GDX file the entire GDX file represents a complete data environment (all domains have been resolved, etc.) thus, `transfer` does not need to worry about merge/replace operations. It is possible to merge/replace symbol records when a user is writing data to in-memory data representations with GamsDatabase/GMD. We show a few examples to illustrate this behavior.

Example (Populating a set in GAMS)

```
* note that we need to declare the set i over "*" in order to provide hints about the symbol dimensi
set i(*);

$onembeddedCode Python:
import gams.transfer as gt

m = gt.Container()
i = gt.Set(m, "i", records=["i"+str(i) for i in range(10)])
m.write(gams.db)

$offEmbeddedCode i


embeddedCode Python:
import gams.transfer as gt

m = gt.Container(gams.db)
gams.printLog("")
print(m["i"].records)

endEmbeddedCode
```

Note

> In general, it is possible to use `transfer` to create new symbols in a GamsDatabase and GMD object (and not necessarily merge symbols) but embedded code best practices necessitate the declaration of any GAMS symbols on the GAMS side first, then the records can be filled with `transfer`.

If we break down this example we can see that the set `i` is declared within GAMS (with no records) and then the records for `i` are set by writing a `Container` to the `gams.db` GamsDatabase object (we do this at compile time). The second embedded Python code block runs at execution time and is simply there to read all the records on the set `i` – printing the sets this way adds the output to the `.log` file (we could also use the more common `display i;` operation in GAMS to display the set elements in the LST file).

```
GAMS 43.1.0   Copyright (C) 1987-2023 GAMS Development. All rights reserved
--- Starting compilation
--- test.gms(10) 2 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
--- test.gms(20) 3 Mb
--- Starting execution: elapsed 0:00:01.464
--- test.gms(13) 4 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
---   uni   element_text
0    i0
1    i1
2    i2
3    i3
4    i4
5    i5
6    i6
7    i7
8    i8
9    i9

*** Status: Normal completion
```

Example (Merging set records)

```
set i / i1, i2 /;

$onmulti
$onembeddedCode Python:
import gams.transfer as gt

m = gt.Container()
i = gt.Set(m, "i", records=["i"+str(i) for i in range(10)])
m.write(gams.db, merge_symbols="i")

$offEmbeddedCode i
$offmulti

embeddedCode Python:
import gams.transfer as gt

m = gt.Container(gams.db)
gams.printLog("")
print(m["i"].records)

endEmbeddedCode
```

In this example we need to make use of $onMulti/$offMulti in order to merge new set elements into the the set `i` (the same would be true if we were merging other symbol types) – any symbol that already has records defined (in GAMS) and is being added to with Python (and `transfer`) must be wrapped with $onMulti/$offMulti. As with the previous example, the second embedded Python code block runs at execution time and is simply there to read all the records on the set `i`. Note that the UEL order will be different in this case (`i1` and `i2` come before `i0`).

```
GAMS 43.1.0   Copyright (C) 1987-2023 GAMS Development. All rights reserved
--- Starting compilation
--- test.gms(11) 3 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
--- test.gms(21) 3 Mb
--- Starting execution: elapsed 0:00:01.535
--- test.gms(14) 4 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
---   uni   element_text
0    i1
1    i2
2    i0
3    i3
4    i4
5    i5
6    i6
7    i7
8    i8
9    i9

*** Status: Normal completion
```

Example (Replacing set records)

```
set i / x1, x2 /;

$onmultiR
$onembeddedCode Python:
import gams.transfer as gt

m = gt.Container()
i = gt.Set(m, "i", records=["i"+str(i) for i in range(10)])
m.write(gams.db)

$offEmbeddedCode i
$offmulti

embeddedCode Python:
import gams.transfer as gt

m = gt.Container(gams.db)
gams.printLog("")
print(m["i"].records)

endEmbeddedCode
```

In this example we want to replace the `x1` and `x2` set elements and built up a totally new element list with set elements from the `Container`. Instead of `$onMulti`/`$offMulti` we must use `$onMultiR`/`$offMulti` to ensure that the replacement happens in GAMS; we also need to remove the set `i` from the `merge_symbols` argument.

Attention

If the user seeks to replace all records in a symbol they must use the `$onMultiR` syntax. It is not sufficient to simply remove them from the `merge_symbols` argument in `transfer`. If the user mistakenly uses `$onMulti` the symbols will end up merging without total replacement.

```
GAMS 43.1.0   Copyright (C) 1987-2023 GAMS Development. All rights reserved
--- Starting compilation
--- test.gms(11) 3 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
--- test.gms(21) 3 Mb
--- Starting execution: elapsed 0:00:01.482
--- test.gms(14) 4 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
---    uni   element_text
0     i0
1     i1
2     i2
3     i3
4     i4
5     i5
6     i6
7     i7
8     i8
9     i9

*** Status: Normal completion
```

Example (Merging parameter records)

```
set i;
parameter a(i<) /
i1 1.23
i2 5
/;

$onmulti
$onembeddedCode Python:
import gams.transfer as gt

m = gt.Container()
i = gt.Set(m, "i", records=["i"+str(i) for i in range(10)])
a = gt.Parameter(m, "a", domain=i, records=[("i"+str(i),i) for i in range(10)])
m.write(gams.db, merge_symbols="a")

$offEmbeddedCode i, a
$offmulti

embeddedCode Python:
import gams.transfer as gt

m = gt.Container(gams.db)
gams.printLog("")
print(m["a"].records)
endEmbeddedCode
```

In this example we also need to make use of `$onMulti/$offMulti` in order to merge new set elements into the the set `i`, however the set `i` also needs to contain the elements that are defined in the parameter – here we make use of the < operator that will add the set elements from `a(i)` into the set `i`

Note

It would also be possible to run this example by explicitly defining the `set i /i1, i2/;` before the parameter declaration.

Attention

`transfer` will overwrite all duplicate records when merging. The original values of `a("i1")` and `a("i2")` have been replaced with their new values when writing the Container in this example (see output below).

```
GAMS 43.1.0   Copyright (C) 1987-2023 GAMS Development. All rights reserved
--- Starting compilation
--- test.gms(16) 3 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
--- test.gms(25) 3 Mb
--- Starting execution: elapsed 0:00:01.467
--- test.gms(19) 4 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
---   i    value
0  i1    1.0
1  i2    2.0
2  i3    3.0
3  i4    4.0
4  i5    5.0
5  i6    6.0
6  i7    7.0
7  i8    8.0
8  i9    9.0

*** Status: Normal completion
```

Example (Advanced Matrix Generation and Inversion w/ Write Operation)

```
set i / i1*i1000 /;
alias(i,j);

parameter a(i,j);
a(i,j) = 1 / (ord(i)+ord(j) - 1);
a(i,i) = 1;

parameter inv_a(i,j);
parameter ident(i,j);

embeddedCode Python:
import gams.transfer as gt
import numpy as np
```

```
import time

gams.printLog("")
gams.printLog("")

s = time.time()
m = gt.Container(gams.db)
gams.printLog(f"read data: {round(time.time() - s, 3)} sec")

s = time.time()
A = m["a"].toDense()
gams.printLog(f"create matrix A: {round(time.time() - s, 3)} sec")

s = time.time()
invA = np.linalg.inv(A)
gams.printLog(f"calculate inv(A): {round(time.time() - s, 3)} sec")

s = time.time()
m["inv_a"].setRecords(invA)
gams.printLog(f"convert matrix to records for inv(A): {round(time.time() - s, 3)} sec")

s = time.time()
I = np.dot(A,invA)
tol = 1e-9
I[np.where((I<tol) & (I>-tol))] = 0
gams.printLog(f"calculate A*invA + small number cleanup: {round(time.time() - s, 3)} sec")

s = time.time()
m["ident"].setRecords(I)
gams.printLog(f"convert matrix to records for I: {round(time.time() - s, 3)} sec")

s = time.time()
m.write(gams.db, ["inv_a","ident"])
gams.printLog(f"write to GamsDatabase: {round(time.time() - s, 3)} sec")

gams.printLog("")
endEmbeddedCode inv_a, ident

display ident;
```

In this example we extend the example shown in Read GamsDatabases to read data from GAMS, calculate a matrix inversion, do the matrix multiplication, and then write both the `A^-1` and `A*A^-1` (i.e., the identity matrix) back to GAMS for display in the LST file. This data round trip highlights the benefits of using a `transfer` Container (and the linked symbol structure) as the mechanism to move data – converting back and forth from a records format to a matrix format can be cumbersome, but here, `transfer` takes care of all the indexing for the user.

The first few lines of GAMS code generates a 1000x1000 `A` matrix as a parameter (at execution time), we then define two more parameters that we will fill with results of the embedded Python code – specifically we want to fill a parameter with the matrix `A^-1` and we want to verify that another parameter (`ident`) contains the identity matrix (i.e., `I`). Stepping through the code:

1. We start the embedded Python code section (execution time) by importing both `transfer` and Numpy and by reading all the symbols that currently exist in the GamsDatabase. We must read in all this information in order to get the domain set information – `transfer` needs these domain sets in order to generate matricies with the proper size.
2. Generate the matrix `A` by calling `.toDense()` on the symbol object in the Container.

3. Take the inverse of `A` with `np.linalg.inv()`.
4. The Parameter symbol for `inv_a` already exists in the Container, but it does not have any records (i.e., `m["inv_a"].records is None` will evaluate to True). We use `.setRecords()` to convert the `invA` back into a records format.
5. We continue the computations by performing the matrix multiplication using `np.dot()` – we must clean up a lot of small numbers in `I`.
6. The Parameter symbol for `ident` already exists in the Container, but it does not have any records. We use `.setRecords()` to convert `I` back into a records format.
7. Since we are calculating these parameter values at execution time, it is not possible to modify the domain set information (or even merge/replace it). Therefore we only want to write the parameter values to GAMS. We achieve this by writing a subset of the Container symbols out with the `m.write(gams.db, ["inv_a","ident"])` call. This partial write preserves symbol validity in the Container and it does not violate other GAMS requirements.
8. Finally, we can verify that the (albeit large) identity matrix exists in the LST file (or in another GDX file).

Note

It was not possible to just use `np.round` because small negative numbers that round to `-0.0` will be interpreted by `transfer` as the GAMS EPS special value.

The output for this example is shown below:

```
GAMS 43.1.0   Copyright (C) 1987-2023 GAMS Development. All rights reserved
--- Starting compilation
--- matrix.gms(52) 3 Mb
--- Starting execution: elapsed 0:00:00.004
--- matrix.gms(11) 36 Mb
--- Initialize embedded library libembpycclib64.dylib
--- Execute embedded library libembpycclib64.dylib
---
---
--- read data: 1.083 sec
--- create matrix A: 0.016 sec
--- calculate inv(A): 0.032 sec
--- convert matrix to records for inv(A): 0.176 sec
--- calculate A*invA + small number cleanup: 0.027 sec
--- convert matrix to records for I: 0.17 sec
--- write to GamsDatabase: 1.937 sec
---
--- matrix.gms(52) 68 Mb
*** Status: Normal completion
```

Container Read

Containers can read from other `Container` instances. The syntax and behavior is much the same as reading from GDX and GMD sources. It is important to note that a deepcopy of all data is made when reading from these sources. The container object can be passed into the constructor (to be consistent with the shorthand notation) or the object can be passed as a argument to the `.read()` method.

```
import gams.transfer as gt
m = gt.Container("trnsport.gdx")
In [1]: m.data
Out[1]:
```

```
{'i': <Set `i` (0x7fc1d86d8e80)>,
 'j': <Set `j` (0x7fc1d86d8e50)>,
 'a': <Parameter `a` (0x7fc1d86d8df0)>,
 'b': <Parameter `b` (0x7fc1d86d8fa0)>,
 'd': <Parameter `d` (0x7fc1d86d84c0)>,
 'f': <Parameter `f` (0x7fc1d86d9000)>,
 'c': <Parameter `c` (0x7fc1d86d9120)>,
 'x': <Positive Variable `x` (0x7fc1d86d90f0)>,
 'z': <Free Variable `z` (0x7fc1d86d8fd0)>,
 'cost': <Eq Equation `cost` (0x7fc1d86d8cd0)>,
 'supply': <Leq Equation `supply` (0x7fc1d86d8c40)>,
 'demand': <Geq Equation `demand` (0x7fc1d86d8c10)>}
m2 = gt.Container()
m2.read(m)
# equivalent to m2 = gt.Container(m)
In [7]: m2.data
Out[7]:
{'i': <Set `i` (0x7fc1c8153fa0)>,
 'j': <Set `j` (0x7fc1c8153730)>,
 'a': <Parameter `a` (0x7fc1c8153cd0)>,
 'b': <Parameter `b` (0x7fc1d86fc790)>,
 'd': <Parameter `d` (0x7fc1f87dd240)>,
 'f': <Parameter `f` (0x7fc1c8153eb0)>,
 'c': <Parameter `c` (0x7fc1f87ddb40)>,
 'x': <Positive Variable `x` (0x7fc1c81536a0)>,
 'z': <Free Variable `z` (0x7fc1f87ddc60)>,
 'cost': <Eq Equation `cost` (0x7fc1c81539a0)>,
 'supply': <Leq Equation `supply` (0x7fc1f87dcd00)>,
 'demand': <Geq Equation `demand` (0x7fc1d86ff700)>}
```

Combining two containers

In this example we create two containers (which could have been populated from GDX files) and add in all symbol that do not currently exist in the first `Container`

```
import gams.transfer as gt
m1 = gt.Container()
i = gt.Set(m1, "i", records=[f"i{i}" for i in range(10)])
j = gt.Set(m1, "j", records=[f"j{i}" for i in range(10)])
k = gt.Set(m1, "k", records=[f"k{i}" for i in range(10)])
m2 = gt.Container()
a = gt.Set(m2, "a", records=[f"a{i}" for i in range(10)])
b = gt.Set(m2, "b", records=[f"b{i}" for i in range(10)])
k = gt.Set(m2, "k", records=[f"k{i}" for i in range(10)])
# now read in everything from m2 that does not exist in m1 (will read `a` and `b`)
m1.read(m2, [symname for symname, obj in m2 if symname not in m1])
In [1]: m1.data
Out[1]:
{'i': <Set `i` (0x7f9e504231c0)>,
 'j': <Set `j` (0x7f9e3043cc10)>,
 'k': <Set `k` (0x7f9e509bcd00)>,
 'a': <Set `a` (0x7f9e50423760)>,
 'b': <Set `b` (0x7f9e508ed0c0)>}
In [2]: m1.isValid()
Out[2]: True
```

7.5.5 Getting Started

Users have many options to manage their Python installation; in many cases it is advantageous to create a Python "environment" to sandbox an instance of Python (something which can be done with `venv` or `conda`). We recommend that users download a version of `miniconda`. We direct users to the `conda` documentation for installation issues. After the user has installed `miniconda` (or `conda`) we will:

- Verify that `conda` is working
- Create a new scratch Python environment
- Enter the new environment and verify the `python` version
- Install the GAMS API with `pip`

Note

`miniconda` and `conda` are synonymous – both are package + environment managers – however, `conda` comes preloaded with a number of useful data science-related packages. We emphasize `miniconda` because of the smaller install size. The terminal commands we used here apply to both `miniconda` and `conda` versions. The remaining documentation will only use the term `conda`.

Attention

The new API structure cannot be used to simply "update" previous versions – users should build new Python environments from scratch before attempting to install.

7.5.5.1 Verify Conda Installation

To verify that `conda` is accessible from the terminal we simply need to check for the version number. Your version of `conda` may differ. The installation of the GAMS API does not depend on the `conda` version.

7.5.5.2 Create a New Conda Environment

`Conda` can create, manage, and delete Python environments easily – this flexibility allows users to experiment quickly with different tools. If experiments fail, the entire environment can be removed without damaging the rest of the system. In short, each `conda` environment is an isolated sandbox. We will now create a new `conda` environment called `gams` that we will used when installing the GAMS API.

Note

Best practice is to use an environment rather than install into conda's base environment

Note

It is necessary to specify the version of Python to install into the new `conda` environment. The GAMS API currently supports Python 3.8 to 3.12.

We must now "activate" the `conda` environment (i.e., enter the Python sandbox).

Once in the activated environment we should verify the Python version is the one that was specified at creation.

7.5.5.3 Install

The GAMS Python API is distributed through the Python Package Index. The `gamsapi` package comes with several install options (via `pip extras`) that change how dependencies are resolved. `pip` will not install any third-party dependencies if an `extra` label was not provided. For example, to install the `transfer` data tool (and its dependencies to `pandas` and `scipy`).

Note

`xx.y.z` represents your installed GAMS version number (e.g., 46.4.1)

The `extras` that are available for the GAMS Python API are:

extra	Third-party Dependencies to Install
connect	pandas, pyyaml, openpyxl, sqlalchemy, cerberus, pyodbc, psycopg2-binary, pymysql, pymssql
control	urllib3
core	ply
engine	python_dateutil, urllib3
magic	ipython, pandas
tools	pandas
transfer	pandas, scipy
all	installs all third-party dependencies for all sub-modules – a complete install

Attention

Users can chain several `extras` together (separated by commas) in one `pip` install command to install dependencies from several sub-modules at once.

7.5.5.4 macOS Compatibility

The default shell for macOS 10.15+ (Catalina) is `zsh`. Users that wish to install the `gamsapi` will need to modify their install syntax slightly (note the quotations).

Note

`xx.y.z` represents your installed GAMS version number (e.g., 46.4.1)

Apple M1/M2

Apple users that have a M1/M2 (`arm`) chipset must be careful to match build architectures (i.e., `x86` or `arm`) of both the GAMS system and miniconda (Python). Ideally, M1/M2 users will only install native `arm` compatible programs. Apple's Rosetta 2 does allow users to install and run `x86` compiled programs on M1/M2. However, mixed installations (i.e., an `arm` GAMS system but a `x86` miniconda or vice versa) will fail because the `gamsapi` will not be able to properly load the necessary shared libraries.

The `gams audit` tool will return the GAMS build architecture (`x86` or `arm` will be in the returned string):

The build architecture of the Python installation is available with the following command:

Attention

The user must reinstall either a GAMS or Python system if these two returns do not match.

7.5.5.5 Verify the GAMS API

`pip` provides feedback that suggests that the GAMS API was successfully installed, however, it is still wise to verify this. The best way to test is to actually create a short Python script that imports `gams`. The following 1 line will run an `import` operation and, if successful, will output the API version number. The API was not successfully installed if an import error is raised.

Attention

Example problems can be found in the `[PATH TO GAMS]/api/python/examples` folder (organized by sub-module).

7.5.5.6 Uninstall the GAMS API

Removal of the GAMS API is straightforward with `pip`:

7.5.5.7 Remove Conda Environment

Removal of the entire `conda` environment is also straightforward with the following operations:

7.5.5.8 Other Useful Conda Commands

Users are directed to the full `conda` documentation but some useful commands are provided here as a quick reference.

Conda Command	Description
`conda env list`	List all conda environments
`conda list`	List all installed packages in the active environment
`conda deactivate`	Deactivate the current Python environment
`conda remove --name XXX --all`	Remove the `XXX` environment, must be deactivated first
`conda install XXX`	Install package `XXX` with the conda system, not all packages can be installed from `conda` directly

7.5.5.9 Python Virtual Environment (venv)

This documentation assumed users will want to use `conda` to manage their Python environments, but other tools such as `venv` can be used to manage separate Python environments. The details of the `venv` setup, activation, deactivation and removal differ from `conda`, but the `pip` install commands are the same as in `conda`. Interested users are referred to the official `venv` documentation for details on how to create a virtual environment. Users are also directed to additional documentation on: `Installing packages using pip and virtual environments`.

7.5.5.10 Working with Python and multiple GAMS Installations

Some users may want to run multiple versions of GAMS on their system – we recommend that users create separate Python environments in order to compare the behavior between versions of the Python API.

7.5.5.11 A note on using Python site package

Previous API versions could be used by a Python interpreter if the path to the API directory was included in a `sitecustomize.py` script that resided in the `site-packages` directory. This type of installation allows customized packages to be found and used, but not necessarily copied into Python's directory structure. Previous versions of the API benefited from this type of installation, we recommend that users create a separate Python environment and actually install the GAMS API into the environment with `pip`. Users that were using the `sitecustomize.py` installation method might experience issues with `pip` installations if their Python finds an old `sitecustomize.py` file that includes a path to old GAMS API files (`pip` might report that the `requirement is already satisfied`). Users can find out where the `USER_SITE` directory is located by running the following command:

Once this directory has been found, it is necessary to remove all paths (in the `sitecustomize.py` file) that point to previous GAMS API folders and reattempt the `pip` install process.

7.6 Matlab API

The GAMS Matlab API is a Matlab package that contains several sub-packages that enable the control of the GAMS system as well as the movement of data between GAMS and Matlab. The following table gives an overview of all available sub-packages:

Sub-Package	Description
gams.control	Enables full control of the GAMS System
gams.transfer	Data Only API – Allows GAMS data to be maintained outside a GAMS script

Attention

Don't use the data exchange tool GDXMRW anymore and switch to **gams.transfer** instead.

7.6.1 Control

The GAMS Matlab Control API provides a Matlab programming interface to the General Algebraic Model System (GAMS). GAMS Matlab Control API objects allow a convenient way to exchange input data and model results with in-memory representation of data (**Database**), and to create and run GAMS models (**Job**) that can be customized by GAMS options (**Options**). Furthermore, they introduce a way to solve a sequence of closely related model instances in a more efficient way (**ModelInstance**).

Note

If you only want to exchange data between Matlab and GAMS, use **GAMS Transfer Matlab** as it is very likely to be more efficient than GAMS Matlab Control API in case of exchanging very large data sets.

The underlying GAMS engine relies to some extent on file based communication (e.g. the listing file) and other unmanaged resources. Since the GAMS Matlab Control API is based on the GAMS Java API, the use of external resources in the Java environment requires special attention. Hence, some objects need to be properly disposed before the Java garbage collector does its job.

A GAMS program can include other source files (e.g. $include), load data from GDX files (e.g. $GDXIN or execute_load), and create PUT files. All these files can be specified with a (relative) path and therefore an anchor into the file system is required. The base object **Workspace** manages the anchor to the file system.

With the exception of **Workspace** the objects in the gams package cannot be accessed across different threads unless the instance is locked. The classes themselves are thread safe and multiple objects of the class can be used from different threads (see below for restrictions on solvers that are not thread safe within the **ModelInstance** class).

Note

If you use multiple instances of the **Workspace** in parallel, you should avoid using the same working directory. Otherwise you may end up with conflicting file names.

Currently only `Cplex`, `Gurobi`, and `SoPlex` fully utilize the power of solving **ModelInstances**. Some solvers will not work in a multi-threaded application using **ModelInstances**. For some solvers this is unavoidable because the solver library is not thread safe (e.g. `MINOS`). Moreover, **ModelInstances** are not available for quadratic model types (QCP, MIQCP, RMIQCP).

This version of the GAMS Matlab Control API also does not provide support for the following GAMS components: acronyms, GAMS compilation/execution errors, structured access to listing file, and solver options.

To get started, see Getting Started.

7.6.1.1 New Class Names

With GAMS 46 the GAMS Matlab Control API has been restructured and classes have been renamed. The following lists a mapping of old to new class names:

- `GAMS.AbstractRunParameters: gams.control.AbstractRunParameters`
- `GAMS.GAMSCheckpoint: gams.control.Checkpoint`
- `GAMS.GAMSDatabase: gams.control.Database`
- `GAMS.GAMSDatabaseDomainViolation: gams.control.DatabaseDomainViolation`
- `GAMS.GAMSEngineConfiguration: gams.control.engine.Configuration`
- `GAMS.GAMSEngineJob: gams.control.engine.Job`
- `GAMS.GAMSEngineJobBuilder: gams.control.engine.JobBuilder`
- `GAMS.GAMSEngineRunParameters: gams.control.engine.RunParameters`
- `GAMS.GAMSEquation: gams.control.Equation`
- `GAMS.GAMSEquationRecord: gams.control.EquationRecord`
- `GAMS.GAMSGlobals: gams.control.Globals`
- `GAMS.DebugLevel: gams.control.globals.DebugLevel`
- `GAMS.UpdateAction: gams.control.globals.UpdateAction`
- `GAMS.ModelStat: gams.control.globals.ModelStat`
- `GAMS.SolveStat: gams.control.globals.SolveStat`
- `GAMS.SetType: gams.control.globals.SetType`
- `GAMS.VarType: gams.control.globals.VarType`
- `GAMS.EquType: gams.control.globals.EquType`
- `GAMS.ArchType: gams.control.globals.ArchType`
- `GAMS.OSType: gams.control.globals.OSType`
- `GAMS.GAMSJob: gams.control.Job`
- `GAMS.GAMSModelInstance: gams.control.ModelInstance`
- `GAMS.SymbolUpdateType: gams.control.globals.SymbolUpdateType`
- `GAMS.GAMSModelInstanceOpt: gams.control.ModelInstanceOpt`
- `GAMS.GAMSModifier: gams.control.Modifier`
- `GAMS.GAMSOptions: gams.control.Options`
- `GAMS.ZeroResRep: gams.control.options.ZeroResRep`
- `GAMS.TraceOpt: gams.control.options.TraceOpt`
- `GAMS.TFormat: gams.control.options.TFormat`
- `GAMS.SysOut: gams.control.options.SysOut`
- `GAMS.Sys11: gams.control.options.Sys11`
- `GAMS.Sys10: gams.control.options.Sys10`

- GAMS.Suppress: gams.control.options.Suppress
- GAMS.SuffixAlgebraVars: gams.control.options.SuffixAlgebraVars
- GAMS.SuffixDLVars: gams.control.options.SuffixDLVars
- GAMS.StringChk: gams.control.options.StringChk
- GAMS.strictSingleton: gams.control.options.strictSingleton
- GAMS.StepSum: gams.control.options.StepSum
- GAMS.SolveOpt: gams.control.options.SolveOpt
- GAMS.SolveLink: gams.control.options.SolveLink
- GAMS.SolPrint: gams.control.options.SolPrint
- GAMS.ShowOSMemory: gams.control.options.ShowOSMemory
- GAMS.SavePoint: gams.control.options.SavePoint
- GAMS.Replace: gams.control.options.Replace
- GAMS.ReferenceLineNo: gams.control.options.ReferenceLineNo
- GAMS.PyMultInst: gams.control.options.PyMultInst
- GAMS.PutNR: gams.control.options.PutNR
- GAMS.ProcTreeMemMonitor: gams.control.options.ProcTreeMemMonitor
- GAMS.PreviousWork: gams.control.options.PreviousWork
- GAMS.PrefixLoadPath: gams.control.options.PrefixLoadPath
- GAMS.PageContr: gams.control.options.PageContr
- GAMS.On115: gams.control.options.On115
- GAMS.NoNewVarEqu: gams.control.options.NoNewVarEqu
- GAMS.MIIMode: gams.control.options.MIIMode
- GAMS.LstTitleLeftAligned: gams.control.options.LstTitleLeftAligned
- GAMS.LogLine: gams.control.options.LogLine
- GAMS.Listing: gams.control.options.Listing
- GAMS.Keep: gams.control.options.Keep
- GAMS.IntVarUp: gams.control.options.IntVarUp
- GAMS.InteractiveSolver: gams.control.options.InteractiveSolver
- GAMS.ImplicitAssign: gams.control.options.ImplicitAssign
- GAMS.HoldFixedAsync: gams.control.options.HoldFixedAsync
- GAMS.HoldFixed: gams.control.options.HoldFixed
- GAMS.gdxUels: gams.control.options.gdxUels
- GAMS.gdxConvert: gams.control.options.gdxConvert
- GAMS.gdxCompress: gams.control.options.gdxCompress
- GAMS.FreeEmbeddedPython: gams.control.options.FreeEmbeddedPython
- GAMS.ForceWork: gams.control.options.ForceWork

- GAMS.Filtered: gams.control.options.Filtered
- GAMS.FileCase: gams.control.options.FileCase
- GAMS.FDOpt: gams.control.options.FDOpt
- GAMS.ExecMode: gams.control.options.ExecMode
- GAMS.ErrMsg: gams.control.options.ErrMsg
- GAMS.Empty: gams.control.options.Empty
- GAMS.ECImplicitLoad: gams.control.options.ECImplicitLoad
- GAMS.DumpParms: gams.control.options.DumpParms
- GAMS.DumpOpt: gams.control.options.DumpOpt
- GAMS.Digit: gams.control.options.Digit
- GAMS.DFormat: gams.control.options.DFormat
- GAMS.CheckErrorLevel: gams.control.options.CheckErrorLevel
- GAMS.CharSet: gams.control.options.CharSet
- GAMS.LstCase: gams.control.options.LstCase
- GAMS.CaptureModelInstance: gams.control.options.CaptureModelInstance
- GAMS.AsyncSolLst: gams.control.options.AsyncSolLst
- GAMS.AppendOut: gams.control.options.AppendOut
- GAMS.AppendExpand: gams.control.options.AppendExpand
- GAMS.Action: gams.control.options.Action
- GAMS.GAMSParameter: gams.control.Parameter
- GAMS.GAMSParameterRecord: gams.control.ParameterRecord
- GAMS.GAMSRunParameters: gams.control.RunParameters
- GAMS.GAMSSet: gams.control.Set
- GAMS.GAMSSetRecord: gams.control.SetRecord
- GAMS.GAMSSymbol: gams.control.Symbol
- GAMS.GAMSSymbolDomainViolation: gams.control.SymbolDomainViolation
- GAMS.GAMSSymbolRecord: gams.control.SymbolRecord
- GAMS.GAMSVariable: gams.control.Variable
- GAMS.GAMSVariableRecord: gams.control.VariableRecord
- GAMS.GAMSWorkspace: gams.control.Workspace
- GAMS.GAMSWorkspaceInfo: gams.control.WorkspaceInfo

7.6.1.2 Getting Started

The GAMS Matlab Control API is built on top of the GAMS Java Control API and provides convenient access to GAMS from within Matlab (2017b or later) and Octave (5.2 or later).

Install

The Matlab API is located in `api/matlab`. Add this directory to the Matlab path:
```
addpath("[PathToGAMS]/api/matlab")
```

Furthermore, make Matlab aware of the GAMS Java API (see also GAMS Java API below):
```
javaaddpath("[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar")
```

GAMS Java API

Instead of adding the GAMS Java API dynamically as done above, it is also possible to add it statically. For this, locate `javaclasspath.txt` and add the path `[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar`. You can find the location with `cd(prefdir)` in Matlab or `which javaclasspath` in Octave. Finally, restart Matlab or Octave. Adding the GAMS Java API statically is more efficient.

Attention

> Since GAMS 44.1.0, the GAMS Java API requires at least Java SE 11 to compile and run. Matlab usually ships Java SE 8. In that case use `GAMSJavaAPI-8.jar` instead of `GAMSJavaAPI.jar`. However, new or updated functionalities that are released after GAMS 43 are not avaiable in this Java API version. Since Matlab R2023a, Matlab does support Java SE 11 (although it may have to be installed by the user). You can use `jenv system` to use the Java installed on the system instead of Matlab's shipped Java.

Important Classes

This section provides a quick overview of some fundamental classes of the GAMS Control package. Their usage is demonstrated by an extensive set of examples (located in `api/matlab/examples`). All GAMS Matlab Control API classes are contained within the package `gams.control` (with subpackages `gams.control.globals`, `gams.control.engine` and `gams.control.options`). The package provides objects to interact with the General Algebraic Modeling System (GAMS). Objects in this package allow convenient exchange of input data and model results (**Database**) and help to create and run GAMS models (**Job**), that can be customized by GAMS options (**Options**). Furthermore, it introduces a way to solve a sequence of closely related models in the most efficient way (**ModelInstance**).

7.7 R API

The R API consists of GAMS Transfer R.

7.7.1 GAMS Transfer R

GAMS Transfer is a package to maintain GAMS data outside a GAMS script in a programming language like Python, Matlab, or R. It allows the user to add GAMS symbols (Sets, Aliases, Parameters, Variables and Equations), to manipulate GAMS symbols, as well as read/write symbols to different data endpoints. GAMS Transfer's main focus is the highly efficient transfer of data between GAMS and the target programming language, while keeping those operations as simple as possible for the user. In order to achieve this, symbol records - the actual and potentially large-scale data sets - are stored in native data structures of the corresponding programming languages. The benefits of this approach are threefold: (1) The user is usually very familiar with these data structures, (2) these data structures come with a large tool box for various data operations, and (3) optimized methods for reading from and writing to GAMS can transfer the data as a bulk - resulting in the high performance of this package.

7.7.1.1 Getting Started

Install

The user must download and install the latest version of GAMS in order to install GAMS Transfer R. GAMS Transfer R can then be installed from either the source package or from the binary package. Installing using binary package is recommended for the new users since it is easier and does not involve compiling the package source.

The binary packages are platform dependent and the instructions for each supported platform are shown below.

Windows
```
install.packages("[PathToGAMS]/apifiles/R/gamstransfer/binary/gamstransfer.zip", type="binary")
```

Linux
```
install.packages("[PathToGAMS]/apifiles/R/gamstransfer/binary/gamstransfer.tar.gz")
```

macOS (same for ARM64 CPUs)
```
install.packages("[PathToGAMS]/apifiles/R/gamstransfer/binary/gamstransfer.tgz", type="binary")
```

GAMS Transfer R depends on packages `R6`, `R.utils`, and `Rcpp`, `collections`.

Installation using the source package can be done with the following command.
```
install.packages("[PathToGAMS]/apifiles/R/gamstransfer/source/gamstransfer_r.tar.gz", dependencies=TRUE)
```

When building from source, GAMS Transfer R also requires the library `zlib`. The user can point to `zlib` by adding the directory containing `zlib` to the environment variable `PATH` or by using `configure.vars` argument of install.packages().

Examples

GDX Read

Reading in all symbols can be accomplished with one line of code (we reference data from the **trnsport.gms** example).
```
library(gamstransfer)
m = Container$new("trnsport.gdx")
```

All symbols are internally stored in the Container data field (dictionary). The users should never have to access or modify the `data` field. The symbols can be accessed via `m[<symbol_name>]` and the records can be accessed via `m[<symbol_name>]$records`. Symbol records are stored in the data frame format.

GDX Write

There are five symbol classes within GAMS Transfer R: Sets, Parameters, Variables, Equations, and Aliases. For purposes of this quick start, we show how to recreate the `distance` data structure from the **trnsport.gms** model (the parameter `d`). This brief example shows how users can achieve "GAMS-like" functionality, but within an R environment. GAMS Transfer R leverages the object oriented programming to simplify the syntax.
```
library(gamstransfer)
m = Container$new()
# create the sets i, j
i = Set$new(m, "i", records = c("seattle", "san-diego"), description = "supply")
j = Set$new(m, "j", records = c("new-york", "chicago", "topeka"), description = "markets")
# add "d" parameter -- domain linked to set objects i and j
d = Parameter$new(m, "d", c(i, j), description = "distance in thousands of miles")
# create some data as a generic data frame
dist = data.frame(
```

```
    from = c("seattle", "seattle", "seattle",
    "san-diego", "san-diego", "san-diego"),
    to = c("new-york", "chicago", "topeka",
      "new-york", "chicago", "topeka"),
    thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
# setRecords will automatically convert the dist data frame into
# a standard data frame format
d$setRecords(dist)
# write the GDX
m$write("out.gdx")
```

This example shows a few fundamental features of GAMS Transfer R:

1. A Container is analogous to a GDX file
2. Symbols will always be linked to a Container (notice that we always pass the Container reference `m` to the symbol constructor)
3. Records can be added to a symbol with the `setRecords()` method, through the `records` constructor argument (internally calls `setRecords()`), or through directly setting the `records` field. GAMS Transfer R will convert many common R data structures into a standard format.
4. Domain linking is possible by passing domain set objects to other symbols.
5. Writing a GDX file can be accomplished in one line with the write() method.

Full Example

It is possible to use GAMS Transfer R to recreate the **trnsport.gms** results in GDX form. As part of this example, we also introduce the write() method (and generate `new.gdx`). We will discuss it in more detail in the following section: Data Exchange with GDX.

```
library(gamstransfer)
# create an empty Container object
m = Container$new()
# add sets
i = Set$new(m, "i", records=c("seattle", "san-diego"), description="supply")
j = Set$new(m, "j", records=c("new-york", "chicago", "topeka"), description="markets")
# add parameters
a = Parameter$new(m, "a", c("*"), description="capacity of plant i in cases")
b = Parameter$new(m, "b", j, description="demand at market j in cases")
d = Parameter$new(m, "d", c(i, j), description="distance in thousands of miles")
f = Parameter$new(
    m, "f", records=90, description="freight in dollars per case per thousand miles"
)
c = Parameter$new(
    m, "c", c(i, j), description="transport cost in thousands of dollars per case"
)
# set parameter records
cap = data.frame(plant = c("seattle", "san-diego"), n_cases = c(350, 600))
a$setRecords(cap)
dem = data.frame(market = c("new-york","chicago", "topeka"), n_cases = c(325, 300, 275))
b$setRecords(dem)
dist = data.frame(
    from = c("seattle", "seattle", "seattle",
    "san-diego", "san-diego", "san-diego"),
    to = c("new-york", "chicago", "topeka",
      "new-york", "chicago", "topeka"),
    thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
d$setRecords(dist)
# c(i,j) = f * d(i,j) / 1000;
cost = d$records
cost$value = f$records$value * cost$value/1000
c$setRecords(cost)
# add variables
q = data.frame(
    from = c("seattle", "seattle", "seattle",
    "san-diego", "san-diego", "san-diego"),
    to = c("new-york", "chicago", "topeka",
      "new-york", "chicago", "topeka"),
    level = c(50, 300, 0, 275, 0, 275),
    marginal = c(0, 0, 0.036, 0, 0.009, 0)
)
x = Variable$new(m, "x", "positive", c(i, j), records=q, description="shipment quantities in cases")
```

```
z = Variable$new(
    m,
    "z",
    records=data.frame(level = 153.675),
    description="total transportation costs in thousands of dollars"
    )
# add equations
cost = Equation$new(m, "cost", "eq", description="define objective function")
supply = Equation$new(m, "supply", "leq", i, description="observe supply limit at plant i")
demand = Equation$new(m, "demand", "geq", j, description="satisfy demand at market j")
# set equation records
cost$setRecords(data.frame(level = 0, marginal = 1, lower = 0, upper = 0))
supplies = data.frame(
  from = c("seattle", "san-diego"),
  level = c(350, 550),
  marginal = c(SpecialValues$EPS, 0),
  lower = c(SpecialValues$NEGINF, SpecialValues$NEGINF),
  upper = c(350, 600)
)
supply$setRecords(supplies)
demands = data.frame(
  from = c("new-york", "chicago", "topeka"),
  level = c(325, 300, 275),
  marginal = c(0.225, 0.153, 0.126),
  lower = c(325, 300, 275)
)
demand$setRecords(demands)
m$write("new.gdx")
```

It can be observed from the above example that a typical work flow for writing using GAMS Transfer R is creating a container, filling it with symbols (Sets, Parameters, Variables, Equations, and Aliases), and write it to a GDX file. To read a GDX file, a Container can simply be initialized with the GDX file name as an argument.

These examples introduced the reader to the GAMS Transfer R syntax. In the remaining sections, we will present details about the core functionality and dig further into the syntax.

7.7.1.2 Manual

Container

Storing, manipulating, and transforming sparse data requires that it lives within an environment. This data can then be linked together to enable various operations. In GAMS Transfer R, we refer to this "environment" as the `Container`, it is the main repository for storing and linking our data. Linking data enables data operations such as implicit set growth, domain checking, and data format transformations (to dense/sparse matrix formats). A Container also enables different read/write operations.

Creating a `Container` is a simple matter of initializing an object. For example:

```
library(gamstransfer)
m = Container$new()
```

This new `Container` object, here called `m`, contains a number of convenient fields and methods that allow the user to interact with the symbols that are in the `Container`. Some of these methods are simply used to filter out different types of symbols, other methods are used to numerically characterize the data within each symbol.

The `Container` constructor arguments are:

Argument	Type	Description	Required	Default
`loadFrom`	`string`, `Container`	Name of the GDX file being read into the `Container`	No	NULL

Container fields

Field	Description	Type
`data`	main `dict` that is used to store all symbol data	`dict` (collections package)
`summary`	output a `list` containing the container information	`list`

Symbols are organized in the `Container` under the `data` field. The users should never have to access the `data` field. Symbols in the container can be accessed via `m[<symbol_name>]`.

Container methods

Method	Description	Arguments/Defaults	Returns
`addAlias`	add an `Alias`	`name` (`string`) `aliasWith` (`Set`,`Alias`)	`Alias` object
`addUniverseAlias`	add a `UniverseAlias`	`name` (`string`)	`UniverseAlias` object
`addEquation`	add an `Equation`	`name` (`string`) `type` (`string`) `domain=NULL` (`string`,`list`) `records=NULL` (`data frame`,`vector`,`NULL`) `domainForwarding=FALSE` (`logical`) `description=""` (`string`)	`Equation` object
`addParameter`	add a `Parameter`	`name` (`string`) `domain=NULL` (`string`,`list`) `records=NULL` (`data frame`,`array`, `matrix`) `domainForwarding=FALSE` (`logical`) `description=""` (`string`)	`Parameter` object
`addSet`	add a `Set`	`name` (`string`) `domain="*"` (`string`,`list`) `isSingleton=FALSE` (`logical`) `records=NULL` (`data frame`,`array`,`matrix`) `domainForwarding=FALSE` (`logical`) `description=""` (`string`)	`Set` object

Method	Description	Arguments/Defaults	Returns
addVariable	add a Variable	name (string) type="free" (string) domain=NULL (string,list) records=NULL (data frame,array,matrix) domainForwarding=FALSE (logical) description="" (string)	Variable object
copy	copies symbols to the destination container. overwrite=TRUE overwrites the symbols with the same name in destination. Symbol domains are relaxed if destination Container does not contain equivalent domain sets	destination (Container), symbols=NULL(character), overwrite=FALSE (logical)	NULL
countDomainViolations	returns a named list containing number of domain violations symbols with non-zero domain violations	symbols=NULL (character) - if NULL, assumes all symbols	list
countDuplicateRecords	returns a named list containing number of duplicate records for symbols with non-zero duplicate records	symbols=NULL (character) - if NULL, assumes all symbols	list
describeAliases	create a summary table with descriptive statistics for Aliases	symbols=NULL (string) - if NULL, assumes all Aliases	data frame
describeEquations	create a summary table with descriptive statistics for Equations	symbols=NULL (string) - if NULL, assumes all equations	data frame
describeParameters	create a summary table with descriptive statistics for Parameters	symbols=NULL (string) - if NULL, assumes all parameters	data frame
describeSets	create a summary table with descriptive statistics for Sets	symbols=NULL (string) - if NULL, assumes all sets	data frame
describeVariables	create a summary table with descriptive statistics for Variables	symbols=NULL (string) - if NULL, assumes all variables	data frame
dropDomainViolations	drops domain violations for symbols in the Container	symbols=NULL (character) - if NULL, assumes all symbols	-
dropDuplicateRecords	drops duplicate records for symbols in the Container	symbols=NULL (character) - if NULL, assumes all symbols, keep= first/last	-

Method	Description	Arguments/Defaults	Returns
equals	Check if two Container objects are equal	other (Container), verbose=FALSE	logical
getAliases	returns a list of object references for Aliases that are isValid	isValid=NULL (logical)	list
getDomainViolations	gets domain violations for symbols in the Container	symbols=NULL (character) - if NULL, assumes all symbols	list of DomainViolation
getEquations	returns a list of object references for Equations that are isValid	isValid=NULL (logical)	list
getParameters	returns a list of object references for Parameters that are isValid	isValid=NULL (logical)	list
getSets	returns a list of object references for Sets that are isValid	isValid=NULL (logical)	list
getSymbolNames	returns the original symbol names for a vector names of any case	names (character)	character
getSymbols	returns a list of object references for symbols	symbols (character) - if NULL assumes all symbols	list
getVariables	returns a list of object references for Variables that are isValid	isValid=NULL (logical)	list
getUELs	returns the UELs used in the Container	symbols=NULL (character), ignoreUnused = FALSE	character vector
hasDomainViolations	returns TRUE if any symbol in the container has domain violations, FALSE otherwise	symbols=NULL (character) - if NULL, assumes all symbols	logical
hasDuplicateRecords	returns TRUE if any symbol in the container has duplicate records, FALSE otherwise	symbols=NULL (character) - if NULL, assumes all symbols	logical
hasSymbols	checks if symbol names for a vector names exists in the container	names (character)	logical
isValid	TRUE if all symbols in the Container are valid	symbols=NULL (character) - if NULL, assumes all symbols, verbose=FALSE (logical) force=FALSE (logical)	logical

Method	Description	Arguments/Defaults	Returns
`listAliases`	list all aliases (`isValid=NULL`), list all valid aliases (`isValid=TRUE`), list all invalid aliases (`isValid=FALSE`) in the container	`isValid=NULL` (`logical`)	`vector`
`listEquations`	list all equations (`isValid=NULL`), list all valid equations (`isValid=TRUE`), list all invalid equations (`isValid=FALSE`) in the container	`isValid=NULL` (`logical`) `types=NULL` (`character` of eqaution types) - if `NULL`, assumes all types	`vector`
`listParameters`	list all parameters (`isValid=NULL`), list all valid parameters (`isValid=TRUE`), list all invalid parameters (`isValid=FALSE`) in the container	`isValid=NULL` (`logical`)	`vector`
`listSets`	list all sets (`isValid=NULL`), list all valid sets (`isValid=TRUE`), list all invalid sets (`isValid=FALSE`) in the container	`isValid=NULL` (`logical`)	`vector`
`listSymbols`	list all symbols (`isValid=NULL`), list all valid symbols (`isValid=TRUE`), list all invalid symbols (`isValid=FALSE`) in the container	`isValid=NULL` (`logical`)	`vector`
`listVariables`	list all variables (`isValid=NULL`), list all valid variables (`isValid=TRUE`), list all invalid variables (`isValid=FALSE`) in the container	`isValid=NULL` (`logical`) `types=NULL` (`character` of variable types) - if `NULL`, assumes all types	`vector`
`read`	main method to read `loadFrom`, can be provided with a list of `symbols` to read in subsets, `records` controls if symbol records are loaded or just metadata	`loadFrom` (`string`, `Container`) `symbols=NULL` (`string`) `records=TRUE` (`logical`)	`NULL`

Method	Description	Arguments/Defaults	Returns
removeSymbols	symbols to remove from the Container, also sets the symbols' container to NULL; removes symbol links from the Container; Aliases are also removed if the parent set is removed	symbols (string) - if NULL assumes all symbols	NULL
removeUELs	removes uels from all symbols the Container	uels=NULL (character), symbols=NULL (character). If Symbols is NULL UELs are removed for all symbols.	NULL
renameSymbol	rename a symbol oldName in the Container	oldName (string), newName (string)	NULL
renameUELs	renames uels in the Container	uels (named character), symbols=NULL (character), allowMerge=FALSE (logical). If allowMerge = TRUE, the underlying integer mapping of a factor is allowed to change to offer additional data flexibility. If Symbols is NULL UELs are renamed for all symbols.	NULL
reorderSymbols	reorder symbols in order to avoid domain violations	-	NULL
write	main bulk write method to a writeTo target	writeTo (string) symbols=NULL (character) - if NULL, assumes all symbols compress=FALSE (logical) uelPriority=NULL (string) mode="mapped" (character "string" or "mapped")	NULL

Symbols

In GAMS Transfer R, a symbol is either a GAMS Set, Parameter, Variable, Equation, Alias. In GAMS Transfer R, a symbol cannot live on its own, but is always part of a Container.

Create a symbol

There are two different ways to create a GAMS set and add it to a Container.

1. Use symbol constructor for Set, Parameter, Variable, Equation, Alias
```
library(gamstransfer)
m = Container$new()
p = Parameter$new(m, "p")
```
2. Use the Container methods addSet, addParameter, addVariable, addEquation, addAlias (which internally calls the symbol constructor)
```
library(gamstransfer)
m = Container$new()
p = m$addParameter("p")
```

Add Symbol Records

Three possibilities exist to assign symbol records:

1. Setting the argument `records` in the symbol constructor/Container method (internally calls `setRecords`)
2. Using the symbol method `setRecords`
3. Setting the field `records` directly

If the data is in a convenient format, a user may want to pass the records directly within the set constructor. This is an optional keyword argument and internally the symbol constructor will simply call the `setRecords` method. The symbol method `setRecords` is a convenience method that transforms the given data into an approved data frame format (see Standard Data Formats). Many native R data types can be easily transformed into data frames, so the `setRecords` method will accept a number of different types for input. The `setRecords` method is called internally on any data structure that is passed through the `records` argument.

The examples of setting records using the above-mentioned methods can be found in the respective documentation for each symbol (Set, Parameter, Variable, Equation, and Alias).

Set

Set Constructor

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the Container object that the symbol is being added to	Yes	-
`name`	`string`	Name of symbol	Yes	-

Argument	Type	Description	Required	Default
`domain`	`list`	String, List of domains given either as a string ("∗" for universe set) or as a reference to a Set object	No	"∗"
`isSingleton`	`logical`	Indicates if set is a singleton set (`TRUE`) or not (`FALSE`)	No	FALSE
`records`	string vector, data frame	Symbol records	No	NULL
`domainForwarding`	`logical`	Flag or a vector of flags that forces set elements to be recursively included in corresponding parent sets (i.e., implicit set definition)	No	FALSE
`description`	`string`	Description of symbol	No	""

Set Fields

Field	Description	Type
`description`	description of symbol	`string`
`dimension`	dimension of symbol, setting dimension is a shorthand notation to set `domain` to a list of size n containing "∗"	`integer`
`domain`	list of domains given either as string (∗ for universe set) or as reference to the Set/Alias object	`list`
`domainForwarding`	flag or a vector of flags that forces set elements to be recursively included in corresponding parent sets (i.e., implicit set definition)	`logical`
`domainLabels`	column headings for the `records` data frame	`list` of `string`
`domainNames`	string version of domain names	`list` of `string`
`domainType`	`none`, `relaxed` or `regular` depending on state of domain links	`string`
`isSingleton`	`logical` if symbol is a singleton set	`logical`
`name`	name of symbol	`string`
`numberRecords`	number of symbol records (i.e., returns `nrow(self$records)` if not `NULL`)	`integer`
`records`	the main symbol records	`data frame`
`container`	reference to the Container that the symbol belongs to	`Container`
`summary`	output a `list` of only the metadata	`list`

Set Methods

Method	Description	Arguments/Defaults	Returns
`addUELs`	adds UELs to the symbol	`uels` (character), `dimension=NULL`	`NULL`

Method	Description	Arguments/Defaults	Returns
copy	copies the symbol to the destination container. overwrite=TRUE overwrites the symbol with the same name in destination. Symbol domains are relaxed if destination Container does not contain equivalent domain sets	destination (Container), overwrite=FALSE	NULL
countDomainViolations	returns the number of domain violations for the symbol	-	numeric
countDuplicateRecords	returns the number of duplicate records for the symbol	-	numeric
findDomainViolations	get a view of records data frame that contain any domain violations	-	data.frame
findDuplicateRecords	get a view of records data frame that contain duplicate records. keep = "first" (finds all duplicates while keeping the first instance as unique), keep="last" (finds all duplicates while keeping the last instance as unique), or keep=FALSE (finds all duplicates)	keep= first	data.frame
dropDomainViolations	drops domain violations for the symbol	-	-
dropDuplicateRecords	drops duplicate records for the Symbol	keep= first/last	-
equals	Check if two Symbol objects are equal	other (Symbol), checkUELs=TRUE, checkElementText=TRUE, checkMetaData=TRUE, verbose=FALSE	logical

Method	Description	Arguments/Defaults	Returns
generateRecords	convenience method to set standard data.frame formatted records. Will generate records with the Cartesian product of all domain sets. The density argument can take any value on the interval [0,1]. If density is < 1, then randomly selected records will be removed. density will accept a numeric of length 1 or dimension. This allows users to specify a density per symbol dimension (when vector) or density of the records dataframe. Random number state can be set with seed argument.	density=1.0 (numeric) seed=NULL (integer)	NULL
getDomainViolations	gets domain violations for the symbol	-	list of DomainViolation
getSparsity	get the sparsity of the symbol w.r.t the size of full cartesian product of the domain sets	-	numeric
getUELs	returns the UELs used in the Symbol	dimension = NULL, codes = NULL, ignoreUnused=FALSE	character vector
hasDomainViolations	returns TRUE if the symbol contains domain violations, FALSE otherwise	-	logical
hasDuplicateRecords	returns TRUE if the symbol contains duplicate records, FALSE otherwise	-	logical
isValid	checks if the symbol is in a valid format, throw exceptions if verbose=TRUE, recheck a symbol if force=TRUE	verbose=FALSE force=FALSE	logical
removeUELs	removes UELs from the symbol	uels=NULL (character), dimension=NULL	NULL

Method	Description	Arguments/Defaults	Returns
renameUELs	renames UELs in the symbol	uels(character of same length as current UELs or named character vector), dimension=NULL, allowMerge=FALSE (logical). If allowMerge = TRUE, the underlying integer mapping of a factor is allowed to change to offer additional data flexibility	NULL
reorderUELs	reorders UELs in the symbol that appear in the symbol dimensions. If uels is NULL, the UELs are reordered based on symbol records, unused UELs are moved to the end. If dimensions is NULL then reorder UELs in all dimensions of the symbol	uels=NULL(NULL or character of same length as current UELs or named character vector), dimension=NULL (numeric)	NULL
setRecords	main convenience method to set standard data frame formatted records	records (string vector, list, data frame)	NULL
setUELs	sets UELs for the Symbol	uels(character), dimension, rename=FALSE	NULL

Adding Set Records

Three possibilities exist to assign symbol records to a set: We show a few examples of ways to create differently structured sets.

Example #1 - Create a 1D set from a vector

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = c("seattle", "san-diego"))
# NOTE: the above syntax is equivalent to -
# i = Set$new(m, "i")
# i$setRecords(c("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# m$addSet("i", records= c("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# i = m$addSet("i")
# i$setRecords(c("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# m$addSet("i")
# m["i"]$setRecords(c("seattle", "san-diego"))
> i$records
        uni
1   seattle
2 san-diego
```

Example #2 - Create a 2D set from a list

```
library(gamstransfer)
m = Container$new()
k = Set$new(m, "k", c("*", "*"), records=list("seattle", "san-diego"))
# NOTE: the above syntax is equivalent to -
# k = Set$new(m, "k", c("*", "*"))
# k$setRecords(list("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# m$addSet("k", c("*","*"), records=list("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# k = m$addSet("k", c("*","*"))
# k$setRecords(list("seattle", "san-diego"))
# NOTE: the above syntax is also equivalent to -
# m$addSet("k", c("*", "*"))
# m["k"]$setRecords(list("seattle", "san-diego"))
> k$records
    uni_1     uni_2
1 seattle san-diego
```

Example #3 - Create a 1D set from a data frame slice

```
library(gamstransfer)
m = Container$new()
dist = data.frame(
    from = c("seattle", "seattle", "seattle",
    "san-diego", "san-diego", "san-diego"),
    to = c("new-york", "chicago", "topeka",
      "new-york", "chicago", "topeka"),
    thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
l = Set$new(m, "l", records = unique(dist[["from"]]))
# NOTE: the above syntax is equivalent to -
# l = Set$new(m, "l")
# l$setRecords(unique(dist[["from"]]))
# NOTE: the above syntax is also equivalent to -
# m$addSet("l", records=unique(dist[["from"]]))
# NOTE: the above syntax is also equivalent to -
# l = m$addSet("l")
# l$setRecords(unique(dist[["from"]]))
# NOTE: the above syntax is also equivalent to -
# m$addSet("l")
# m["l"]$setRecords(unique(dist[["from"]]))
> l$records
      uni
1   seattle
2 san-diego
```

Set element text is very handy when labeling specific set elements within a set. A user can add a set element text directly with a set element. Note that it is not required to label all set elements, as can be seen in the following example.

Example #4 - Add set element text

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i",
records = data.frame(city=c("seattle", "san-diego", "washington_dc"),
text=c("home of sub pop records", "", "former gams hq")))
# NOTE: the above syntax is equivalent to -
#
# i = Set$new(m, "i")
# i_recs = data.frame(city=c("seattle", "san-diego", "washington_dc"),
# text=c("home of sub pop records", "", "former gams hq"))
#
# i$setRecords(i_recs)
# NOTE: the above syntax is also equivalent to -
# m$addSet("i", records=i_recs)
# NOTE: the above syntax is also equivalent to -
# i = m$addSet("i")
# i$setRecords(i_recs)
# NOTE: the above syntax is also equivalent to -
# m$addSet("i")
# m["i"]$setRecords(i_recs)
> i$records
          city            element_text
1       seattle home of sub pop records
```

```
2     san-diego
3 washington_dc          former gams hq
```

The primary advantage of the `setRecords` method is that GAMS Transfer R will convert many different (and convenient) data types into the standard data format (a data frame). Users that require higher performance will want to directly pass the Container a reference to a valid data frame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process.
In this section, we walk the user through an example of how to set records directly.

Example #5 - Directly set records (1D set)

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", description = "supply")
# create a standard format data frame
df_i = data.frame(uni_1 = c("seattle", "san-diego"),
element_text = c("", ""))
# need to create categorical column type, referencing elements already in df_i
df_i$uni_1 = factor(df_i$uni_1, ordered = TRUE)
# set the records directly
i$records = df_i
> i$isValid()
[1] TRUE
```

Stepping through this example we take the following steps:

1. Create an empty Container
2. Create a GAMS set `i` in the Container, but do not set the `records`
3. Create a data frame (manually, in this example) taking care to follow the standard format
4. The data frame has the right shape and column labels so we can proceed to set the records.
5. We need to cast the `uni_1` column as a `factor`, so we create a custom ordered category type using `factor`
6. Finally, we set the records directly by passing a reference to `df_i` into the symbol records attribute. The setter function of `records` checks that a data frame is being set, but does not check validity. Thus, as a final step, we call the `$isValid()` method to verify that the symbol is valid.

Note

Users can debug their data frames by running <`symbol_name`>`$isValid(verbose=TRUE)` to get feedback about their data.

Example #6 - Directly set records (1D subset)

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("seattle", "san-diego"), description="supply")
j = Set$new(m, "j", i, description="supply")
# create a standard format data frame
df_j = data.frame(i_1 = c("seattle"), "element_text" = c(""))
# create the categorical column type
df_j$i_1 = factor(df_j$i_1, levels = i$records[, 1], ordered = TRUE)
# set the records
j$records = df_j
> j$isValid()
[1] TRUE
```

This example is more subtle in that we want to create a set `j` that is a subset of `i`. We create the set `i` using the `setRecords` method but then set the records directly for `j`. There are two important details to note: 1) the column labels in `df_j` now reflect the standard format for a symbol with a domain set (as opposed to the universe) and 2) we create the factors by referencing the parent set (`i`) for the `levels` (instead of referencing itself).

Note

One can also use the genereateRecords() method to automatically populate randomly generated symbol records in the standard format.

Parameter

Parameter Constructor

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the Container object that the symbol is being added to	Yes	-
`name`	`string`	Name of symbol	Yes	-
`domain`	`list`	List of domains given either as a string ("∗" for universe set) or as a reference to a Set object, an empty domain list will create a scalar parameter	No	NULL
`records`	many	Symbol records	No	NULL
`domainForwarding`	`logical`	Flag or a vector of flags that forces parameter elements to be recursively included in corresponding parent sets (i.e.,implicit set definition)	No	FALSE
`description`	`string`	Description of symbol	No	""

Parameter Fields

Field	Description	Type
`container`	reference to the Container that the symbol belongs to	`Container`
`defaultValues`	default values for the symbol	`numeric`
`description`	description of symbol	`character`
`dimension`	dimension of symbol, setting dimension is a shorthand notation to set `domain` to a list of size n containing "∗"	`numeric,integer`
`domain`	list of domains given either as a string (`"*"` for universe set) or as a reference to the Set/Alias object	`list`
`domainForwarding`	Flag or a vector of flags that forces parameter elements to be recursively included in corresponding parent sets (i.e.,implicit set definition)	`logical`
`domainLabels`	column headings for the `records` data frame	`list` of `string`
`domainNames`	string version of domain names	`list` of `string`
`domainType`	`none`, `relaxed` or `regular` depending on state of domain links	`string`
`isScalar`	TRUE if `self$dimension = 0`	`logical`
`name`	name of symbol	`string`
`numberRecords`	number of symbol records (i.e., returns `nrow(self$records)` if not `NULL`)	`integer`
`records`	the main symbol records	`data frame`

Field	Description	Type
shape	a vector describing the array dimensions if records were converted with $toDense()	vector
summary	output a list of only the metadata	list

Parameter Methods

Method	Description	Arguments/Defaults	Returns
addUELs	adds UELs to the symbol	uels (character), dimension=NULL	NULL
copy	copies the symbol to the destination container. overwrite=TRUE overwrites the symbol with the same name in destination. Symbol domains are relaxed if destination Container does not contain equivalent domain sets	destination (Container), overwrite=FALSE	NULL
countDomainViolations	returns the number of domain violations for the symbol	-	numeric
countDuplicateRecords	returns the number of duplicate records for the symbol	-	numeric
countEps	total number of SpecialValues$EPS in value column	-	integer
countNA	total number of SpecialValues[["NA"]] in value column	-	integer
countNegInf	total number of SpecialValues$NEGINF in value column	-	integer
countPosInf	total number of SpecialValues$POSINF in value column	-	integer
countUndef	total number of SpecialValues$UNDEF in value column	-	integer
dropDomainViolations	drops domain violations for the symbol	-	-

Method	Description	Arguments/Defaults	Returns
dropDuplicateRecords	drops duplicate records for the Symbol. keep = "first" (drops all duplicates while keeping the first instance as unique), keep="last" (drops all duplicates while keeping the last instance as unique), or keep=FALSE (drops all duplicates)	keep = "first"	-
equals	Check if two Symbol objects are equal	other (Symbol), checkUELs=TRUE, checkMetaData=TRUE, rtol=0 (relative tolernace), atol=0 (absolute tolerance), verbose=FALSE	logical
findDomainViolations	get a view of records data frame that contain any domain violations	-	data.frame
findDuplicateRecords	get a view of records data frame that contain duplicate records. keep = "first" (finds all duplicates while keeping the first instance as unique), keep="last" (finds all duplicates while keeping the last instance as unique), or keep=FALSE (finds all duplicates)	keep= first	data.frame

Method	Description	Arguments/Defaults	Returns
generateRecords	convenience method to set standard data.frame formatted records. Will generate records with the Cartesian product of all domain sets. The density argument can take any value on the interval [0,1]. If density is < 1, then randomly selected records will be removed. density will accept a numeric of length 1 or dimension. This allows users to specify a density per symbol dimension (when vector) or the density of records dataframe. Random numbers can be generated by passing a user-defined function func(size) to func argument (runif() by default). Random number state can be set with seed argument.	density=1.0 (numeric) func=runif() seed=NULL (integer)	NULL
getDomainViolations	gets domain violations for the symbol	-	list of DomainViolation
getMaxValue	get the maximum value in value column	-	numeric
getMaxAbsValue	get the maximum absolute value in value column	-	numeric
getMeanValue	get the mean value in value column	-	numeric
getMinValue	get the minimum value in value column	-	numeric
getSparsity	get the sparsity of the symbol w.r.t the size of full cartesian product of the domain sets	-	numeric
getUELs	returns the UELs used in the Symbol	dimension = NULL, codes = NULL, ignoreUnused=FALSE	character vector
hasDomainViolations	returns TRUE if the symbol contains domain violations, FALSE otherwise	-	logical
hasDuplicateRecords	returns TRUE if the symbol contains duplicate records, FALSE otherwise	-	logical

Method	Description	Arguments/Defaults	Returns
isValid	checks if the symbol is in a valid format, throw exceptions if verbose=TRUE, recheck a symbol if force=TRUE	verbose=FALSE force=FALSE	logical
removeUELs	removes UELs from the symbol	uels=NULL (character), dimension=NULL	NULL
renameUELs	renames UELs in the symbol	uels(character of same length as current UELs or named character vector), dimension=NULL, allowMerge=FALSE (logical). If allowMerge = TRUE, the underlying integer mapping of a factor is allowed to change to offer additional data flexibility	NULL
reorderUELs	reorders UELs in the symbol that appear in the symbol dimensions. If uels is NULL, the UELs are reordered based on symbol records, unused UELs are moved to the end. If dimensions is NULL then reorder UELs in all dimensions of the symbol	uels=NULL(NULL or character of same length as current UELs or named character vector), dimension=NULL (numeric)	NULL
setRecords	main convenience method to set standard data frame records	records (many types)	NULL
setUELs	sets UELs for the Symbol	uels(character), dimension, rename=FALSE	NULL
toDense	convert value column of a symbol to a dense matrix or array format	-	array or matrix
whereMax	find the row number in records data frame with a maximum value (return the first instance only)	-	integer
whereMaxAbs	find the row number in records data frame with a maximum absolute value (return the first instance only)	-	integer

Method	Description	Arguments/Defaults	Returns
`whereMin`	find the row number in records data frame with a minimum value (return the first instance only)	-	`integer`

Adding Parameter Records

Three possibilities exist to assign symbol records to a parameter: We show a few examples of ways to create differently structured parameters:

Example #1 - Create a GAMS scalar

```
library(gamstransfer)
m = Container$new()
pi = Parameter$new(m, "pi", records = 3.14159)
# NOTE: the above syntax is equivalent to -
# pi = Parameter$new(m, "pi")
# pi$setRecords(3.14159)
# NOTE: the above syntax is also equivalent to -
# m$addParameter("pi", records=3.14159)
# NOTE: the above syntax is also equivalent to -
# pi = m$addParameter("pi")
# pi$setRecords(3.14159)
# NOTE: the above syntax is also equivalent to -
# m$addParameter("pi")
# m["pi"]$setRecords(3.14159)
> pi$records
    value
1 3.14159
```

Note

GAMS Transfer R will still convert scalar values to a standard format (i.e., a data frame with a single row and column).

Example #2 - Create a 2D parameter (defined over a set) from a

data frame slice

```
library(gamstransfer)
dist = data.frame(
    from = c("seattle", "seattle", "seattle",
    "san-diego", "san-diego", "san-diego"),
    to = c("new-york", "chicago", "topeka",
      "new-york", "chicago", "topeka"),
    thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
m = Container$new()
i = Set$new(m, "i", "*", records = unique(dist$from))
j = Set$new(m, "j", "*", records = unique(dist$to))
a = Parameter$new(m, "a", c(i, j), records = dist)
> a$toDense()
     [,1] [,2] [,3]
[1,]  2.5  1.7  1.8
[2,]  2.5  1.8  1.4
# use a$toDense() to create a new (and identicial) parameter a2
a2 = Parameter$new(m, "a2", c(i, j), records = a$toDense())
> a2$records
          i        j value
1   seattle new-york   2.5
2   seattle  chicago   1.7
3   seattle   topeka   1.8
4 san-diego new-york   2.5
5 san-diego  chicago   1.8
6 san-diego   topeka   1.4
```

Example #3 - Create a 2D parameter from an array using setRecords

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i_", 1:5))
j = Set$new(m, "j", records=paste0("j_", 1:5))
# create the parameter with linked domains (these will control the
# $shape of the symbol)
a = Parameter$new(m, "a", c(i, j))
# here we use the $shape field to easily generate a dense random array
a$setRecords(array(runif(prod(a$shape()), min = 1, max = 10),
dim = a$shape() ))
> a$toDense()
         [,1]     [,2]     [,3]     [,4]     [,5]
[1,] 3.837345 3.632743 9.003275 4.097475 8.608477
[2,] 7.217257 2.465452 3.286330 2.366017 8.822535
[3,] 8.421044 8.546226 5.403918 2.286660 6.319740
[4,] 3.960100 8.538932 2.210829 2.437113 5.324722
[5,] 1.333846 4.508688 7.411279 5.653044 7.248775
```

As with Sets, the primary advantage of the `setRecords` method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (data frame). Users that require higher performance will want to directly pass the Container a reference to a valid data frame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process. In this section, we walk the user through an example of how to set records directly.

Example #4 - Correctly set records (directly)

```
library(gamstransfer)
df = data.frame(h_1 = paste0("h", 1:8760), m_2 = paste0("m", 1:60),
s_3 = paste0("s", 1:60))
df$value = runif(nrow(df), min = 0, max = 100)
m = Container$new()
hrs = Set$new(m, "h", records = unique(df$h_1))
mins = Set$new(m, "m", records = unique(df$m_2))
secs = Set$new(m, "s", records = unique(df$s_3))
df$h_1 = factor(df$h_1, ordered = TRUE)
df$m_2 = factor(df$m_2, ordered = TRUE)
df$s_3 = factor(df$s_3, ordered = TRUE)
a = Parameter$new(m, "a", c(hrs, mins, secs))
# set records
a$records = df
> a$isValid()
[1] TRUE
```

In this example, we create a large parameter (31,536,000 records and 8880 unique domain elements. We mimic data that is labeled for every second in one year) and assign it to a parameter with `a$records`. GAMS Transfer R requires that all domain columns must be ordered factors. The `records` setter function does very little work other than checking if the object being set is a data frame. This places more responsibility on the user to create a data frame that complies with the standard format. In Example #1, we take care to properly reference the factor from the domain sets, and ensure that the symbol `a` is valid with `a$isValid() = TRUE`.

Users will need to use the `$isValid(verbose=TRUE)` method to debug any structural issues. As an example, we incorrectly generate categorical data types by passing the data frame constructor the generic `factor` argument. This creates factor columns, but they are not ordered and they do not reference the underlying domain set. These errors result in `a` being invalid.

Example #5 - Incorrectly set records (directly)

```
library(gamstransfer)
df = data.frame(h_1 = paste0("h", 1:8760), m_2 = paste0("m", 1:60),
s_3 = paste0("s", 1:60))
df$value = runif(nrow(df), min = 0, max = 100)
m = Container$new()
hrs = Set$new(m, "h", records = unique(df$h_1))
mins = Set$new(m, "m", records = unique(df$m_2))
secs = Set$new(m, "s", records = unique(df$s_3))
df$h_1 = factor(df$h_1)
df$m_2 = factor(df$m_2)
df$s_3 = factor(df$s_3)
a = Parameter$new(m, "a", c(hrs, mins, secs))
# set records
a$records = df
> a$isValid()
[1] FALSE
> a$isValid(verbose=TRUE)
Domain information in column h_1 must be an ORDERED factor
[1] FALSE
```

Note

One can also use the genereateRecords() method to automatically populate randomly generated symbol records in the standard format.

Variable

Variable Constructor

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the Container object that the symbol is being added to	Yes	-
`name`	`string`	Name of symbol	Yes	-
`type`	`string`	Type of variable being created [`binary`, `integer`, `positive`, `negative`, `free`, `sos1`, `sos2`, `semicont`, `semiint`]	No	`free`
`domain`	`list`, `vector`	List, vector of domains given either as a string (`"*"` for universe set) or as a reference to a Set object, an empty domain list will create a scalar variable	No	NULL
`records`	many	Symbol records	No	NULL
`domainForwarding`	`logical`	Flag or a vector of flags that forces variable elements to be recursively included in corresponding parent sets (i.e.,implicit set definition)	No	FALSE
`description`	`string`	Description of symbol	No	""

Variable Fields

Field	Description	Type
`container`	reference to the Container that the symbol belongs to	`Container`
`defaultValues`	default values for the symbol	`numeric` (named vector)

Field	Description	Type
description	description of symbol	string
dimension	dimension of symbol, setting dimension is a shorthand notation to set domain to a list of size n containing "*"	integer
domain	list of domains given either as string (* for universe set) or as reference to the Set/Alias object	list
domainForwarding	Flag or a vector of flags that forces variable elements to be recursively included in corresponding parent sets (i.e.,implicit set definition)	logical
domainLabels	column headings for the records data frame	list of string
domainNames	string version of domain names	list of string
domainType	none, relaxed or regular depending on state of domain links	string
isScalar	TRUE if self$dimension = 0	logical
name	name of symbol	string
numberRecords	number of symbol records (i.e., returns nrow(self$records) if not NULL)	integer
records	the main symbol records	data frame
shape	a vector describing the array dimensions if records were converted with $toDense()	vector
summary	output a list of only the metadata	list
type	string type of variable	list

Variable Methods

Method	Description	Arguments/Defaults	Returns
addUELs	adds UELs to the symbol	uels (character), dimension=NULL	NULL
copy	copies the symbol to the destination container. overwrite=TRUE overwrites the symbol with the same name in destination. Symbol domains are relaxed if destination Container does not contain equivalent domain sets	destination (Container), overwrite=FALSE	NULL
countDomainViolations	returns the number of domain violations for the symbol	-	numeric
countDuplicateRecords	returns the number of duplicate records for the symbol	-	numeric
countEps	total number of SpecialValues$EPS across all columns	columns = "level"	integer
countNA	total number of SpecialValues[["NA"]] across all columns	columns = "level"	integer

Method	Description	Arguments/Defaults	Returns
countNegInf	total number of SpecialValues$NEGINF across all columns	columns = "level"	integer
countPosInf	total number of SpecialValues$POSINF across all columns	columns = "level"	integer
countUndef	total number of SpecialValues$UNDEF across all columns	columns = "level"	integer
dropDomainViolations	drops domain violations for the symbol	-	-
dropDuplicateRecords	drops duplicate records for the Symbol. keep = "first" (drops all duplicates while keeping the first instance as unique), keep="last" (drops all duplicates while keeping the last instance as unique), or keep=FALSE (drops all duplicates)	keep = "first"	-
equals	Check if two Symbol objects are equal	other (Symbol), columns=NULL (character) (if NULL all columns are compared), checkUELs=TRUE, checkMetaData=TRUE, rtol=0 (relative tolernace), atol=0 (absolute tolerance), verbose=FALSE	logical
findDomainViolations	get a view of the records data frame that contain any domain violations	-	data.frame
findDuplicateRecords	get a view of records data frame that contain duplicate records. keep = "first" (finds all duplicates while keeping the first instance as unique), keep="last" (finds all duplicates while keeping the last instance as unique), or keep=FALSE (finds all duplicates)	keep= first	data.frame

Method	Description	Arguments/Defaults	Returns
generateRecords	convenience method to set standard data.frame formatted records. Will generate records with the Cartesian product of all domain sets. The density argument can take any value on the interval [0,1]. If density is < 1, then randomly selected records will be removed. density will accept a numeric of length 1 or dimension. This allows users to specify a density per symbol dimension (when vector) or the density of records dataframe. Random numbers can be generated by passing a user-defined function func(size) to func argument (runif() by default). Random number state can be set with seed argument.	density=1.0 (numeric) func=runif() (named list of functions, function, NULL) seed=NULL (integer)	NULL
getDomainViolations	gets domain violations for the symbol	-	list of DomainViolation
getMaxValue	get the maximum value across all columns	columns = "level"	numeric
getMaxAbsValue	get the maximum absolute value across all columns	columns = "level"	numeric
getMinValue	get the minimum value across all columns	columns = "level"	numeric
getMeanValue	get the mean value across all columns	columns = "level"	numeric
getSparsity	get the sparsity of the symbol w.r.t the size of full cartesian product of the domain sets	-	numeric
getUELs	returns the UELs used in the Symbol	dimension = NULL, codes = NULL, ignoreUnused=FALSE	character vector
hasDomainViolations	returns TRUE if the symbol contains domain violations, FALSE otherwise	-	logical
hasDuplicateRecords	returns TRUE if the symbol contains duplicate records, FALSE otherwise	-	logical

Method	Description	Arguments/Defaults	Returns
`isValid`	checks if the symbol is in a valid format, throw exceptions if `verbose=TRUE`, recheck a symbol if `force=TRUE`	`verbose=FALSE` `force=FALSE`	`logical`
`removeUELs`	removes UELs from the symbol	`uels=NULL` (character), `dimension=NULL`	`NULL`
`renameUELs`	renames UELs in the symbol	`uels`(character of same length as current UELs or named character vector), `dimension=NULL`, `allowMerge=FALSE` (logical). If `allowMerge = TRUE`, the underlying integer mapping of a factor is allowed to change to offer additional data flexibility	`NULL`
`reorderUELs`	reorders UELs in the symbol that appear in the symbol `dimensions`. If `uels` is `NULL`, the UELs are reordered based on symbol records, unused UELs are moved to the end. If `dimensions` is `NULL` then reorder UELs in all dimensions of the symbol	`uels=NULL`(`NULL` or character of same length as current UELs or named character vector), `dimension=NULL` (numeric)	`NULL`
`setRecords`	main convenience method to set standard `data frame` records	`records` (many types)	`NULL`
`setUELs`	sets UELs for the Symbol	`uels`(character), `dimension`, `rename=FALSE`	`NULL`
`toDense`	convert symbol to a dense `matrix` or `array` format	column = `level`	`array` or `matrix`
`whereMax`	find the row number in records data frame with a maximum value (return the first instance only)	`column = "level"`	`integer`
`whereMaxAbs`	find the row number in records data frame with a maximum absolute value (return the first instance only)	`column = "level"`	`integer`

Method	Description	Arguments/Defaults	Returns
`whereMin`	find the row number in records data frame with a minimum value (return the first instance only)	`column = "level"`	`integer`

Adding Variable Records

Three possibilities exist to assign symbol records to a variable: We show a few examples of ways to create differently structured variables:

Example #1 - Create a GAMS scalar variable

```
library(gamstransfer)
m = Container$new()
pi = Variable$new(m, "pi", records = data.frame(level = 3.14159))
# NOTE: the above syntax is equivalent to -
# pi = Variable$new(m, "pi", "free")
# pi$setRecords(data.frame(level = 3.14159))
# NOTE: the above syntax is also equivalent to -
# m$addVariable("pi", "free", records=data.frame(level = 3.14159))
> pi$records
    level
1 3.14159
```

Example #2 - Create a 2D positive variable, specifying no numerical data

```
library(gamstransfer)
m = Container$new()
v = Variable$new(m, "v", "positive", c("*", "*"), records =
data.frame(from=c("seattle", "chicago"), to=c("san-diego", c("madison"))))
> v$records
     from        to
1 seattle san-diego
2 chicago   madison
```

Example #3 - Create a 2D variable (defined over a set) from a matrix

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", "*", records = paste0("i", 1:5))
j = Set$new(m, "j", "*", records = paste0("j", 1:5))
# creating records for parameter a
ij = list(i_1 = paste0("i", 1:5), j_2 = paste0("j", 1:5))
df = rev(expand.grid(rev(ij)))
df$value = 1:25
a = Parameter$new(m, "a", c(i, j), records = df)
# create a free variable and set the level and marginal attributes from matricies
v = Variable$new(m, "v", domain = c(i, j), records = list(level = a$toDense(), marginal = a$toDense()))
# if not specified, the toDense() method will convert the level values to a matrix
> v$toDense()
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
[5,]   21   22   23   24   25
```

As with Sets, the primary advantage of the `setRecords` method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (data frame). Users that require higher performance will want to directly pass the Container a reference to a valid data frame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process. In this section, we walk the user through an example of how to set records directly.

Example #4 - Correctly set records (directly)

```
library(gamstransfer)
df = data.frame(h_1 = paste0("h", 1:8760), m_2 = paste0("m", 1:60),
s_3 = paste0("s", 1:60))
df$level = runif(nrow(df), min = 0, max = 100)
df$marginal = 0.0
df$lower = SpecialValues$NEGINF
df$upper = SpecialValues$POSINF
df$scale = 1.0
m = Container$new()
hrs = Set$new(m, "h", records = unique(df$h_1))
mins = Set$new(m, "m", records = unique(df$m_2))
secs = Set$new(m, "s", records = unique(df$s_3))
df$h_1 = factor(df$h_1, ordered = TRUE)
df$m_2 = factor(df$m_2, ordered = TRUE)
df$s_3 = factor(df$s_3, ordered = TRUE)
a = Variable$new(m, "a", domain = c(hrs, mins, secs))
# set records
a$records = df
> a$isValid()
[1] TRUE
```

In this example, we create a large variable (31,536,000 records and 8880 unique domain elements. We mimic data that is labeled for every second in one year) and assign it to a variable with `a$records`. GAMS Transfer R requires that all domain columns must be ordered factors. The `records` setter function does very little work other than checking if the object being set is a data frame. This places more responsibility on the user to create a data frame that complies with the standard format. In Example #1, we take care to properly reference the factor from the domain sets and ensure that the symbol `a` is valid with `a$isValid() = TRUE`.

As with Set and Parameters, users can use the `$isValid(verbose=TRUE)` method to debug any structural issues.

Note

One can also use the genereateRecords() method to automatically populate randomly generated symbol records in the standard format.

Equation

Equation Constructor

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the Container object that the symbol is being added to	Yes	-
`name`	`string`	Name of symbol	Yes	-
`type`	`string`	Type of equation being created [`eq` (or `E`/`e`), `geq` (or `G`/`g`), `leq` (or `L`/`l`), `nonbinding` (or `N`/`n`), `external` (or `X`/`x`)]	No	`free`
`domain`	`list`, `vector`	List, vector of domains given either as a string (`"*"` for universe set) or as a reference to a Set object, an empty domain list will create a scalar equation	No	NULL
`records`	many	Symbol records	No	NULL
`domainForwarding`	`logical`	Flag or a vector of flags that forces equation elements to be recursively included in corresponding parent sets (i.e.,implicit set definition)	No	FALSE
`description`	`string`	Description of symbol	No	""

Equation Fields

Field	Description	Type
`container`	reference to the Container that the symbol belongs to	`Container`
`defaultValues`	default values for the symbol	`numeric` (named vector)
`description`	description of symbol	`string`
`dimension`	dimension of symbol, setting dimension is a shorthand notation to set `domain` to a list of size n containing "$*$"	`integer`
`domain`	list of domains given either as string ($*$ for universe set) or as reference to the Set/Alias object	`list`
`domainForwarding`	Flag or a vector of flags that forces equation elements to be recursively included in corresponding parent sets (i.e.,implicit set definition)	`logical`
`domainLabels`	column headings for the `records` data frame	`list` of `string`
`domainNames`	string version of domain names	`list` of `string`
`domainType`	`none`, `relaxed` or `regular` depending on state of domain links	`string`
`isScalar`	TRUE if `self$dimension = 0`	`logical`
`name`	name of symbol	`string`
`numberRecords`	number of symbol records (i.e., returns `nrow(self$records)` if not `NULL`)	`integer`
`records`	the main symbol records	`data frame`
`shape`	a vector describing the array dimensions if `records` were converted with `$toDense()`	`vector`
`summary`	output a `list` of only the metadata	`list`
`type`	`string` type of equation	`list`

Equation Methods

Method	Description	Arguments/Defaults	Returns
`addUELs`	adds UELs to the symbol	`uels` (character), `dimension=NULL`	`NULL`
`copy`	copies the symbol to the `destination` container. `overwrite=TRUE` overwrites the symbol with the same name in `destination`. Symbol domains are `relaxed` if `destination` Container does not contain equivalent domain sets	`destination` (Container), `overwrite=FALSE`	`NULL`
`countDomainViolations`	returns the number of domain violations for the symbol	-	`numeric`

Method	Description	Arguments/Defaults	Returns
`countDuplicateRecords`	returns the number of duplicate records for the symbol	-	`numeric`
`countEps`	total number of `SpecialValues$EPS` across all `columns`	`columns = "level"`	`integer`
`countNA`	total number of `SpecialValues[["NA"]]` across all `columns`	`columns = "level"`	`integer`
`countNegInf`	total number of `SpecialValues$NEGINF` across all `columns`	`columns = "level"`	`integer`
`countPosInf`	total number of `SpecialValues$POSINF` across all `columns`	`columns = "level"`	`integer`
`countUndef`	total number of `SpecialValues$UNDEF` across all `columns`	`columns = "level"`	`integer`
`dropDomainViolations`	drops domain violations for the symbol	-	-
`dropDuplicateRecords`	drops duplicate records for the Symbol. `keep = "first"` (drops all duplicates while keeping the first instance as unique), `keep="last"` (drops all duplicates while keeping the last instance as unique), or `keep=FALSE` (drops all duplicates)	`keep = "first"`	-
`equals`	Check if two Symbol objects are equal	`other` (`Symbol`), `columns=NULL` (character) (if `NULL` all columns are compared), `checkUELs=TRUE`, `checkMetaData=TRUE`, `rtol=0` (relative tolernace), `atol=0` (absolute tolerance), `verbose=FALSE`	logical
`findDomainViolations`	get a view of records data frame that contain any domain violations	-	`data.frame`

Method	Description	Arguments/Defaults	Returns
`findDuplicateRecords`	get a view of records data frame that contain duplicate records. `keep = "first"` (finds all duplicates while keeping the first instance as unique), `keep="last"` (finds all duplicates while keeping the last instance as unique), or `keep=FALSE` (finds all duplicates)	`keep= first`	`data.frame`
`generateRecords`	convenience method to set standard `data.frame` formatted records. Will generate records with the Cartesian product of all domain sets. The `density` argument can take any value on the interval `[0,1]`. If `density` is < 1, then randomly selected records will be removed. `density` will accept a `numeric` of length 1 or `dimension`. This allows users to specify a density per symbol dimension (when vector) or the density of `records` dataframe. Random numbers can be generated by passing a user-defined function `func(size)` to `func` argument (`runif()` by default). Random number state can be set with `seed` argument.	`density=1.0` (`numeric`) `func=runif()` (named list of functions, function, NULL) `seed=NULL` (`integer`)	`NULL`
`getDomainViolations`	gets domain violations for the symbol	-	list of `DomainViolation`
`getMaxValue`	get the maximum value across all `columns`	`columns = "level"`	`numeric`
`getMaxAbsValue`	get the maximum absolute value across all `columns`	`columns = "level"`	`numeric`
`getMinValue`	get the minimum value iacross all `columns`	`columns = "level"`	`numeric`
`getMeanValue`	get the mean value across all `columns`	`columns = "level"`	`numeric`

Method	Description	Arguments/Defaults	Returns
`getSparsity`	get the sparsity of the symbol w.r.t the size of full cartesian product of the domain sets	-	`numeric`
`getUELs`	returns the UELs used in the Symbol	`dimension = NULL`, `codes = NULL`, `ignoreUnused=FALSE`	`character vector`
`hasDomainViolations`	returns TRUE if the symbol contains domain violations, FALSE otherwise	-	`logical`
`hasDuplicateRecords`	returns TRUE if the symbol contains duplicate records, FALSE otherwise	-	`logical`
`isValid`	checks if the symbol is in a valid format, throw exceptions if `verbose=TRUE`, recheck a symbol if `force=TRUE`	`verbose=FALSE` `force=FALSE`	`logical`
`removeUELs`	removes UELs from the symbol	`uels=NULL` (character), `dimension=NULL`	`NULL`
`renameUELs`	renames UELs in the symbol	`uels`(character of same length as current UELs or named character vector), `dimension=NULL`, `allowMerge=FALSE` (logical). If `allowMerge = TRUE`, the underlying integer mapping of a factor is allowed to change to offer additional data flexibility	`NULL`
`reorderUELs`	reorders UELs in the symbol that appear in the symbol `dimensions`. If `uels` is `NULL`, the UELs are reordered based on symbol records, unused UELs are moved to the end. If `dimensions` is `NULL` then reorder UELs in all dimensions of the symbol	`uels=NULL`(`NULL` or character of same length as current UELs or named character vector), `dimension=NULL` (numeric)	`NULL`
`setRecords`	main convenience method to set standard `data frame` records	`records` (many types)	`NULL`
`setUELs`	sets UELs for the Symbol	`uels`(character), `dimension`, `rename=FALSE`	`NULL`
`toDense`	convert symbol to a dense `matrix` or `array` format	`column = "value"`	`array` or `matrix`

Method	Description	Arguments/Defaults	Returns
whereMax	find the row number in records data frame with a maximum value (return the first instance only)	column="level"	integer
whereMaxAbs	find the row number in records data frame with a maximum absolute value (return the first instance only)	column="level"	integer
whereMin	find the row number in records data frame with a minimum value (return the first instance only)	column="level"	integer

Adding Equation Records

Three possibilities exist to assign symbol records to an equation: We show a few examples of ways to create differently structured equations:

Example #1 - Create a GAMS scalar equation

```
library(gamstransfer)
m = Container$new()
pi = Equation$new(m, "pi", type="eq", records = data.frame(level = 3.14159))
# NOTE: the above syntax is equivalent to -
# pi = Equation$new(m, "pi", type="eq")
# pi$setRecords(data.frame(level = 3.14159))
# NOTE: the above syntax is also equivalent to -
# m$addEquation("pi", type="eq", records=data.frame(level = 3.14159))
> pi$records
    level
1 3.14159
```

Example #2 - Create a 2D positive equation, specifying no numerical data

```
library(gamstransfer)
m = Container$new()
e = Equation$new(m, "e", "eq", c("*", "*"), records =
data.frame(from=c("seattle", "chicago"), to=c("san-diego", c("madison"))))
> e$records
     from        to
1 seattle san-diego
2 chicago   madison
```

Example #3 - Create a 2D equation (defined over a set) from a matrix

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", "*", records = paste0("i", 1:5))
j = Set$new(m, "j", "*", records = paste0("j", 1:5))
# creating records for parameter a
ij = list(i_1 = paste0("i", 1:5), j_2 = paste0("j", 1:5))
df = rev(expand.grid(rev(ij)))
df$value = 1:25
a = Parameter$new(m, "a", c(i, j), records = df)
# create a free variable and set the level and marginal attributes from matrices
e = Equation$new(m, "e", "nonbinding", domain = c(i, j), records = list(level = a$toDense(), marginal = a$toDense()))
> e$records
    i  j level marginal
1  i1 j1     1        1
2  i2 j1     6        6
3  i3 j1    11       11
4  i4 j1    16       16
5  i5 j1    21       21
6  i1 j2     2        2
7  i2 j2     7        7
8  i3 j2    12       12
9  i4 j2    17       17
10 i5 j2    22       22
11 i1 j3     3        3
12 i2 j3     8        8
13 i3 j3    13       13
14 i4 j3    18       18
15 i5 j3    23       23
16 i1 j4     4        4
17 i2 j4     9        9
18 i3 j4    14       14
19 i4 j4    19       19
20 i5 j4    24       24
21 i1 j5     5        5
22 i2 j5    10       10
23 i3 j5    15       15
24 i4 j5    20       20
25 i5 j5    25       25
# if not specified, the toDense() method will convert the level values to a matrix
> e$toDense()
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
[5,]   21   22   23   24   25
```

As with sets, parameters, and variables the primary advantage of the `setRecords` method is that GAMS Transfer will convert many different (and convenient) data types into the standard data format (data frame). Users that require higher performance will want to directly pass the Container a reference to a valid data frame, thereby skipping some of these computational steps. This places more burden on the user to pass the data in a valid standard form, but it speeds the records setting process. In this section, we walk the user through an example of how to set records directly.

Example #4 - Correctly set records (directly)

```
library(gamstransfer)
df = data.frame(h_1 = paste0("h", 1:8760), m_2 = paste0("m", 1:60),
s_3 = paste0("s", 1:60))
df$level = runif(nrow(df), min = 0, max = 100)
df$marginal = 0.0
df$lower = SpecialValues$NEGINF
df$upper = SpecialValues$POSINF
df$scale = 1.0
m = Container$new()
hrs = Set$new(m, "h", records = unique(df$h_1))
mins = Set$new(m, "m", records = unique(df$m_2))
secs = Set$new(m, "s", records = unique(df$s_3))
df$h_1 = factor(df$h_1, ordered = TRUE)
df$m_2 = factor(df$m_2, ordered = TRUE)
df$s_3 = factor(df$s_3, ordered = TRUE)
a = Equation$new(m, "a", "eq", domain = c(hrs, mins, secs))
# set records
a$records = df
> a$isValid()
[1] TRUE
```

In this example, we create a large equation (31,536,000 records and 8880 unique domain elements. We mimic data that is labeled for every second in one year) and assign it to an equation with `a$records`.

GAMS Transfer R requires that all domain columns must be ordered factors. The `records` setter function does very little work other than checking if the object being set is a data frame. This places more responsibility on the user to create a data frame that complies with the standard format. In Example #1, we take care to properly reference the factor from the domain sets and ensure that the symbol `a` is valid in the end with `a$isValid() = TRUE`.

As with sets, parameters, and variables, users can use the `$isValid(verbose=TRUE)` method to debug any structural issues.

Note

One can also use the genereateRecords() method to automatically populate randomly generated symbol records in the standard format.

Alias

Alias Constructor

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the Container object that the symbol is being added to	Yes	-
`name`	`string`	Name of symbol	Yes	-
`aliasWith`	`Set` object	set object to create an alias for	Yes	-

Example - Creating an alias from a set

GAMS Transfer R only stores the reference to the parent set as part of the alias structure. Most properties that are called from an alias object simply point to the properties of the parent set (with the exception of `container`, `name`, and `aliasWith`). It is possible to create an alias from another alias object. In this case, a recursive search will be performed to find the root parent set. This is the set that will ultimately be stored as the `aliasWith` field. We can see this behavior in the following example:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = paste0("i", 1:5))
ip = Alias$new(m, "ip", i)
ipp = Alias$new(m, "ipp", ip)
> ip$aliasWith$name
[1] "i"
> ipp$aliasWith$name
[1] "i"
```

Alias Fields

Field	Description	Type
`aliasWith`	aliased object	`Set`
`description`	description of symbol	`string`
`dimension`	dimension of symbol, setting dimension is a shorthand notation to set `domain` to a list of size n containing "∗"	`integer`

Field	Description	Type
`domainForwarding`	Flag or a vector of flags that forces Alias elements to be recursively included in corresponding parent sets (i.e.,implicit set definition)	`logical`
`domainLabels`	column headings for the `records` data frame	`list` of `string`
`domainNames`	string version of domain names	`list` of `string`
`domainType`	`none`, `relaxed` or `regular` depending on state of domain links	`string`
`isSingleton`	`logical` if symbol is a singleton set	`logical`
`name`	name of symbol	`string`
`numberRecords`	number of symbol records (i.e., returns `nrow(self$records)` if not `NULL`)	`integer`
`records`	the main symbol records	`data frame`
`container`	reference to the Container that the symbol belongs to	`Container`
`summary`	output a `list` of only the metadata	`list`

Alias Methods

Method	Description	Arguments/Defaults	Returns
`addUELs`	adds UELs to the symbol	`uels` (character), `dimension=NULL`	`NULL`
`copy`	copies the `Alias` and the parent `Set` to the `destination` container. `overwrite=TRUE` overwrites the symbol with the same name in `destination`. Symbol domains are `relaxed` if `destination` Container does not contain equivalent domain sets	`destination` (Container), `overwrite=FALSE`	`NULL`
`countDomainViolations`	returns the number of domain violations for the symbol	-	`numeric`
`countDuplicateRecords`	returns the number of duplicate records for the symbol	-	`numeric`
`dropDomainViolations`	drops domain violations for the symbol	-	-
`dropDuplicateRecords`	drops duplicate records for the Symbol. `keep = "first"` (drops all duplicates while keeping the first instance as unique), `keep="last"` (drops all duplicates while keeping the last instance as unique), or `keep=FALSE` (drops all duplicates)	`keep = "first"`	-

Method	Description	Arguments/Defaults	Returns
equals	Check if two Symbol objects are equal	other (Symbol), checkUELs=TRUE, checkElementText=TRUE, checkMetaData=TRUE, verbose=FALSE	logical
findDomainViolations	get a view of records data frame that contain any domain violations	-	data.frame
findDuplicateRecords	get a view of records data frame that contain duplicate records. keep = "first" (finds all duplicates while keeping the first instance as unique), keep="last" (finds all duplicates while keeping the last instance as unique), or keep=FALSE (finds all duplicates)	keep= first	data.frame
generateRecords	convenience method to set standard data.frame formatted records. Will generate records with the Cartesian product of all domain sets. The density argument can take any value on the interval [0,1]. If density is < 1, then randomly selected records will be removed. density will accept a numeric of length 1 or dimension. This allows users to specify a density per symbol dimension (when vector) or the density of records dataframe. Random number state can be set with seed argument.	density=1.0 (numeric) seed=NULL (integer)	NULL
getDomainViolations	gets domain violations for the symbol	-	list of DomainViolation
getSparsity	get the sparsity of the symbol w.r.t the size of full cartesian product of the domain sets	-	numeric
getUELs	returns the UELs used in the Symbol	dimension = NULL, codes = NULL, ignoreUnused=FALSE	character vector

Method	Description	Arguments/Defaults	Returns
`hasDomainViolations`	returns TRUE if the symbol contains domain violations, FALSE otherwise	-	`logical`
`hasDuplicateRecords`	returns TRUE if the symbol contains duplicate records, FALSE otherwise	-	`logical`
`isValid`	checks if the symbol is in a valid format, throw exceptions if `verbose=TRUE`, recheck a symbol if `force=TRUE`	`verbose=FALSE` `force=FALSE`	`logical`
`removeUELs`	removes UELs from the symbol	`uels=NULL` (character), `dimension=NULL`	`NULL`
`renameUELs`	renames UELs in the symbol	`uels`(character of same length as current UELs or named character vector), `dimension=NULL`, `allowMerge=FALSE` (logical). If `allowMerge = TRUE`, the underlying integer mapping of a factor is allowed to change to offer additional data flexibility	`NULL`
`reorderUELs`	reorders UELs in the parent set that appear in the parent set `dimensions`. If `uels` is `NULL`, the UELs are reordered based on symbol records, unused UELs are moved to the end. If `dimensions` is `NULL` then reorder UELs in all dimensions of the symbol	`uels=NULL`(`NULL` or character of same length as current UELs or named character vector), `dimension=NULL` (numeric)	`NULL`
`setRecords`	main convenience method to set standard `data frame` formatted records	`records` (string vector, list, data frame)	`NULL`
`setUELs`	sets UELs for the Symbol	`uels`(character), `dimension`, `rename=FALSE`	`NULL`

Adding Alias Records

The linked structure of Aliases offers some unique opportunities to access some of the setter functionality of the parent set. Specifically, GAMS Transfer allows the user to change the `domain`, `description`, `dimension`, and `records` of the underlying parent set as a shorthand notation. We can see this behavior if we look at a modified Example #1 from adding set records.

Example - Creating set records through an alias link

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i")
ip = Alias$new(m, "ip", i)
ip$description = "adding new descriptive set text"
ip$domain = c("*", "*")
ij = list(paste0("i", 1:3), paste0("j", 1:3))
ip_recs = rev(expand.grid(rev(ij)))
colnames(ip_recs) = c("i", "j")
ip$setRecords(ip_recs)
> i$domain
[1] "*" "*"
> i$records
   i  j
1 i1 j1
2 i1 j2
3 i1 j3
4 i2 j1
5 i2 j2
6 i2 j3
7 i3 j1
8 i3 j2
9 i3 j3
```

Note

An alias `$isValid()=TRUE` when the underlying parent set is also valid. If the parent set is removed from the Container the alias will no longer be valid.

One can also use the genereateRecords() method to automatically populate randomly generated symbol records in the standard format.

UniverseAlias

UniverseAlias Constructor

Argument	Type	Description	Required	Default
`container`	`Container`	A reference to the Container object that the symbol is being added to	Yes	-
`name`	`string`	Name of symbol	Yes	-

Example - Creating an alias for the Universe

In GAMS it is possible to create aliases to the universe (i.e., the entire list of UELs) with the syntax:

```
set i / i1, i2 /;
alias(h,*);
set j / j1, j2 /;
```

GAMS Transfer R allows creating aliases to the universe set.

In this small example, `h` would be associated with all four UELs (`i1`, `i2`, `j1` and `j2`) even though set `j` was defined after the alias declaration. GAMS Transfer mimics this behavior with the `UniverseAlias` class. Internally, the `records` attribute will always call the <`Container`>`$getUELs()`. The `UniverseAlias` class is fundamentally different from the `Alias` class because it does not point to a parent set object. It is not possible to perform operations (like `setRecords` or `findDomainViolations`) on the parent set

through a `UniverseAlias` (because there is no parent set object). This means that a `UniverseAlias` can be created by only defining the symbol name. We can see this behavior in the following example:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = c("i1", "i2"))
h = UniverseAlias$new(m, "h")
j = Set$new(m, "j", records = c("j1","j2"))
> m$listSymbols()
[1] "i" "h" "j"
> m["h"]
GAMS Transfer: R6 object of class UniverseAlias.
    Use h$summary for details
> h$records
 uni
1 i1
2 i2
3 j1
4 j2
```

Note

Unlike other sets, the universe does not hold on to set `element_text`, thus the returned `data frame` for the `UniverseAlias` will only have one column.

UniverseAlias Fields

Field	Description	Type
`aliasWith`	always `"*"`	`string`
`description`	always `Aliased with *`	`string`
`dimension`	always `1`	`integer`
`domainLabels`	always `"uni"`	`character vector`
`domainNames`	always `"*"`	`character vector`
`domainType`	always `none`	`string`
`isSingleton`	always `FALSE`	`logical`
`name`	name of symbol	`string`
`numberRecords`	number of symbol records (i.e., returns `nrow(self$records)` if not `NULL`)	`integer`
`records`	the main symbol records	`data frame`
`container`	reference to the Container that the symbol belongs to	`Container`
`summary`	output a `list` of only the metadata	`list`

UniverseAlias Methods

Method	Description	Arguments/Defaults	Returns
`equals`	Check if two Symbol objects are equal	`other` (`Symbol`), `checkMetaData=TRUE`, `verbose=FALSE`	logical
`copy`	copies the symbol to the `destination` container. `overwrite=TRUE` overwrites the symbol with the same name in `destination`.	`destination` (Container), `overwrite=FALSE`	`NULL`

Method	Description	Arguments/Defaults	Returns
getSparsity	always 0	-	numeric
getUELs	returns the UELs from the Container. Returns only UELs in the data if ignoreUnused=TRUE, otherwise returns all UELs	ignoreUnused=FALSE	character vector
isValid	checks if the symbol is in a valid format, throw exceptions if verbose=TRUE, recheck a symbol if force=TRUE	verbose=FALSE force=FALSE	logical

7.7.1.3 Additional Features

Validating Data

GAMS Transfer R requires that the records for all symbols exist in a standard format (Standard Data Formats) in order for them to be understood by the Container. It is possible that the data could end up in a state that is inconsistent with the standard format (especially if setting symbol attributes directly). GAMS Transfer R includes the $isValid() method in order to determine if a symbol is valid. This method returns a logical. This method does not guarantee that a symbol will be successfully written to GDX. Other data errors (duplicate records, long UEL names, or domain violations) could exist that are not tested in $isValid(). For example, we create two valid sets and then check them with $isValid() to be sure.

It is possible to run $isValid() on both the Container as well as the symbol object. The Container method $isValid() will also return FALSE if there are any invalid symbols in the Container object. In addition, the Container$isValid() method also detects broken Container references in symbols and inconsistent symbol naming between the Container$data field and the symbol objects.

Example (valid data)

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = c("seattle", "san-diego", "washington_dc"))
j = Set$new(m, "j", i, records = c("san-diego", "washington_dc"))
> i$isValid()
[1] TRUE
> j$isValid()
[1] TRUE
> m$isValid()
[1] TRUE
```

Now we create some data that is invalid due to incorrect column names in the records for set j.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = c("seattle", "san-diego", "washington_dc"))
j = Set$new(m, "j", i)
j$records = data.frame(cities=c("grayslake", "washington_dc"))
> i$isValid()
[1] TRUE
> j$isValid()
[1] FALSE
> m$isValid()
[1] FALSE
```

In this example, we use records field of the symbol j to set the records. As mentioned in setting the records, when setting records directly, users must adhere to Standard Data Formats. In this example, symbol j does not have the correct number of columns. Moreover, the column headings do not follow the convention (should be "i_1"). The user can get more detailed error reporting if the verbose argument is set to TRUE. For example:

```
> j$isValid(verbose=TRUE)
Symbol 'records' does not have the correct number of columns (<symbol dimension> + 1)
[1] FALSE
```

The $isValid() method checks:

- If the symbol belongs to a Container
- If all domain set symbols exist in the Container
- If all domain set symbols objects are valid
- If all domain sets are one dimensional and not singleton sets
- If records are a data frame (or `NULL`)
- If the records `domainLabels` are unique
- The shape of the records is congruent with the dimensionality of the symbol
- That all data columns are type `numeric`
- To make sure that all domain categories are type `string`

Note

Calling `$isValid()` too often may have a significant impact on the performance.

Comparing GAMS Transfer objects

Equivalence between two symbols or between two containers can be tested using `equals` method. Since the order of records in the `records` data frame is not important from a GDX point of view, `equals` method compares symbol `records` independently from the order in which they are stored in the records data frame. As this requires a merge operation over the domain columns, `equals` is a computationally expensive call.

Attention

We do not recommend using `equals` method inside large loops or when performance is critical. It is, however, very useful for data debugging.

A quick example shows the syntax of `equals`:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i",1:5), description="set i")
j = Set$new(m, "j", records=paste0("i",1:5), description="set j")
> i$equals(j)
[1] FALSE
```

One can debug the reason for inequality using the option `verbose`.

```
> i$equals(j, verbose=TRUE)
Symbol names do not match i != j
[1] FALSE
```

By default, `equals` takes the strictest view of symbol "equality", i.e., everything must be equal. In this case, the symbol names and descriptions differ between the two sets `i` and `j`. We can relax this with a combination of argument flags. Comparing the two symbols again, but ignoring the meta data (i.e., ignoring the symbol name, description and type (if a Variable or Equation)):

```
> i$equals(j, checkMetaData=FALSE)
[1] TRUE
```

The `checkUELs` argument will ensure that the symbol "universe" is the same (in order and content) between two symbols, as illustrated in the following example:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
ip = Set$new(m, "ip", records=c("i1","i3","i2"))
> i$equals(ip, checkMetaData=FALSE)
[1] FALSE
> i$equals(ip, checkMetaData=FALSE, checkUELs=FALSE)
[1] TRUE
```

Numerical comparisons are enabled for `Parameters`, `Variables` and `Equations`. Equality can be flexibly defined through the `equals` method arguments. Again, the strictest view of equality is taken as the default behavior of `equals`.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
a = Parameter$new(m, "a", i, records=data.frame(c("i1","i2","i3"), c(1,2,3)))
ap = Parameter$new(m, "ap", i, records=data.frame(c("i1","i2","i3"), c(1+1e-9,2,3)))
> a$equals(ap, checkMetaData=FALSE)
[1] FALSE
> a$equals(ap, checkMetaData=FALSE, verbose=TRUE)
Symbol records contain numeric differences in the value attribute that are outside the specified tolerances
            rtol=0, atol=0
> a$equals(ap, checkMetaData=FALSE, atol=1e-8)
[1] TRUE
```

In the case of variables and equations, it is possible for the user to confine the numerical comparison to certain attributes (`level`, `marginal`, `lower`, `upper` and `scale`) by specifying the `columns` argument as the following example illustrates:

```
library(gamstransfer)
m = Container$new()
a = Variable$new(m, "a", "free", records=data.frame(level=100))
ap = Variable$new(m, "ap", "free", records=data.frame(level=101))
> a$records
  level
1   100
> ap$records
  level
1   101
> a$equals(ap, checkMetaData=FALSE)
[1] FALSE
> a$equals(ap, checkMetaData=FALSE, columns="level")
[1] FALSE
> a$equals(ap, checkMetaData=FALSE, columns="marginal")
[1] TRUE
```

Similar to symbols, one can compare two `Container` objects using the `equals` method. When comparing `Containers`, the `data` fields are compared and if the same symbol keys exist in the Containers under comparison, symbol `equals` method is used to compare the symbols. Here is a brief example:

```
> library(gamstransfer)
> m = Container$new()
> i = Set$new(m, "i")
> m1 = Container$new()
> i1 = Set$new(m1, "i")
> m$equals(m1)
[1] TRUE
> j = Set$new(m1, "j")
> m$equals(m1)
[1] FALSE
> m$equals(m1, verbose=TRUE)
Error in m$equals(m1, verbose = TRUE) :
  Containers contain different number of symbols.
 self: 1
 other :2
> k = Set$new(m, "k")
> m$equals(m1)
[1] FALSE
> m$equals(m1,verbose=TRUE)
Error in m$equals(m1, verbose = TRUE) :
  Container ‘data‘ field keys do not match. Keys not present in ‘other‘ :k
```

Domain Forwarding

GAMS includes the ability to define sets directly from data using the implicit set notation (see: Implicit Set Definition (or: Domain Defining Symbol Declarations)). This notation has an analogue in GAMS Transfer R called `domainForwarding`.

Note

It is possible to recursively update a subset tree in GAMS Transfer R.

Domain forwarding is available as an argument to all symbol object constructors; the user would simply need to pass `domainForwarding=TRUE` to forward domain across all dimensions or as a logical vector domainForwarding=c(TRUE, FALSE,....) to forward domain across selected dimensions.

In this example, we have raw data that is in the `dist` data frame, and we want to send the domain information into the `i` and `j` sets. We take care to pass the set objects as the domain for parameter `c`.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i")
j = Set$new(m, "j")
dist = data.frame(
    from = c("seattle", "seattle", "seattle",
    "san-diego", "san-diego", "san-diego"),
    to = c("new-york", "chicago", "topeka",
      "new-york", "chicago", "topeka"),
    thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
c = Parameter$new(m, "c", c(i, j), records = dist, domainForwarding = TRUE)
> i$records
        uni
1   seattle
2 san-diego
> j$records
       uni
1 new-york
2  chicago
3   topeka
> c$records
       from       to value
1   seattle new-york   2.5
2   seattle  chicago   1.7
3   seattle   topeka   1.8
4 san-diego new-york   2.5
5 san-diego  chicago   1.8
6 san-diego   topeka   1.4
```

Note

The element order in the sets `i` and `j` mirrors that in the raw data.

We can also selectively use domainForwarding for part of the domain by passing a logical vector to the `domainForwarding` argument as shown in the following example. The domain records are forwarded only to the domain set `i` but not to the domain set `j`.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i")
j = Set$new(m, "j")
dist = data.frame(
    from = c("seattle", "seattle", "seattle",
    "san-diego", "san-diego", "san-diego"),
    to = c("new-york", "chicago", "topeka",
      "new-york", "chicago", "topeka"),
    thousand_miles = c(2.5, 1.7, 1.8, 2.5, 1.8, 1.4)
)
c = Parameter$new(m, "c", c(i, j), records = dist, domainForwarding = c(TRUE, FALSE))
> i$records
        uni
1   seattle
4 san-diego
> j$records
NULL
> c$records
       from       to value
1   seattle new-york   2.5
2   seattle  chicago   1.7
3   seattle   topeka   1.8
4 san-diego new-york   2.5
5 san-diego  chicago   1.8
6 san-diego   topeka   1.4
```

In this example, we show that domain forwarding will also work recursively to update the entire set lineage. The domain forwarding occurs at the creation of every symbol object. The correct order of elements in set `i` is `(z, a, b, c)` because the records from `j` are forwarded first, and then the records from `k` are propagated through (back to `i`).

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i")
```

```
j = Set$new(m, "j", i, records = c("z"), domainForwarding = TRUE)
k = Set$new(m, "k", j, records = c("a", "b", "c"), domainForwarding = TRUE)
> i$records
    uni
1     z
2     a
3     b
4     c
> j$records
    i
1   z
2   a
3   b
4   c
> k$records
    j
1   a
2   b
3   c
```

Describing Data

The methods describeSets, describeParameters, describeVariables, and describeEquations allow the user to get a summary view of key data statistics. The returned data frame aggregates the output for a number of other methods (depending on symbol type). A description of each Container method is provided in the following subsections:

describeSets

Argument	Type	Description	Required	Default
`symbols`	`list`, `string`	A list of sets in the Container to include in the output. describeSets will include aliases if they are explicitly passed by the user.	No	`NULL` (if `NULL` specified, will assume all sets)

Returns: `data frame`

The following table includes a short description of the column headings in the return.

Field / Statistic	Description
`name`	name of the symbol
`isSingleton`	`logical` if the set is a singleton set
`domain`	domain labels for the symbol
`domainType`	`none`, `relaxed` or `regular` depending on the symbol state
`dimension`	dimension
`numberRecords`	number of records in the symbol
`sparsity`	`1 - numberRecs/dense` where `dense` is the size of full cartesian product of the domain sets

Example #1

```
library(gamstransfer)
m = Container$new("trnsport.gdx")
> m$describeSets()
  name isSingleton domain domainType dimension numberRecords sparsity
1    i       FALSE      *       none         1             2       NA
2    j       FALSE      *       none         1             3       NA
```

Example #2 – with aliases

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:10))
j = Set$new(m, "j", records=paste0("j", 1:10))
ip = Alias$new(m, "ip", i)
jp = Alias$new(m, "jp", j)
> m$describeSets()
  name isSingleton domain domainType dimension numberRecords sparsity
1    i       FALSE      *       none         1            10       NA
2    j       FALSE      *       none         1            10       NA
> m$describeSets(append(m$listSets(), m$listAliases()))
  name isSingleton domain domainType dimension numberRecords sparsity
1    i       FALSE      *       none         1            10       NA
3   ip       FALSE      *       none         1            10       NA
2    j       FALSE      *       none         1            10       NA
4   jp       FALSE      *       none         1            10       NA
```

describeParameters

Argument	Type	Description	Required	Default
`symbols`	`list`, `string`	A list of parameters in the Container to include in the output	No	`NULL` (if `NULL` specified, will assume all parameters)

Returns: `data frame`

The following table includes a short description of the column headings in the return.

Field / Statistic	Description
`name`	name of the symbol
`domain`	domain labels for the symbol
`domainType`	`none`, `relaxed` or `regular` depending on the symbol state
`dimension`	dimension
`numberRecords`	number of records in the symbol
`min`	min value in data
`mean`	mean value in data
`max`	max value in data
`whereMin`	row number min value (if multiple, returns only first occurrence)
`sparsity`	`1 - numRecs/dense` where where `dense` is the size of full cartesian product of the domain sets

```
library(gamstransfer)
m = Container$new("trnsport.gdx")
>  name domain domainType dimension numberRecords     min     mean     max
1    a      i    regular         1             2 350.000 475.0000 600.000
2    b      j    regular         1             3 275.000 300.0000 325.000
5    c    i j    regular         2             6   0.126   0.1755   0.225
3    d    i j    regular         2             6   1.400   1.9500   2.500
4    f     NA       none         0             1  90.000  90.0000  90.000
  whereMin whereMax sparsity
1        1        2        0
2        3        1        0
5        6        1        0
3        6        1        0
4        1        1       NA
```

describeVariables

Argument	Type	Description	Required	Default
symbols	list, string	A list of variables in the Container to include in the output	No	NULL (if NULL specified, will assume all variables)

Returns: data frame

The following table includes a short description of the column headings in the return.

Field / Statistic	Description
name	name of the symbol
type	type of variable (i.e., binary,integer,positive,negative,free,sos1,sos2,semicont,semiint)
domain	domain labels for the symbol
domainType	none, relaxed or regular depending on the symbol state
dimension	dimension
numberRecords	number of records in the symbol
sparsity	1 - numRecs/dense, where dense is the size of full cartesian product of the domain sets
minLevel	min value in the level column
meanLevel	mean value in the level column
maxLevel	max value in the level column
whereMaxAbsLevel	max absolute value in the level column

```
library(gamstransfer)
m = Container$new("trnsport.gdx")
> m$describeVariables()
  name     type domain domainType dimension numberRecords sparsity minLevel
1    x positive    i j    regular         2             6        0    0.000
2    z     free     NA       none         0             1       NA  153.675
  meanLevel maxLevel whereMaxAbsLevel
1   150.000  300.000                2
2   153.675  153.675                1
```

describeEquations

Argument	Type	Description	Required	Default
symbols	list, string	A list of equations in the Container to include in the output	No	NULL (if NULL specified, will assume all variables)

Returns: data frame

The following table includes a short description of the column headings in the return.

Field / Statistic	Description
name	name of the symbol
type	type of variable (i.e., binary,integer,positive,negative,free,sos1,sos2,semicont,semiint)
domain	domain labels for the symbol
domainType	none, relaxed or regular depending on the symbol state
dimension	dimension
numberRecords	number of records in the symbol

Field / Statistic	Description
sparsity	1 - numRecs/dense, where dense is the size of full cartesian product of the domain sets
minLevel	min value in the level column
meanLevel	mean value in the level column
maxLevel	max value in the level column
whereMaxAbsLevel	max absolute value in the level column

```
library(gamstransfer)
m = Container$new("trnsport.gdx")
> m$describeEquations()
    name type domain domainType dimension numberRecords sparsity minLevel
1   cost   eq     NA       none         0             1       NA        0
3 demand  geq      j    regular         1             3        0      275
2 supply  leq      i    regular         1             2        0      350
  meanLevel maxLevel whereMaxAbsLevel
1         0        0                1
3       300      325                1
2       450      550                2
```

describeAliases

Argument	Type	Description	Required	Default
symbols	list, string	A list of aliases in the Container to include in the output.	No	NULL (if NULL specified, will assume all aliases)

Returns: data frame

The following table includes a short description of the column headings in the return.

Field / Statistic	Description
name	name of the symbol
aliasWith	name of the parent set
isSingleton	logical if an alias of a singleton set
domain	domain labels for the symbol
domainType	none, relaxed or regular depending on the symbol state
dimension	dimension
numberRecords	number of records in the symbol
sparsity	1 - numberRecs/dense, where dense is the size of full cartesian product of the domain sets

Example #1

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = paste0("i",1:5))
j = Set$new(m, "j", records = paste0("j",1:10))
ip = Alias$new(m, "ip", i)
ipp = Alias$new(m, "ipp", ip)
jp = Alias$new(m, "jp", j)
> m$describeAliases()
  name aliasWith isSingleton domain domainType dimension numberRecords sparsity
1   ip         i       FALSE      *       none         1             5       NA
2  ipp         i       FALSE      *       none         1             5       NA
3   jp         j       FALSE      *       none         1            10       NA
```

Matrix Generation

GAMS Transfer R stores data in a "flat" format, that is, one record entry per data frame row. However, it is often necessary to convert this data format into a matrix/array format. GAMS Transfer R enables users to do this with relative ease using the `toDense` symbol methods This method will return a dense `N`-dimensional array (matrix for 2-Dimensions) with each dimension corresponding to the GAMS symbol dimension; it is possible to output an array up to 20 dimensions (a GAMS limit).

Example (1D data w/o domain linking (i.e., a relaxed domain))

```
library(gamstransfer)
m = Container$new()
a = Parameter$new(m, "a", "i", records = data.frame(uni=c("a","c"), element_text=c(1,3)))
> a$toDense()
[1] 1 3
```

Note that the parameter `a` is not linked to another symbol, so when converting to a matrix, the indexing is referenced to the data structure in `a$records`. Defining a sparse parameter `a` over a set `i` allows us to extract information from the `i` domain and construct a very different dense matrix, as the following example shows:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = c("a", "b", "c", "d"))
a = Parameter$new(m, "a", i, records = data.frame(c("a","c"), c(1,3)))
> a$toDense()
[1] 1 0 3 0
```

Example (2D data w/ domain linking)

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = c("a", "b", "c", "d"))
a = Parameter$new(m, "a", c(i, i), records =
data.frame(i=c("a","c"), i=c("a","c"), c(1,3)))
> i$records
    uni
1     a
2     b
3     c
4     d
> a$records
    i i.1 value
1   a   a     1
2   c   c     3
> a$toDense()
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    0    0    0
[3,]    0    0    3    0
[4,]    0    0    0    0
```

Note

If there are unused UELs in the domain symbol, `toDense()` requires that the unused UELs are at the end of UEL list. One can achieve this by calling the `reorderUELs()` method for the domain symbol. Similarly, if the symbol records are in a different order than that of domain symbol UEL, UELs should be reordered to follow the record order. This can also be achived using `reorderUELs()` symbol method.

The Universe Set

A Unique Element (UEL) is an `(i,s)` pair where `i` is an identification number for a string `s`. A list of all UELs is also known as 'the universe' or 'the universe set'. GAMS uses UELs to efficiently store domain entries of a record by storing the UEL ID `i` of a domain entry instead of the actual string `s`. This avoids storing the same string multiple times. The concept of UELs also exists in R and is called a "factor". GAMS Transfer R leverages these types in order to efficiently store strings and enable domain checking within the R environment.

Each domain column in a data frame can be a factor, the effect is that each symbol maintains its own list of UELs per dimension. R lets the user choose if levels in a factor should be regarded as ordered. GAMS Transfer R relies exclusively on `ordered` factors. By using ordered factors, GAMS Transfer R will order the UELs such that elements appear in the order in which they appeared in the data (which is how GAMS defines the UELs). `GAMSTransfer` allows the user to reorder the UELs with the `uelPriority` argument in the `$write()` method.

GAMS Transfer R does not keep track of the UELs separately from other symbols in the Container, it will be created internal to the `$write()` method and is based on the order in which data is added to the container. The user can access the current state of UELs with the `$getUELs()` container method. For example, we set a two dimensional set:

```
library(gamstransfer)
m = Container$new()
j = Set$new(m, "j", c("*", "*"), records = data.frame(i=c("i1","i2"), j=c("j1","j2")))
> j$records
      i     j
1    i1    j1
2    i2    j2
> m$getUELs()
[1] "i1" "i2" "j1" "j2"
```

Customize The Universe Set

GAMS Transfer R allows the users to customize the Universe Set with the help of Symbol UELs methods shown in the following table.

Method	Description	Arguments/Defaults	Returns
`addUELs`	adds UELs to the symbol	`uels` (character), `dimension=NULL`	`NULL`
`getUELs`	returns the UELs used in the Symbol	`dimension = NULL`, `codes = NULL`, `ignoreUnused=FALSE`	`character vector`
`removeUELs`	removes UELs from the symbol	`uels=NULL` (character), `dimension=NULL`	`NULL`
`renameUELs`	renames UELs in the symbol	`uels`(character of same length as current UELs or named character vector), `dimension=NULL`, `allowMerge=FALSE` (logical). If `allowMerge = TRUE`, the underlying integer mapping of a factor is allowed to change to offer additional data flexibility	`NULL`
`reorderUELs`	reorders UELs in the symbol that appear in the symbol `dimensions`. If `uels` is `NULL`, the UELs are reordered based on symbol records, unused UELs are moved to the end. If `dimensions` is `NULL`	`uels=NULL`(`NULL` or character of same length as current UELs or named character vector), `dimension=NULL` (numeric)	`NULL`
	then reorder UELs in all dimensions of the symbol		

Method	Description	Arguments/Defaults	Returns
`setUELs`	sets UELs for the Symbol	`uels`(character), `dimension`, `rename=FALSE`	`NULL`

Note

Symbols should be valid in order to use these methods.

when `dimension` argument is optional, the method is applied to all dimensions

Some of these methods are also available as Container methods as shown in the following table. Container methods internally call the corresponding Symbol methods.

Method	Description	Arguments/Defaults	Returns
`getUELs`	returns the UELs used in the Container	`symbols=NULL` (character), `ignoreUnused = FALSE`	`character vector`
`removeUELs`	removes `uels` from all symbols the Container	`uels=NULL` (character)	`NULL`
`renameUELs`	renames `uels` in the Container	`uels` (named character), `allowMerge=FALSE` (logical). If `allowMerge = TRUE`, the underlying integer mapping of a factor is allowed to change to offer additional data flexibility	`NULL`

getUELs Examples

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
j = Set$new(m, "j", records=c("j1","j2","j3"))
a = Parameter$new(m, "a", c(i, j), records=data.frame(paste0("i",1:4), paste0("j",1:4), 1:4))
> i$getUELs()
[1] "i1" "i2" "i3"
> m$getUELs()
[1] "i1" "i2" "i3" "j1" "j2" "j3" "i4" "j4"
> m$getUELs("j")
[1] "j1" "j2" "j3"
```

addUELs Examples

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
j = Set$new(m, "j", records=c("j1","j2","j3"))
a = Parameter$new(m, "a", c(i, j), records=data.frame(paste0("i",1:3), paste0("j",1:3), 1:3))
i$addUELs("ham")
a$addUELs("and", 1)
a$addUELs("cheese", 2)
> i$getUELs()
[1] "i1"  "i2"  "i3"  "ham"
> a$getUELs()
[1] "i1"     "i2"     "i3"     "and"    "j1"     "j2"     "j3"     "cheese"
```

In this example we have added three new (unused) UELs: `ham`, `and`, `cheese`. These three UELs will now appear in the GAMS universe set (accessible with `m$getUELs()`). The addition of unused UELs does not impact the validity of the symbols (i.e., unused UELs will not trigger domain violations).

removeUELs Examples

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
j = Set$new(m, "j", records=c("j1","j2","j3"))
a = Parameter$new(m, "a", c(i, j), records=data.frame(paste0("i",1:3), paste0("j",1:3), 1:3))
i$addUELs("ham")
a$addUELs("and", 1)
a$addUELs("cheese", 2)
# remove symbol UELs explicitly by dimension
i$removeUELs("ham", 1)
a$removeUELs("and", 1)
a$removeUELs(c("and", "cheese"), 2)
# remove symbol UELs for the entire symbol
i$removeUELs("ham")
a$removeUELs(c("and", "cheese"))
# remove ONLY unused UELs from each symbol, independently
i$removeUELs()
a$removeUELs()
# remove ONLY unused UELs from the entire container (all symbols)
m$removeUELs()
> m$getUELs()
[1] "i1" "i2" "i3" "j1" "j2" "j3"
```

If a user removes a UEL that appears in data, that data will be lost perminately. The domain label will be transformed into an NA as seen in this example:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
j = Set$new(m, "j", records=c("j1","j2","j3"))
a = Parameter$new(m, "a", c(i, j), records=data.frame(i=paste0("i",1:3), j=paste0("j",1:3), 1:3))
m$removeUELs("i1")
> i$records
   uni
1 <NA>
2   i2
3   i3
> a$records
     i  j value
1 <NA> j1     1
2   i2 j2     2
3   i3 j3     3
```

Attention

A container cannot be written if there are NA entries in any of the domain columns (in any symbol). An error is thrown if there are missing domain labels.

renameUELs Examples

`renameUELs` is a method of all GAMS `Symbol` classes as well as the `Container` class. This method allows the user to rename UELs in a symbol dimension(s), over several symbols, or over the entire container. This method is handy when attempting to harmonize labeling schemes between data structures that originated from different sources. For example:

```
library(gamstransfer)
m = Container$new()
a = Parameter$new(m, "a", c("*","*"),
records = data.frame(from=c("WI","IL","WI"),
to=c("IL", "IN", "IN"), quantity=c(10, 12.5, 8.7)),
description = "shipment quantities")
b = Parameter$new(m, "b", c("*"),
records = data.frame(state=c("wisconsin","illinois","indiana"),
c(1.2, 1.7, 1.2)), description = "multipliers")
```

results in the following records:

```
> a$records
   from    to value
1    WI    IL  10.0
2    IL    IN  12.5
3    WI    IN   8.7
> b$records
      state value
1 wisconsin   1.2
```

```
2  illinois   1.7
3   indiana   1.2
```

However, two different data sources were used to generate the parameters `a` and `b` – one data source used the uppercase postal abbreviation of the state name and the other source used a lowercase full state name as the uniqe identifier. With the following syntax the user can harmonize to a mixed case postal code labeling scheme (without losing any of the original UEL ordering).

```
m$renameUELs(c("WI"="Wi", "IL"="Il", "IN"="In",
"wisconsin"="Wi", "illinois"="Il","indiana"="In"))
```

This results in the following records and the universe set:

```
> a$records
   from    to value
1    Wi    Il  10.0
2    Il    In  12.5
3    Wi    In   8.7
> b$records
  state value
1    Wi   1.2
2    Il   1.7
> m$getUELs()
[1] "Wi" "Il" "In"
```

reorderUELs Examples

`reorderUELs` is a method of all GAMS symbol classes. This method allows the user to reorder UELs of a specific symbol dimension. `reorderUELs` will not add/remove any UELs. For example:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
> i$getUELs()
[1] "i1" "i2" "i3"
> m$getUELs()
[1] "i1" "i2" "i3" "j1" "j2" "j3"
```

But perhaps we want to reorder the UELs `i1`, `i2`, `i3` to `i3`, `i2`, `i1`.

```
i$reorderUELs(c("i3","i2","i1"))
> i$getUELs()
[1] "i3" "i2" "i1"
> i$records
    uni
1    i1
2    i2
3    i3
```

Note

> This example does not change the indexing scheme of the data frame. It only changes the underlying integer numbering scheme for the factor levels.
> We can see this by looking at the `levels`:

```
> as.integer(i$records$uni)
[1] 3 2 1
```

When `reorderUELs()` is used without the `uels` argument, the UELs are rearranged based on the records order as illustrated in the following example.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
i$setUELs(c("i2","i3","i1"))
> i$getUELs()
[1] "i2" "i3" "i1"
> i$reorderUELs()
> i$getUELs()
[1] "i1" "i2" "i3"
```

Moreover, if there are unused UELs, they are moved to the end as shown below. Here, `i4` is the unused UELs that gets moved to the end after using `reorderUELs`.

```
> i$setUELs(c("i2","i3","i4","i1"))
> i$getUELs()
[1] "i2" "i3" "i4" "i1"
> i$reorderUELs()
> i$getUELs()
[1] "i1" "i2" "i3" "i4"
```

setUELs Examples

> `setUELs` is a method of all GAMS symbol classes. This method allows the user to create new UELs, rename UELs, and reorder UELs all in one method. For example:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
```

A user could accomplish a UEL reorder operation with `setUELs`:

```
> i$setUELs(c("i3","i2","i1"))
> i$getUELs()
[1] "i3" "i2" "i1"
> i$records
    uni
1    i1
2    i2
3    i3
```

A user could accomplish a UEL reorder + add UELs operation with `setUELs`:

```
> i$setUELs(c("i3", "i2", "i1", "j1", "j2"))
> i$getUELs()
[1] "i3" "i2" "i1" "j1" "j2"
> i$records
    uni
1    i1
2    i2
3    i3
> as.integer(i$records$uni)
[1] 3 2 1
```

A user could accomplish a UEL reorder + add + rename with `setUELs`:

```
> i$setUELs(c("j3", "j2", "j1", "ham", "cheese"), rename=TRUE)
> i$setUELs(c("j3", "j2", "j1", "ham", "cheese"), rename=TRUE)
> i$getUELs()
[1] "j3"     "j2"     "j1"     "ham"    "cheese"
> i$records
    uni
1    j1
2    j2
3    j3
> as.integer(i$records$uni)
[1] 3 2 1
```

Note

> This example does not change the indexing scheme of the data frame, but the `rename=TRUE` flag means that the records will get updated as if a `renameUELs` call had been made.

If a user wanted to set new UELs on top of this data, without renaming, they would need to be careful to include the current UELs in the UELs being set. It is possible to lose these labels if they are not included (which will prevent the data from being written to GDX).

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("i1","i2","i3"))
i$setUELs(c("j1", "i2", "j3", "ham", "cheese"))
> i$getUELs()
[1] "j1"     "i2"     "j3"     "ham"    "cheese"
> i$records
  uni_1
1  <NA>
2    i2
3  <NA>
```

Removing Symbols

Removing symbols from a Container is easy when using the `removeSymbols` container method; this method accepts either a `string` or a `list` of `string`.

Reordering Symbols

In order to write the contents of the Container, it is required that the symbols are sorted such that, for example, a Set used as a domain of another symbol appears before that symbol. The Container will try to establish a valid ordering when writing the data. This type of situation could be encountered if the user is adding and removing many symbols (and perhaps rewriting symbols with the same name). Users should attempt to only add symbols to a `Container` once, and care must be taken when naming. The method `reorderSymbols` attempts to fix symbol ordering problems. The following example shows how this can occur:

Example Symbol reordering

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = paste0("i", 1:5))
j = Set$new(m, "j", i, records = paste0("i", 1:3))
> m$listSymbols()
[1] "i" "j"
# now we remove the set i and recreate the data
m$removeSymbols("i")
i = Set$new(m, "i", records = paste0("i", 1:5))
> m$isValid()
[1] TRUE
```

Since the link to `i` is broken, Set `j` is now invalid. The user has to manually set the domain again.

```
# fix the domain reference in the set j
j$domain = i
> m$listSymbols()
[1] "j" "i"
```

Now even though `j` is valid, the symbols are out of order. The order can be fixed as follows.

```
# calling reorderSymbols() will order the list properly,
# but the domain reference in j is now broken
m$reorderSymbols()
> m$listSymbols()
[1] "i" "j"
> m$isValid()
[1] TRUE
```

Domain Violations

Domain violations occur when a symbol uses other Sets as domain(s) – and is thus of domain type regular, see Symbol Domain – and uses a domain entry in its records that is not present in the corresponding referenced domain set. Such a domain violation will lead to a GDX error when writing the data.

For example, the symbol `j` in the following example contains domain violations because `j` is defined over domain set `i` and the records entry "grayslake" is not present in `i`.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records = c("seattle", "san-diego", "washington_dc"))
j = Set$new(m, "j", i)
records = data.frame(cities=c("grayslake", "washington_dc"))
j$setRecords(records)
```

Trying to write this container to a GDX file will fail. To ask for domain violations, call the method `Symbol$getDomainViolations()`. It returns a list of `DomainViolation` objects with respect to each dimension of the symbol. This list can then be used to resolve the domain violations.

```
> dv = j$getDomainViolations()
> dv
[[1]]
GAMS Transfer: DomainViolation with properties:
Symbol: j
dimension:
domain: i
violations: grayslake
```

The GAMS Transfer R feature of domain forwarding offers an automatic expansion of the domain set with the violated entries in order to eliminate domain violations.

Note

Checking for domain violations is not a part of Symbol$isValid() for performance reasons.

The method for automatically resolving the domain violations can be convenient, but it effectively disables domain checking, which is a valuable tool for error detection. We encourage to use domain forwarding as rarely as possible. The same holds for using `relaxed` domain informaiton when `regular` domain information would be possible.

GAMS Special Values

The GAMS system contains five special values: `UNDEF` (undefined), `NA` (not available), `EPS` (epsilon), `+INF` (positive infinity), `-INF` (negative infinity). These special values must be mapped to their R equivalents. GAMS Transfer R follows the following convention to generate the `1:1` mapping:

- `+INF` is mapped to `Inf`
- `-INF` is mapped to `-Inf`
- `EPS` is mapped to `-0.0` (mathematically identical to zero)
- `NA` is mapped to a `NA`
- `UNDEF` is mapped to `NaN`

GAMS Transfer R syntax is designed to quickly get data into a form that is usable in further analyses or visualization. The user does not need to remember these constants as they are provided within the class `SpecialValues` as `SpecialValues$POSINF`, `SpecialValues$NEGINF`, `SpecialValues$EPS`, `SpecialValues[["NA"]]`, and `SpecialValues$UNDEF`. Some examples are shown below.

```
library(gamstransfer)
m = Container$new()
x = Parameter$new(
  m, "x", c("*"),
records = data.frame(uni=paste0("i", 1:6), c(1, SpecialValues[["POSINF"]],
SpecialValues[["NEGINF"]], SpecialValues[["EPS"]], SpecialValues[["NA"]],
SpecialValues[["UNDEF"]])),
description = "special values"
)
```

The following data frame for `x` would look like:

```
> x$records
    uni value
1    i1     1
2    i2   Inf
3    i3  -Inf
4    i4     0
5    i5    NA
6    i6   NaN
```

The user can now easily test for specific special values in the `value` column of the DataFrame (returns a logical vector):

```
> SpecialValues$isNA(x$records$value)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE
> SpecialValues$isNA(x$records$value)
[1] FALSE FALSE FALSE FALSE  TRUE FALSE
> SpecialValues$isEps(SpecialValues$EPS)
[1] TRUE
> SpecialValues$isPosInf(SpecialValues$POSINF)
[1] TRUE
> SpecialValues$isNegInf(SpecialValues$NEGINF)
[1] TRUE
> SpecialValues$isNA(SpecialValues$NEGINF)
SpecialValues$isNA
> SpecialValues$isNA(SpecialValues[["NA"]])
[1] TRUE
> SpecialValues$isUndef(SpecialValues$UNDEF)
[1] TRUE
> SpecialValues$isUndef(SpecialValues[["NA"]])
[1] FALSE
> SpecialValues$isNA(SpecialValues$UNDEF)
[1] FALSE
```

Note

The syntax SpecialValues$NA is not allowed in R. Therefore, to access NA, one has to use `SpecialValues[["NA"]]`. As shown in the example above, double bracket syntax works for other special values too.

Standard Data Formats

This section is meant to introduce the standard format that GAMS Transfer R expects for symbol records. It has already been mentioned that we store data as a data frame, but there is an assumed structure to the column headings and column types that will be important to understand. GAMS Transfer R includes convenience functions in order to ease the burden of converting data from a user-centric format to one that is understood by GAMS Transfer R. However, advanced users will want to convert their data first and add it directly to the Container.

Set Records Standard Format

All set records (including singleton sets) are stored as a data frame with `n` or `n+1` number of columns, where `n` is the dimensionality of the symbol. The first `n` columns include the domain elements while the last column, if present, includes the set element text. Records are organized such that there is one record per row.

The names of the domain columns (domain labels) are unique. If the domain labels are non-unique, they are converted to follow a pattern of <`user_domain_label`>_<`index_position`> by `domainLabels` setter. If the records are passed in a non-data frame format (vector, matrix etc.), domain labels follow a pattern of <`domain_name`>_<`index_position`>. A symbol dimension that is referenced to the universe is labeled `uni`. The explanatory text column is called `element_text` and must take the last position in the data frame.

All domain columns must be `factors` and the `element_text` column must be a `string` type.

Some examples:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("seattle", "san-diego"))
j = Set$new(m, "j", c(i, "*"), records=data.frame(i=c("seattle", "san-diego"), uni=c("new-york", "st-louis")))
k = Set$new(m, "k", i, isSingleton=TRUE, records=c("seattle"))
> i$records
        uni
1   seattle
2 san-diego
> j$records
          i      uni
1   seattle new-york
2 san-diego st-louis
> k$records
        i
1 seattle
```

Parameter Records Standard Format

All parameter records (including scalars) are stored as a data frame with `n` or `n + 1` number of columns, where `n` is the dimensionality of the symbol. The first `n` columns include the domain elements while the last column, if present, includes the numerical value of the records. Records are organized such that there is one record per row. Scalar parameters have zero dimension, therefore they only have one column and one row. In cases where the `value` columns is not present for a parameter, it is assumed that the value is the default parameter value (i.e., 0). Scalar parameters with an empty data frame as records is assumed to contain the default value of 0.

The names of the domain columns (domain labels) are unique. If the domain labels are non-unique, they are converted to follow a pattern of <user_domain_label>_<index_position> by `domainLabels` setter. If the records are passed in a non-data frame format (vector, matrix etc.), domain labels follow a pattern of <domain_name>_<index_position>. A symbol dimension that is referenced to the universe is labeled uni_<index_position>. The value column is called `value` and must take the last position in the data frame.

All domain columns must be `factors` and the `value` column must be a `numeric` type. GAMS Transfer R requires that all the factor levels are of type `string`.

Some examples:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("seattle", "san-diego"))
a = Parameter$new(
  m, "a", "*", records=data.frame(c("seattle", "san-diego"), c(50, 100))
  )
b = Parameter$new(
    m,
    "b",
    c(i, "*"),
    records= data.frame(i=c("seattle", "san-diego"),
    uni=c("new-york", "st-louis"), c(32.2, 123))
)
c = Parameter$new(m, "c", records=90)
> a$records
        uni value
1   seattle    50
2 san-diego   100
> b$records
          i      uni value
1   seattle new-york  32.2
2 san-diego st-louis 123.0
> c$records
  value
1    90
```

Variable/Equation Records Standard Format

Variables and equations share the same standard data format. All records (including scalar variables/equations) are stored as a data frame with `n` to `n` + 5 number of columns, where `n` is the dimensionality of the symbol. The first `n` columns include the domain elements while the remaining columns include the numerical values for different attributes of the records. Records are organized such that there is one record per row. Scalar variables/equations have zero dimension, therefore they have one row. In cases where the records data frame has missing attribute columns, it is assumed that the column contains the default value. Similarly, if the records data frame for a scalar is empty, it is assumed that all the attribute values are at their default. Default values for variables and equations depend on their type.

The names of the domain columns (domain labels) are unique. If the domain labels are non-unique, they are converted to follow a pattern of <user_domain_label>_<index_position> by `domainLabels` setter. If the records are passed in a non-data frame format (vector, matrix etc.), domain labels follow a pattern of <domain_name>_<index_position>. A symbol dimension that is referenced to the universe is labeled uni_<index_position>. The attribute columns are called `level`, `marginal`, `lower`, `upper`, and `scale`. These attribute columns must appear in this order. Attributes that are not supplied by the user will be assigned the default GAMS values for that variable/equation type. It is possible to not pass any attributes, GAMS Transfer R would then simply assign default values to all attributes.

All domain columns must be `factors` and the attribute columns must be a `numeric` type. GAMS Transfer R requires that all the factor levels are of type `string`.

Some examples:

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("seattle", "san-diego"))
a = Variable$new(
    m,
```

```
    "a",
    "free",
    domain= i,
    records=data.frame(
      city = c("seattle", "san-diego"),
      level = c(50, 100)
    )
)
> a$records
       city level
1   seattle    50
2 san-diego   100
```

Generate Records

Container symbol records in standard format can also be generated using the convenience method `generateRecords()`. This method generates records with the cartesian product of domain all sets. If the argument `density` is less than 1, randomly selected records are removed. For symbols that are not scalar, using this method requires that the symbol domain type is "regular" (i.e., <`symbol_name`>`$domainType = "regular"`). A few examples using the method `generateRecords()` for each type of Container symbol are provided below.

Set

Example #1 Create a large (dense) 4D set

Generating the initial `data.frame` could be difficult for `Set` symbols that have a large number of records and a small number of UELs. These higher dimensional symbols will benefit from the generateRecords convenience method.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:50))
j = Set$new(m, "j", records=paste0("j", 1:50))
k = Set$new(m, "k", records=paste0("k", 1:50))
l = Set$new(m, "l", records=paste0("l", 1:50))
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = Set$new(m, "a", c(i, j, k, l))
# generate the records
a$generateRecords()
> a$isValid()
[1] TRUE
> head(a$records)
   i  j  k  l element_text
1 i1 j1 k1 l1
2 i2 j1 k1 l1
3 i3 j1 k1 l1
4 i4 j1 k1 l1
5 i5 j1 k1 l1
6 i6 j1 k1 l1
> tail(a$records)
          i   j   k   l element_text
6249995 i45 j50 k50 l50
6249996 i46 j50 k50 l50
6249997 i47 j50 k50 l50
6249998 i48 j50 k50 l50
6249999 i49 j50 k50 l50
6250000 i50 j50 k50 l50
```

Example #2 Create a large (sparse) 4D set

It is also possible to generate a sparse set (randomly selected rows are removed from the dense dataframe) with the `density` argument to `generateRecords`.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:50))
j = Set$new(m, "j", records=paste0("j", 1:50))
k = Set$new(m, "k", records=paste0("k", 1:50))
l = Set$new(m, "l", records=paste0("l", 1:50))
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = Set$new(m, "a", c(i, j, k, l))
# generate the records
a$generateRecords(density = 0.05)
> a$isValid()
[1] TRUE
> head(a$records)
    i   j   k   l element_text
1 i15  j1  k1  l1
2 i41  j1  k1  l1
3 i37  j2  k1  l1
4 i17  j3  k1  l1
5 i21  j3  k1  l1
6 i37  j3  k1  l1
> tail(a$records)
         i   j   k   l element_text
312495  i6 j48 k50 l50
312496  i9 j49 k50 l50
312497 i14 j49 k50 l50
312498 i41 j49 k50 l50
312499 i44 j49 k50 l50
312500 i35 j50 k50 l50
```

Example #3 Create a large 4D set with 1 sparse dimension

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:50))
j = Set$new(m, "j", records=paste0("j", 1:50))
k = Set$new(m, "k", records=paste0("k", 1:50))
l = Set$new(m, "l", records=paste0("l", 1:50))
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = Set$new(m, "a", c(i, j, k, l))
# generate the records
a$generateRecords(density = c(1, 0.05, 1, 1))
> a$isValid()
[1] TRUE
> head(a$records)
    i   j   k   l element_text
1  i1 j29  k1  l1
2  i2 j29  k1  l1
3  i3 j29  k1  l1
4  i4 j29  k1  l1
5  i5 j29  k1  l1
6  i6 j29  k1  l1
> tail(a$records)
         i   j   k   l element_text
249995 i45 j45 k50 l50
249996 i46 j45 k50 l50
249997 i47 j45 k50 l50
249998 i48 j45 k50 l50
249999 i49 j45 k50 l50
250000 i50 j45 k50 l50
```

Parameter

Example #1 Create a large (dense) 4D Parameter

Generating the initial `data.frame` could be difficult for `Parameter` symbols that have a large number of records and a small number of UELs. These higher dimensional symbols will benefit from the `generateRecords` convenience method.

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:50))
j = Set$new(m, "j", records=paste0("j", 1:50))
```

```
k = Set$new(m, "k", records=paste0("k", 1:50))
l = Set$new(m, "l", records=paste0("l", 1:50))
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = Parameter$new(m, "a", c(i, j, k, l))
# generate the records
a$generateRecords()
> a$isValid()
[1] TRUE
> head(a$records)
    i   j   k   l      value
1  i1  j1  k1  l1 0.47998665
2  i2  j1  k1  l1 0.20015289
3  i3  j1  k1  l1 0.57701174
4  i4  j1  k1  l1 0.73032070
5  i5  j1  k1  l1 0.08637669
6  i6  j1  k1  l1 0.45913994
> tail(a$records)
         i   j   k   l      value
6249995 i45 j50 k50 l50 0.91182978
6249996 i46 j50 k50 l50 0.79016549
6249997 i47 j50 k50 l50 0.77912069
6249998 i48 j50 k50 l50 0.63232201
6249999 i49 j50 k50 l50 0.04274219
6250000 i50 j50 k50 l50 0.71523280
```

Note

In Example #1 a large 4D parameter was generated. by default, the value of these records are randomly drawn numbers from the interval `[0,1]` (uniform distribution).

Example #2 - Create a large (sparse) 4D parameter with normally distributed values

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:50))
j = Set$new(m, "j", records=paste0("j", 1:50))
k = Set$new(m, "k", records=paste0("k", 1:50))
l = Set$new(m, "l", records=paste0("l", 1:50))
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = Parameter$new(m, "a", c(i, j, k, l))
# create a custom function to pass to ‘generateRecords‘
value_dist = function(size) {
  return(rnorm(n=size, mean = 10, sd = 2.3))
}
# generate the records
a$generateRecords(density = 0.05, func = value_dist)
> a$isValid()
[1] TRUE
> head(a$records)
    i   j   k   l     value
1 i50  j1  k1  l1 12.499060
2  i6  j2  k1  l1 12.009952
3 i14  j2  k1  l1  9.931126
4 i49  j2  k1  l1 13.073977
5  i7  j3  k1  l1  5.330898
6 i22  j3  k1  l1  7.887725
> tail(a$records)
        i   j   k   l     value
312495 i14 j48 k50 l50 10.213841
312496 i20 j48 k50 l50  4.831503
312497 i26 j48 k50 l50  8.129577
312498 i17 j49 k50 l50 11.570570
312499 i48 j49 k50 l50 11.321228
312500 i35 j50 k50 l50  1.714614
> mean(a$records$value)
[1] 10.00273
> sd(a$records$value)
[1] 2.303193
```

Note

The custom function passed to the argument `func` must expose a `size` argument. It might be tedious to know the exact number of the records that will be generated, especially if a fractional density is specified; therefore, the `generateRecords` method will pass in the correct size automatically.

Example #3 - Create a large 4D parameter with a random number seed

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:50))
j = Set$new(m, "j", records=paste0("j", 1:50))
k = Set$new(m, "k", records=paste0("k", 1:50))
l = Set$new(m, "l", records=paste0("l", 1:50))
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = Parameter$new(m, "a", c(i, j, k, l))
a2 = Parameter$new(m, "a2", c(i, j, k, l))
# generate the records
a$generateRecords(density = 0.05, seed = 123)
a2$generateRecords(density = 0.05)
> a$equals(a2, checkMetaData = FALSE)
[1] FALSE
a2$generateRecords(density = 0.05, seed = 123)
> a$equals(a2, checkMetaData = FALSE)
[1] TRUE
```

Note

The `seed` is an `int` that will set the random number generator state (enables reproducible sequences of random numbers).

Variable and Equation

Generating records for the symbol types `Variable` and `Equation` is similar to that of previously shown examples of parameters and sets. However, since there are more than one attributes to variables and equations, there are a few differences. By default, the random sampling is done is only for the `level` attribute with default values being passed to the other attributes. To randomly generate other attributes, one can use the custom `func` argument. This is shown in the following example.

Example #1 Create a large (sparse) 4D variable and Equation

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:50))
j = Set$new(m, "j", records=paste0("j", 1:50))
k = Set$new(m, "k", records=paste0("k", 1:50))
l = Set$new(m, "l", records=paste0("l", 1:50))
# create and define the symbol ‘a‘ with ‘regular‘ domains
av = Variable$new(m, "av", "free", c(i, j, k, l))
ae = Equation$new(m, "ae", "eq", c(i, j, k, l))
# user can pass the function in-line as follows
av$generateRecords(density=0.05,
func=list(level= function(size) rnorm(n=size, mean=10, sd=2.3),
marginal = function(size) rnorm(n=size, mean=0.5, sd=0.1)))
# functions can also be defined first and then passed
uniform_distr = function(size) return(runif(size))
normal_distr = function(size) return(rnorm(n=size))
ae$generateRecords(density=0.05, func=list(level=uniform_distr, marginal=normal_distr))
> head(av$records)
    i   j   k   l     level  marginal lower upper scale
1 i23  j1  k1  l1 12.244702 0.5587150  -Inf   Inf     1
2 i29  j1  k1  l1  8.265612 0.4242353  -Inf   Inf     1
3  i2  j2  k1  l1 14.164058 0.4166124  -Inf   Inf     1
4 i17  j2  k1  l1 13.786874 0.5993234  -Inf   Inf     1
5 i22  j2  k1  l1  8.489724 0.4924503  -Inf   Inf     1
6 i36  j2  k1  l1  7.962292 0.4757125  -Inf   Inf     1
> tail(av$records)
         i   j   k   l     level  marginal lower upper scale
312495 i33 j48 k50 l50  6.648296 0.4870270  -Inf   Inf     1
312496 i37 j48 k50 l50 10.012486 0.5478388  -Inf   Inf     1
312497 i20 j49 k50 l50  7.931512 0.4221189  -Inf   Inf     1
312498 i41 j49 k50 l50 10.869332 0.5191488  -Inf   Inf     1
312499 i42 j49 k50 l50  9.316445 0.4263974  -Inf   Inf     1
312500 i44 j49 k50 l50  8.153729 0.6101864  -Inf   Inf     1
> head(ae$records)
    i   j   k   l      level   marginal lower upper scale
1  i5  j1  k1  l1 0.74525909  0.8910060     0     0     1
```

```
2 i10  j1  k1  l1 0.72308699  1.6090443     0     0     1
3 i22  j1  k1  l1 0.70425801 -1.2204379     0     0     1
4 i47  j1  k1  l1 0.06490871  0.7270846     0     0     1
5 i24  j2  k1  l1 0.94752455  0.7864338     0     0     1
6 i35  j2  k1  l1 0.08555602 -0.2912885     0     0     1
> tail(ae$records)
          i   j   k   l     level     marginal lower upper scale
312495  i5 j49 k50 l50 0.7844452 -0.569529636     0     0     1
312496 i46 j49 k50 l50 0.2224596 -0.182441937     0     0     1
312497 i11 j50 k50 l50 0.9291730 -0.474982758     0     0     1
312498 i16 j50 k50 l50 0.3347919  0.009303616     0     0     1
312499 i20 j50 k50 l50 0.3590295 -0.533782269     0     0     1
312500 i27 j50 k50 l50 0.7681852 -1.126704380     0     0     1
```

Alias

The method `generateRecords` for an alias simply calls the corresponding method for its referenced set.

Example #1 Create a large (dense) 4D set from an Alias

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=paste0("i", 1:50))
j = Set$new(m, "j", records=paste0("j", 1:50))
k = Set$new(m, "k", records=paste0("k", 1:50))
l = Set$new(m, "l", records=paste0("l", 1:50))
# create and define the symbol ‘a‘ with ‘regular‘ domains
a = Set$new(m, "a", c(i, j, k, l))
# create an Alias ap for the set a
ap = Alias$new(m, "ap", a)
# generate the records
ap$generateRecords()
> ap$isValid()
[1] TRUE
> head(ap$records)
    i   j   k   l element_text
1  i1  j1  k1  l1
2  i2  j1  k1  l1
3  i3  j1  k1  l1
4  i4  j1  k1  l1
5  i5  j1  k1  l1
6  i6  j1  k1  l1
> tail(ap$records)
          i   j   k   l element_text
6249995 i45 j50 k50 l50
6249996 i46 j50 k50 l50
6249997 i47 j50 k50 l50
6249998 i48 j50 k50 l50
6249999 i49 j50 k50 l50
6250000 i50 j50 k50 l50
```

Data Exchange with GDX

Up until now, we have been focused on using GAMS Transfer R to create symbols in an empty Container using the symbol constructors (or their corresponding container methods). These tools will enable users to ingest data from many different formats and add them to a `Container`. However, it is also possible to read in symbol data directly from GDX files using the `read` container method. In the following sections, we will discuss this method in detail as well as the `write` method, which allows users to write out to new GDX files.

Reading from GDX

There are two main ways to read in GDX based data.

- Pass the file path directly to the Container constructor (will read all symbols and records)

- Pass the file path directly to the `read` method (default read all symbols, but can read partial files)

The first option here is provided for convenience and will, internally, call the `read` method. For the following examples, we leverage the GDX output generated from the **trnsport.gms** model file.

Example (reading full data into a Container using the constructor)

```
library(gamstransfer)
m = Container$new("trnsport.gdx")
> m$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
> m$describeParameters()
  name domain domainType dimension numberRecords     min     mean     max
1    a      i    regular         1             2 350.000 475.0000 600.000
2    b      j    regular         1             3 275.000 300.0000 325.000
5    c    i j    regular         2             6   0.126   0.1755   0.225
3    d    i j    regular         2             6   1.400   1.9500   2.500
4    f     NA       none         0             1  90.000  90.0000  90.000
  whereMin whereMax sparsity
1        1        2        0
2        3        1        0
5        6        1        0
3        6        1        0
4        1        1       NA
```

A user could also read in data with the read method as shown in the following example.

Example (reading full data into a Container with `read` method)

```
library(gamstransfer)
m = Container$new()
m$read("trnsport.gdx")
> m$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
```

It is also possible to read in a partial GDX file with the read method, as shown in the following example:

```
library(gamstransfer)
m = Container$new()
m$read("trnsport.gdx", "x")
> m$listSymbols()
[1] "x"
> m["x"]$records
         i        j level marginal lower upper scale
1   seattle new-york    50    0.000     0   Inf     1
2   seattle  chicago   300    0.000     0   Inf     1
3   seattle   topeka     0    0.036     0   Inf     1
4 san-diego new-york   275    0.000     0   Inf     1
5 san-diego  chicago     0    0.009     0   Inf     1
6 san-diego   topeka   275    0.000     0   Inf     1
```

This syntax assumes that the user will always want to read in both the metadata as well as the actual data records, but it is possible to skip the reading of the records by passing the argument `records=FALSE`.

```
library(gamstransfer)
m = Container$new()
m$read("trnsport.gdx", "x", records = FALSE)
> m$listSymbols()
[1] "x"
> m["x"]$summary
$name
[1] "x"
$description
[1] "shipment quantities in cases"
$type
[1] "positive"
$domain
[1] "i" "j"
$domainType
[1] "relaxed"
$dimension
[1] 2
$numberRecords
[1] 0
> m["x"]$records
NULL
```

Attention

> The `read` method attempts to preserve the symbol domain type from the source but if domain sets are not part of the read operation there is no choice but to default to a "relaxed" `domainType`. This can be seen in the last example where we only read in the variable `x` and not the domain sets (`i` and `j`) that the variable is defined over. All the data will be available to the user, but domain checking is no longer possible. The symbol `x` will remain with "relaxed" domain type even if the user were to read in sets `i` and `j` in a second `read` call.

Writing to GDX

A user can write data to a GDX file by simply passing a file path (as a string). The `write` method will then create the GDX and write all data in the `Container`.

Note

> It is not possible to write the Container when any of its symbols are invalid. If any symbols are invalid an error will be raised and the user will need to inspect the problematic symbols (perhaps using a combination of the `listSymbols(isValid=FALSE)` and `isValid(verbose=TRUE)` methods).

Example

```
m$write("path/to/file.gdx")
```

Example (write a compressed GDX file)

```
m$write("path/to/file.gdx", compress = TRUE)
```

Advanced users might want to specify an order to their UELs (i.e., the universe set); recall that the UEL ordering follows that dictated by the data. As a convenience, it is possible to prepend the list of UELs with a user specified order using the `uelPriority` argument.

Example (change the order of the UEL)

```
library(gamstransfer)
m = Container$new()
i = Set$new(m, "i", records=c("a", "b", "c"))
m$write("foo.gdx", uelPriority=c("a", "c"))
```

The original UEL order for this GDX file would have been `c("a", "b", "c")`, but since this example reorders the UELs with `uelPriority`, the positions of `b` and `c` have been swapped. This can be verified with the gdxdump utility (using the `uelTable` argument):

```
gdxdump foo.gdx ueltable=foo

Set foo /
  'a' ,
  'c' ,
  'b' /;
$onEmpty

Set i(*) /
'a',
'c',
'b' /;

$offEmpty
```

Data Exchange between Containers

GAMS Transfer R allows data exchange between two Containers with the help of read and copy methods.

Data exchange with read

Similar to reading from a GDX file, a Container can read from another Container object. Following examples demonstrate this with the help of the GDX output generated from **trnsport.gms** model file.

Example (reading data from Container)

```
> library(gamstransfer)
> c = Container$new("trnsport.gdx")
> c$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
> m = Container$new(c)
> m$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
> m = Container$new()
> m$read(c)
> m$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
> m = Container$new()
> m$read(c, symbols=c("d","f"))
> m$listSymbols()
[1] "d" "f"
```

Example (reading data when symbol with the same name already exists)

```
> library(gamstransfer)
> c = Container$new()
> i = Set$new(c, "i")
> p = Parameter$new(m, "p")
> m = Container$new()
> i = Set$new(m, "i")
> m$read(c, symbols="i")
Error in private$.containerRead(loadFrom, symbols, records) :
  Attempting to add symbol i, however, one already exists in the Container. Symbol replacement is only possible if the
      symbol is first removed from theContainer with the removeSymbols() method.
```

The container read method does not allow reading from another source (Container, or a GDX file) when a symbol with the same name already exists. The existing symbol must be removed or renamed.

Data exchange with copy

Symbol `copy` method provides an alternative way to exchange data between Containers. Following examples demonstrate this starting from a Container containing data from GDX output generated from **trnsport.gms** model file.

Example (copy symbol from one container to another)

```
> library(gamstransfer)
> c = Container$new("trnsport.gdx")
> c$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
> m = Container$new()
> c["f"]$copy(m)
> m$listSymbols()
[1] "f"
```

The above example copies symbol `f` from Container `c` to Container `m`. If one copies a symbol with domain that does not exist in the destination Container, domain is relaxed as shown in the following example.

Example (copy symbol to another container without domain symbols)

```
> library(gamstransfer)
> c = Container$new("trnsport.gdx")
> c$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
m = Container$new()
> c["d"]$copy(m)
> m$listSymbols()
[1] "d"
> m["d"]$domain
[1] "i" "j"
> m["d"]$domainType
[1] "relaxed"
```

Example (copy symbol to another container with overwrite)

```
> library(gamstransfer)
> c = Container$new()
> i = Set$new(c, "i", records=c("i1","i2"))
> i$records
   uni
1   i1
2   i2
> m = Container$new()
> i = Set$new(m, "i", records= c("i3","i4"))
> i$records
   uni
1   i3
2   i4
# the following command throws an error
> c["i"]$copy(m)
Error in private$.copy(destination, overwrite) :
  Symbol i already exists in ‘destination‘
> c["i"]$copy(m, overwrite = TRUE)
> m["i"]$records
   uni
1   i1
2   i2
```

Example (bulk copy operation via Container `copy` method)

A bulk operation is also possible via Container `copy` method as shown in the following example.

```
> library(gamstransfer)
> c = Container$new("trnsport.gdx")
> c$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
> m = Container$new()
> c$copy(m) # copy all symbols
> c$listSymbols()
 [1] "i"      "j"      "a"      "b"      "d"      "f"      "c"      "x"
 [9] "z"      "cost"   "supply" "demand"
> m = Container$new()
> c$copy(m, symbols=c("a","b","d")) # copy a subset of symbols
> m$listSymbols()
[1] "a" "b" "d"
> c$copy(m, symbols="a", overwrite = TRUE) # copy symbols with overwrite
> m$listSymbols()
[1] "a" "b" "d"
```

7.8 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS C# API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known `transportation problem`. These examples introduce several API features step by step.

- Getting Started A quick introduction about how to create and configure C# project
- Important Classes of the API Overview of some fundamental classes of the GAMS C# API
- How to use API An extensive set of examples how to use API components

7.8.1 Getting Started

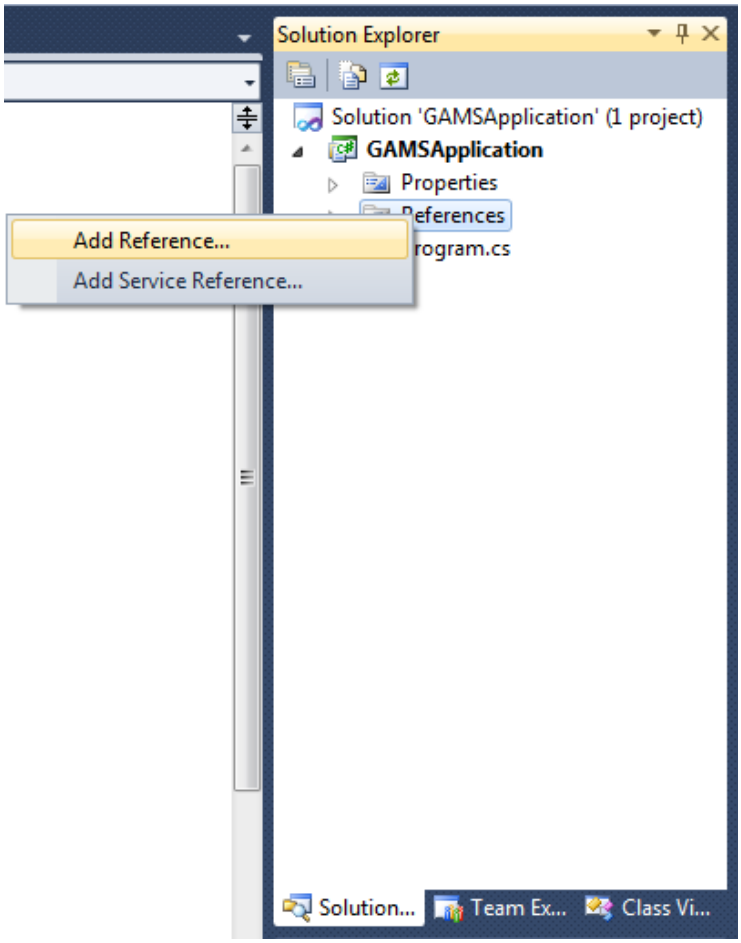
This section takes you through the basic steps of creating and configuring a C# project in Visual Studio 2010 for use with the GAMS C# API. At the end of this section there is also a paragraph about Mono as an alternative to Microsoft Visual Studio.

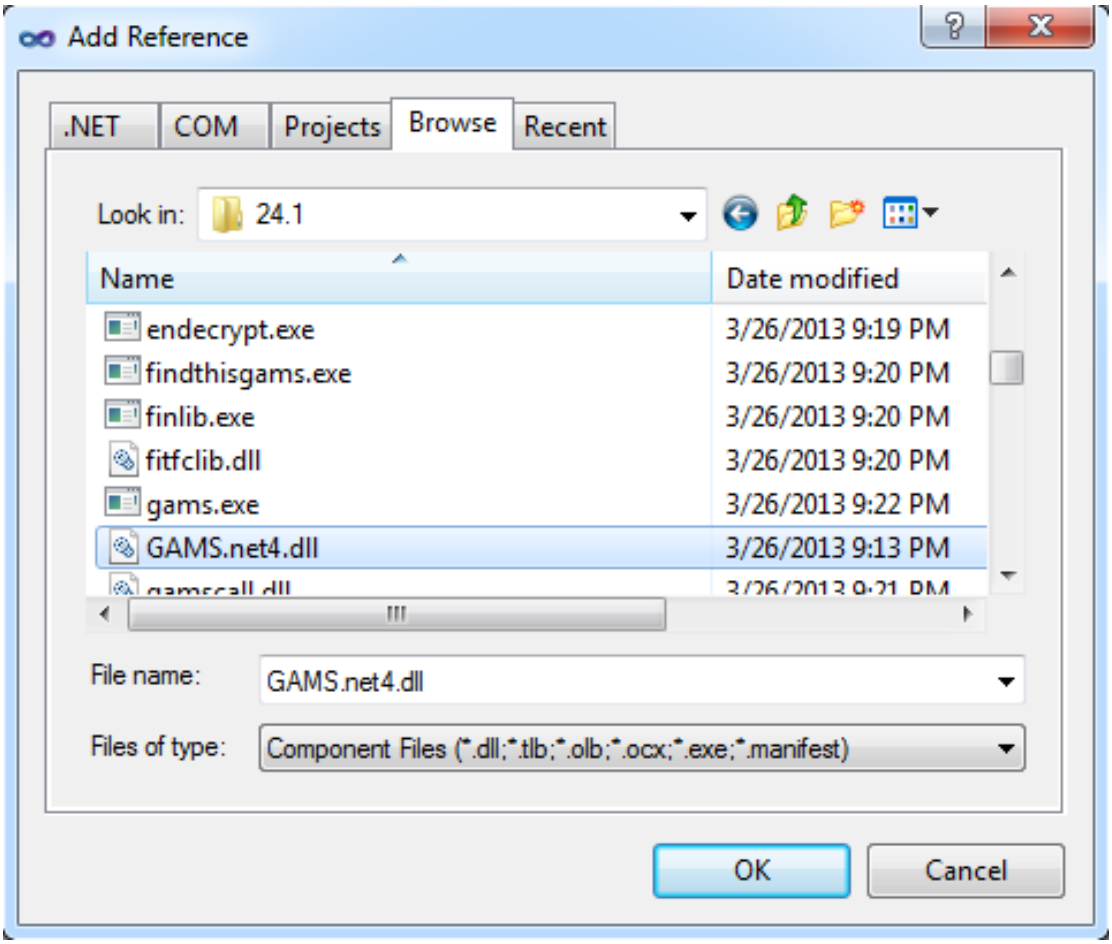
7.8.1.1 Open a new Project

After opening Microsoft Visual Studio you can open the New Project wizard by choosing **File** > **New** > **Project** from the ribbon menu and type a name for your application, e.g. GAMSApplication.

7.8.1.2 Add a Reference to a Project

Once you opened the project add a reference by a right click on **References** inside the solution explorer. Then choose **Add Reference..** and select the **Browse** tab. Next navigate to the location where you installed `GAMS.net4.dll`, typically the GAMS system directory, e.g. `C:\GAMS\win64\`46.4. Then choose **GAMS.net4.dll** and click **OK**.





7.8.1.3 Solving your first Model

This section illustrates how a model from the GAMS model library can be imported and solved.

Open a new project and reference the `GAMS.net4.dll` in your project as explained above. Open `Program.cs` which should be part of your project and replace the content by:

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using GAMS;
namespace TransportSeq
{
  class Transport0
  {
      static void Main(string[] args)
      {
          GAMSWorkspace ws = new GAMSWorkspace();
          ws.GamsLib("trnsport");
          // create a GAMSJob from file and run it with default settings
          GAMSJob t0 = ws.AddJobFromFile("trnsport.gms");
          t0.Run();
          Console.WriteLine("Ran with Default:");
          foreach (GAMSVariableRecord rec in t0.OutDB.GetVariable("x"))
              Console.WriteLine("x(" + rec.Keys[0] + "," + rec.Keys[1] + "): level=" + rec.Level + " marginal=" +
      rec.Marginal);
          Console.WriteLine("z=" + t0.OutDB.GetVariable("z").LastRecord().Level);
       }
   }
}
```

Then click the highlighted button from figure (a) down below or press **F5** to start debugging. You can also start the program without debugging by pressing **Control + F5** or choosing **Debug > Start Without Debugging** from the ribbon menu as illustrated in figure (b).

Figure (a)

Figure(b)

By following the latter instruction the command line shell remains open and show the produced output.

7.8.1.4 Mono

The GAMS .NET API works also on non-Windows platforms where no Visual Studio is available. It can be used for example with the open source .Net framework Mono (http://www.mono-project.com). Mono offers a IDE that could be used similarly to Visual Studio. If one prefers to compile and execute from a shell, the following code shows how this can be done using the example Transport1 which comes with the GAMS system (this assumes that xbuild and mono are in the PATH):

```
cd <GAMS Dir>/apifiles/CSharp/Transport1
xbuild /t:rebuild /p:Configuration=Release Transport1.csproj
mono bin/Release/Transport1.exe ../../..
```

7.8.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS Namespace. Their usage is demonstrated by an extensive set of examples.

- **GAMS** Namespace
- **GAMSWorkspace** Class
- **GAMSJob** Class
- **GAMSDatabase** Class
- **GAMSOptions** Class
- **GAMSModelInstance** Class

7.8.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the C# API. `<GAMS system directory>\apifiles\CSharp` contains a file `TransportSeq.sln` that can be opened in Microsoft Visual Studio. The contained projects deal with the well-known transportation problem. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- How to choose the GAMS system (Transport1)
- How to export data to GDX (TransportGDX)
- How to import data from GDX (TransportGDX)
- How to run a GAMSJob from file (Transport1)
- How to specify the solver (Transport1)
- How to run a job with a solver option file and capture its log output (Transport1)

- How to use include files (Transport2)
- How to read data from string and export to GDX (Transport3)
- How to run a job using data from GDX (Transport3)
- How to run a job using implicit database communication (Transport3)
- How to define data using C# data structures (Transport4)
- How to prepare a GAMSDatabase from C# data structures (Transport4)
- How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)
- How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)
- How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)
- How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)
- How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)
- How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)
- How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)
- How to fill a GAMSDatabase by reading from MS Access (Transport9)
- How to fill a GAMSDatabase by reading from MS Excel (Transport10)
- How to create and use a save/restart file (Transport11)

7.8.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing Transport1.exe with C:/GAMS/win64/46.4 we use the 64-bit version of GAMS 46.4 to run `Transport1` even if our default GAMS system might be a different one. This is managed by the following code:

```
...
GAMSWorkspace ws;
if (Environment.GetCommandLineArgs().Length > 1)
  ws = new GAMSWorkspace(systemDirectory: Environment.GetCommandLineArgs()[1]);
else
  ws = new GAMSWorkspace();
...
```

Remember that the bitness of the GAMS system has to match the bitness of your .NET program.

7.8.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented .NET API offers much more than exchanging data between .NET and GDX, a common use case is the export and import of GDX files. The central class for this purpose is GAMSDatabase. We assume that the data to be exported is available in .NET data structures.

```
...
List<string> plants = new List<string>()
{
  "Seattle", "San-Diego"
};
List<string> markets = new List<string>()
{
  "New-York", "Chicago", "Topeka"
};
Dictionary<string, double> capacity = new Dictionary<string, double>()
{
  { "Seattle", 350.0 }, { "San-Diego", 600.0 }
```

```
};
Dictionary<string, double> demand = new Dictionary<string, double>()
{
  { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
Dictionary<Tuple<string, string>, double> distance = new Dictionary<Tuple<string, string>, double>()
{
  { new Tuple<string,string> ("Seattle",   "New-York"), 2.5 },
  { new Tuple<string,string> ("Seattle",   "Chicago"),  1.7 },
  { new Tuple<string,string> ("Seattle",   "Topeka"),   1.8 },
  { new Tuple<string,string> ("San-Diego", "New-York"), 2.5 },
  { new Tuple<string,string> ("San-Diego", "Chicago"),  1.8 },
  { new Tuple<string,string> ("San-Diego", "Topeka"),   1.4 }
};
...
```

Different GAMS symbols are represented using different .NET data structures. The data for the GAMS sets is represented using lists of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by dictionaries (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with .NET data structures.

A new GAMSDatabase instance can be created using GAMSWorkspace.AddDatabase.

```
...
// prepare a GAMSDatabase with data from the C# data structures
GAMSDatabase db = ws.AddDatabase();
...
```

We start adding GAMS sets using the method `GAMSDatabase.AddSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A foreach-loop iterates through `plants` and adds new records to the recently created GAMSSet instance `i` using `GAMSSet.AddRecord`.

```
...
// add 1-dimensional set 'i' with explanatory text 'canning plants' to the GAMSDatabase
GAMSSet i = db.AddSet("i", 1, "canning plants");
foreach (string p in plants)
  i.AddRecord(p);
...
```

GAMSParameter instances can be added by using the method `GAMSDatabase.AddParameter`. It has the same signature as `GAMSDatabase.AddSet`. Anyhow, in this example we use an overload of the method which takes a list of GAMSSet instances instead of the dimension for creating a parameter with domain information.

```
...
// add parameter 'a' with domain 'i'
GAMSParameter a = db.AddParameter("a", "capacity of plant i in cases", i);
foreach (string p in plants)
  a.AddRecord(p).Value = capacity[p];
...
```

As soon as all data is prepared in the GAMSDatabase, the method GAMSDatabase.Export can be used to create a GDX file.

```
...
// export the GAMSDatabase to a GDX file with name 'data.gdx' located in the 'workingDirectory' of the GAMSWorkspace
db.Export("data.gdx");
...
```

7.8.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using `GAMSWorkspace.AddDatabaseFromGDX`. The method takes a path to a GDX file and creates a GAMSDatabase instance.

```
...
// add a new GAMSDatabase and initialize it from the GDX file just created
GAMSDatabase db2 = ws.AddDatabaseFromGDX("data.gdx");
...
```

Reading the data from the GAMSSet `i` into a list can be done as follows:

```
...
// read data from symbols into .NET data structures
List<string> plants2 = new List<string>();
foreach (GAMSSetRecord item in db2.GetSet("i"))
```

```
  plants2.Add(item.Key(0));
...
```

A new list `plants2` is created. `i` is retrieved by calling `GAMSDatabase.GetSet` on `db2`. The returned GAMSSet object can be iterated using a foreach-loop to access the records of the set. Each record is of type GAMSSetRecord and can be asked for its keys.

You can do the same for GAMSParameters. Instead of creating a list, we want to have the data in the form of a dictionary. GAMSParameterRecords can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter `a` into a `Dictionary<string, double>`.

```
...
Dictionary<string, double> capacity2 = new Dictionary<string, double>();
foreach (GAMSParameterRecord item in db2.GetParameter("a"))
  capacity2.Add(item.Key(0), item.Value);
...
```

For multi dimensional symbols, we choose the dictionary keys to be tuples instead of string.

```
...
Dictionary<Tuple<string, string>, double> distance2 = new Dictionary<Tuple<string, string>, double>();
foreach (GAMSParameterRecord item in db2.GetParameter("d"))
  distance2.Add(new Tuple<string, string>(item.Key(0), item.Key(1)), item.Value);
...
```

7.8.3.4 How to run a GAMSJob from file (Transport1)

Here we load the model `trnsport` from the GAMS Model Library. In doing so it is made available in the current working directory and can be loaded by the `GAMSWorkspace.AddJobFromFile` Method afterwards. Apparently this method also works with any other gms file you might have created on your own as long as it is located in the current working directory. Then the GAMSJob `t1` is defined from that file and run by the `GAMSJob.Run` method.

The following lines create the solution output and illustrate the usage of the `GAMSJob.OutDB` property to get access to the `GAMSDatabase` created by the `Run` method. To retrieve the content of variable `x` we use the `GAMSVariableRecord` class and the `GAMSDatabase.GetVariable` method.

```
...
ws.GamsLib("trnsport");
// create a GAMSJob from file and run it with default settings
GAMSJob t1 = ws.AddJobFromFile("trnsport.gms");
t1.Run();
Console.WriteLine("Ran with Default:");
foreach (GAMSVariableRecord rec in t1.OutDB.GetVariable("x"))
  Console.WriteLine("x(" + rec.Keys[0] + "," + rec.Keys[1] + "): level=" + rec.Level + " marginal=" + rec.Marginal);
...
```

7.8.3.5 How to specify the solver (Transport1)

The solver can be specified via the GAMSOptions class and the `GAMSWorkspace.AddOptions` method. The `GAMSOptions.AllModelTypes` property sets `xpress` as default solver for all model types which the solver can handle.

```
...
// run the job again with another solver
using (GAMSOptions opt = ws.AddOptions())
{
   opt.AllModelTypes = "xpress";
   t1.Run(opt);
}
...
```

7.8.3.6 How to run a job with a solver option file and capture its log output (Transport1)

At first we use the StreamWriter to create the file `xpress.opt` which will be used as solver option file and is stored in the current working directory. Then we write `algorithm=barrier` to this file and specify the barrier algorithm as algorithm before we close the solver option file. We choose xpress as solver just like in the preceding example and set the `GAMSOptions.OptFile` Property to 1 to tell the solver to look for a solver option file. In addition, we specify the argument `output` in order to write the log of the GAMSJob into the file `transport1_xpress.log`.

```
...
using (StreamWriter optFile = new StreamWriter(Path.Combine(ws.WorkingDirectory, "xpress.opt")))
using (TextWriter logFile = new StreamWriter(Path.Combine(ws.WorkingDirectory, "transport1_xpress.log")))
using (GAMSOptions opt = ws.AddOptions())
{
   optFile.WriteLine("algorithm=barrier");
   optFile.Close();
   opt.AllModelTypes = "xpress";
   opt.OptFile = 1;
   t1.Run(opt, output: logFile);
}
...
```

Instead of writing the log output to a file, it can be written into any `TextWriter`. In order to write the log directly to `Console.Out`, we can use the following code:

```
...
   t1.Run(opt, output: Console.Out);
...
```

7.8.3.7 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings `GetDataText` and `GetModelText` are using GAMS syntax.

At first we write an include file `tdata.gms` that contains the data but not the model text:

```
...
using (StreamWriter writer = new StreamWriter(ws.WorkingDirectory + Path.DirectorySeparatorChar + "tdata.gms"))
{
   writer.Write(GetDataText());
}
...
```

Afterwards we create a `GAMSJob` using the `GAMSWorkspace.AddJobFromString` method. The `GAMSOptions.Defines` field is used like the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line.

```
...
using (GAMSOptions opt = ws.AddOptions())
{
  GAMSJob t2 = ws.AddJobFromString(GetModelText());
   opt.Defines.Add("incname", "tdata");
   t2.Run(opt);
   ...
}
...
```

Note that the string GetModelText contains the following lines to read in the data.

```
...
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

7.8.3.8 How to read data from string and export to GDX (Transport3)

We read the data from the string `GetDataText` as we did in the preceding example. Note that this contains no solve statement but only data definition in GAMS syntax. By running the corresponding `GAMSJob` a `GAMSDatabase` is created that is available via the `GAMSJob.OutDb` property. We can use the `GAMSDatabase.Export` method to write the content of this database to a gdx file `tdata.gdx`.

```
...
GAMSJob t3 = ws.AddJobFromString(GetDataText());
t3.Run();
t3.OutDB.Export(ws.WorkingDirectory + Path.DirectorySeparatorChar + "tdata.gdx");
...
```

7.8.3.9 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in `Transport2` - How to use include files (Transport2).

```
...
t3 = ws.AddJobFromString(GetModelText());
using (GAMSOptions opt = ws.AddOptions())
{
   opt.Defines.Add("gdxincname", "tdata");
   opt.AllModelTypes = "xpress";
   t3.Run(opt);
   ...
}
...
```

Note that there are some changes in `GetModelText` due to the usage of a GDX file instead of an include file.

```
...
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
...
```

7.8.3.10 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two GAMSJobs `t3a` and `t3b` where the first one contains only the data and the second one contains only the model without data. After running `t3a` the corresponding `OutDB` can be read in directly just like a gdx file. Note that the database needs to be passed to the `GAMSJob.Run` method as additional argument.

```
...
using (GAMSOptions opt = ws.AddOptions())
{
   GAMSJob t3a = ws.AddJobFromString(GetDataText());
   GAMSJob t3b = ws.AddJobFromString(GetModelText());
   t3a.Run();
   opt.Defines.Add("gdxincname", t3a.OutDB.Name);
   opt.AllModelTypes = "xpress";
   t3b.Run(opt, t3a.OutDB);
   ...
}
...
```

7.8.3.11 How to define data using C# data structures (Transport4)

We use the `List<T>` class, the `Dictionary<TKey, TValue>` class and the `Tuple<T1, T2>` class to define C# data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```
...
List<string> plants = new List<string>()
{
  "Seattle", "San-Diego"
};
List<string> markets = new List<string>()
{
  "New-York", "Chicago", "Topeka"
};
Dictionary<string, double> capacity = new Dictionary<string, double>()
{
   { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
Dictionary<string, double> demand = new Dictionary<string, double>()
{
   { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
Dictionary<Tuple<string,string>, double> distance = new Dictionary<Tuple<string,string>, double>()
{
   { new Tuple<string,string> ("Seattle",   "New-York"), 2.5 },
   { new Tuple<string,string> ("Seattle",   "Chicago"),  1.7 },
   { new Tuple<string,string> ("Seattle",   "Topeka"),   1.8 },
   { new Tuple<string,string> ("San-Diego", "New-York"), 2.5 },
   { new Tuple<string,string> ("San-Diego", "Chicago"),  1.8 },
   { new Tuple<string,string> ("San-Diego", "Topeka"),   1.4 }
};
...
```

7.8.3.12 How to prepare a GAMSDatabase from C# data structures (Transport4)

At first we create an empty GAMSDatabase `db` using the `GAMSWorkspace.AddDatabase` method. Afterwards we prepare the database. To add a set to the database we use the `GAMSSet` class and the `GAMSDatabase.AddSet` method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our C# data structure and add them by using the `GAMSSet.AddRecord` method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2.

The `GAMSJob` can be run like explained in the preceding example about implicit database communication.

```
...
GAMSDatabase db = ws.AddDatabase();
GAMSSet i = db.AddSet("i", 1, "canning plants");
foreach (string p in plants)
   i.AddRecord(p);
...
GAMSParameter a = db.AddParameter("a", 1, "capacity of plant i in cases");
foreach (string p in plants)
   a.AddRecord(p).Value = capacity[p];
...
GAMSParameter d = db.AddParameter("d", 2, "distance in thousands of miles");
foreach(Tuple<string,string> t in distance.Keys)
   d.AddRecord(t.Item1,t.Item2).Value = distance[t];
GAMSParameter f = db.AddParameter("f", 0, "freight in dollars per case per thousand miles");
f.AddRecord().Value = 90;
// run a job using data from the created GAMSDatabase
GAMSJob t4 = ws.AddJobFromString(GetModelText());
using (GAMSOptions opt = ws.AddOptions())
{
   opt.Defines.Add("gdxincname", db.Name);
   opt.AllModelTypes = "xpress";
   t4.Run(opt, db);
   ...
}
...
```

7.8.3.13 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following two lines of code conduct several operations. While the first line simply creates a GAMS Checkpoint, the second one uses the `GAMSWorkspace.AddJobFromString` method to create a `GAMSJob` containing the model text and data but no solve statement. In contrast to the preceding examples it runs the job immediately using the `GAMSJob.Run` method. Furthermore, it passes an additional checkpoint argument to the Run method. That means the GAMSCheckpoint `cp` captures the state of the `GAMSJob`.

```
...
GAMSCheckpoint cp = ws.AddCheckpoint();
...
ws.AddJobFromString(GetModelText()).Run(cp);
...
```

This creates the same checkpoint as for example the following code snippet:

```
GAMSCheckpoint cp = ws.AddCheckpoint();
GAMSJob t5a = ws.AddJobFromString(GetModelText());
t5a.Run(cp);
```

7.8.3.14 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `GetModelText()` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```
...
Scalar bmult  demand multiplier /1/;
...
demand(j) ..   sum(i, x(i,j))  =g=  bmult*b(j) ;
...
Scalar ms 'model status', ss 'solve status';
...
```

We create a list with eight different values for this demand multiplier.

```
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
```

For each entry of that list we create a GAMSJob `t5` using the `GAMSWorkspace.AddJobFromString` method. Besides the string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a `GAMSJob` combined from the checkpoint plus the content provided by the string.

We run the `GAMSJob` and echo some interesting data from the `OutDB` using the `GAMSDatabase.GetParameter` and `GAMSDatabase.GetVariable` methods, the `GAMSParameter.FindRecord` and `GAMSVariable.FindRecord` methods plus the `GAMSParameterRecord.Value` property and the `GAMSVariableRecord.Level` property.

```
...
foreach (double b in bmultlist)
{
  GAMSJob t5 = ws.AddJobFromString("bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat;
      ss=transport.solvestat;", cp);
  t5.Run();
  Console.WriteLine("Scenario bmult=" + b + ":");
  Console.WriteLine("  Modelstatus: " + t5.OutDB.GetParameter("ms").FindRecord().Value);
  Console.WriteLine("  Solvestatus: " + t5.OutDB.GetParameter("ss").FindRecord().Value);
  Console.WriteLine("  Obj: " + t5.OutDB.GetVariable("z").FindRecord().Level);
}
...
```

Note

Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

7.8.3.15 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)

This example illustrates how to run the jobs known from `Transport5` in parallel. We initialize the GAMSCheckpoint `cp` and introduce a demand multiplier as we did before :

```
...
GAMSCheckpoint cp = ws.AddCheckpoint();
ws.AddJobFromString(GetModelText()).Run(cp);
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
...
```

Furthermore, we introduce a new object ioMutex that will be used to avoid mixed up output from the parallel jobs. For each element `b` from the list of demand multipliers we call a delegate of the function RunScenario. Note that these calls are parallel!

```
...
// run multiple parallel jobs using the created GAMSCheckpoint
Object ioMutex = new Object();
System.Threading.Tasks.Parallel.ForEach(bmultlist, delegate(double b) { RunScenario(ws, cp, ioMutex, b); });
...
```

In function RunScenario a `GAMSJob` is created and run just like in the preceding example of `Transport5`. The output section is also the same except for the fact that it is marked as critical section by the `lock` keyword. That means the delegates of `RunScenario` are running in parallel but the output block of different delegates cannot be executed in parallel since it is 'locked' by the same object `ioMutex` for all delegates.

```
...
static void RunScenario(GAMSWorkspace ws, GAMSCheckpoint cp, object ioMutex, double b)
{
  GAMSJob t6 = ws.AddJobFromString("bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat;
      ss=transport.solvestat;", cp);
  t6.Run();
  // we need to make the ouput a critical section to avoid messed up report information
  lock (ioMutex)
  {
      Console.WriteLine("Scenario bmult=" + b + ":");
      Console.WriteLine("  Modelstatus: " + t6.OutDB.GetParameter("ms").FindRecord().Value);
      Console.WriteLine("  Solvestatus: " + t6.OutDB.GetParameter("ss").FindRecord().Value);
      Console.WriteLine("  Obj: " + t6.OutDB.GetVariable("z").FindRecord().Level);
   }
}
```

```
...
```

While the output in `Transport5` is strictly ordered subject to the order of the elements of `bmultlist`, in `Transport6` the output blocks might change their order but the blocks describing one scenario are still appearing together due to the `lock` keyword.

If you want a further impression of the impact of the `lock` keyword, just rerun `Transport6` but comment out the `lock` as follows and compare the output.

```
...
   //lock (ioMutex)
   //{
      Console.WriteLine("Scenario bmult=" + b + ":");
      Console.WriteLine("  Modelstatus: " + t6.OutDB.GetParameter("ms").FindRecord().Value);
      Console.WriteLine("  Solvestatus: " + t6.OutDB.GetParameter("ss").FindRecord().Value);
      Console.WriteLine("  Obj: " + t6.OutDB.GetVariable("z").FindRecord().Level);
   //}
...
```

7.8.3.16 How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)

In `Transport7` the usage of **GAMS::GAMSModelInstance** is demonstrated.

At first checkpoint `cp` is created as in the preceding examples. Then we create the GAMSModelInstance `mi` using the `GAMSCheckpoint.AddModelInstance` method. Note that the `GAMSJob` again contains no solve statement and the demand multiplier is already included with default value 1.

```
...
GAMSCheckpoint cp = ws.AddCheckpoint();
GAMSJob t7 = ws.AddJobFromString(GetModelText());
t7.Run(cp);
GAMSModelInstance mi = cp.AddModelInstance();
...
```

7.8.3.17 How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)

A `GAMSModelInstance` uses a `SyncDB` to maintain the data. We define `bmult` as `GAMSParameter` using the `GAMSDatabase.AddParameter` method and specify `gurobi` as solver. Afterwards the `GAMSModelInstance` is instantiated with arguments opt and GAMSModifier `bmult`. The `GAMSModifier` means that `bmult` is modifiable while all other parameters, variables and equations of ModelInstance mi stay unchanged. We use the `GAMSParameter.AddRecord` method to assign a value to `bmult` that can be varied afterwards using the `GAMSParameter.FirstRecord` method to reproduce our well-known example with different demand multipliers.

```
...
GAMSParameter bmult = mi.SyncDB.AddParameter("bmult", 0, "demand multiplier");
GAMSOptions opt = ws.AddOptions();
opt.AllModelTypes = "gurobi";
mi.Instantiate("transport use lp min z", opt, new GAMSModifier(bmult));
bmult.AddRecord().Value = 1.0;
double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
foreach (double b in bmultlist)
{
   bmult.FirstRecord().Value = b;
   mi.Solve();
   Console.WriteLine("Scenario bmult=" + b + ":");
   Console.WriteLine("  Modelstatus: " + mi.ModelStatus);
   Console.WriteLine("  Solvestatus: " + mi.SolveStatus);
   Console.WriteLine("  Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
}
...
```

7.8.3.18 How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)

We create a `GAMSModelInstance` just like in the next to last example. We define `x` as `GAMSVariable` and its upper bound as GAMSParameter xup. At the following instantiate method `GAMSModifier` has three arguments. The first one says that `x` is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the `GAMSParameter` that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```
...
mi = cp.AddModelInstance();
GAMSVariable x = mi.SyncDB.AddVariable("x", 2, VarType.Positive, "");
GAMSParameter xup = mi.SyncDB.AddParameter("xup", 2, "upper bound on x");
mi.Instantiate("transport use lp min z", modifiers: new GAMSModifier(x,UpdateAction.Upper,xup));
foreach (GAMSSetRecord i in t7.OutDB.GetSet("i"))
  foreach (GAMSSetRecord j in t7.OutDB.GetSet("j"))
  {
       xup.Clear();
       xup.AddRecord(i.Keys[0],j.Keys[0]).Value = 0;
       mi.Solve();
       Console.WriteLine("Scenario link blocked: " + i.Keys[0]  + " - " + j.Keys[0]);
       Console.WriteLine("  Modelstatus: " + mi.ModelStatus);
       Console.WriteLine("  Solvestatus: " + mi.SolveStatus);
       Console.WriteLine("  Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
   }
...
```

7.8.3.19 How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)

We initialize a GAMSCheckpoint `cp` from a `GAMSJob`. Then we define a queue that represents the different values of the demand multiplier. A queue follows the first-in-first-out-principle. The objects `queueMutex` and `ioMutex` are used later to avoid messed up output. Then we call two delegates of the function ScenSolve in parallel that get the same queue as argument.

```
...
GAMSCheckpoint cp = ws.AddCheckpoint();
ws.AddJobFromString(GetModelText()).Run(cp);
Queue<double> bmultQueue = new Queue<double>(new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 });
Object queueMutex = new Object();
Object ioMutex = new Object();
Parallel.For(0, 2, delegate(int i) { ScenSolve(ws, cp, bmultQueue, queueMutex, ioMutex); });
...
```

In function `ScenSolve` we create and instantiate a `GAMSModelInstance` as in the preceding examples and make `bmult` modifiable. The two delegates of the function extract the elements of the queue using `bmultQueue.Dequeue`. Note that we chose `cplex` as solver because it is thread safe (gurobi would also be possible). Once the queue is empty the loop terminates.

```
private static void ScenSolve(GAMSWorkspace ws, GAMSCheckpoint cp, Queue<double> bmultQueue, Object queueMutex, Object
      ioMutex, int i)
{
  GAMSModelInstance mi = cp.AddModelInstance();
  GAMSParameter bmult = mi.SyncDB.AddParameter("bmult", 0, "demand multiplier");
  GAMSOptions opt = ws.AddOptions();
  opt.AllModelTypes = "cplex";
  mi.Instantiate("transport use lp min z", opt, new GAMSModifier(bmult));
  bmult.AddRecord().Value = 1.0;
  while (true)
  {
      double b;
      // dynamically get a bmult value from the queue instead of passing it to the different threads at creation time
      lock (queueMutex)
      {
          if(0 == bmultQueue.Count)
              return;
          b = bmultQueue.Dequeue();
       }
      bmult.FirstRecord().Value = b;
      mi.Solve();
      // we need to make the output a critical section to avoid messed up report informations
```

```
      lock (ioMutex)
      {
          Console.WriteLine("Scenario bmult=" + b + ":");
          Console.WriteLine("  Modelstatus: " + mi.ModelStatus);
          Console.WriteLine("  Solvestatus: " + mi.SolveStatus);
          Console.WriteLine("  Obj: " + mi.SyncDB.GetVariable("z").FindRecord().Level);
       }
   }
}
```

7.8.3.20 How to fill a GAMSDatabase by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a `GAMSDatabase`. We call a function `ReadFromAccess` that finally returns a `GAMSDatabase` as shown below.

```
...
GAMSDatabase db = ReadFromAccess(ws);
...
```

The data we are going to read can be found in <`GAMS system directory`>\`apifiles\Data\transport.accdb`. It might be helpful to open this file for a better understanding. The function begins with the creation of an empty `GAMSDatabase`. Afterwards we connect to the MS Access database by first specifying the provider and then defining the aforementioned file as data source. The `OleDbConnection` represents an open connection to a data source. The following lines initialize the connection and use a try and a catch block for potential exceptions, i.e. errors that occur during application execution. To finally read in GAMS sets and parameters we call the functions `ReadSet` and `ReadParameter`. We will use several classes of the `System.Data.OleDb` namespace that are documented here. Furthermore, we recommend the OLE DB Tutorial.

```
static GAMSDatabase ReadFromAccess(GAMSWorkspace ws)
{
  GAMSDatabase db = ws.AddDatabase();
  // connect to database
  string strAccessConn = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=..\..\..\..\Data\transport.accdb";
  OleDbConnection connection = null;
  try
  {
      connection = new OleDbConnection(strAccessConn);
  }
  catch (Exception ex)
  {
      Console.WriteLine("Error: Failed to create a database connection. \n{0}", ex.Message);
      Environment.Exit(1);
   }
   // read GAMS sets
   ReadSet(connection, db, "SELECT Plant FROM Plant", "i", 1, "canning plants");
   ReadSet(connection, db, "SELECT Market FROM Market", "j", 1, "markets");
   // read GAMS parameters
   ReadParameter(connection, db, "SELECT Plant,Capacity FROM Plant", "a", 1, "capacity of plant i in cases");
   ReadParameter(connection, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases");
   ReadParameter(connection, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in thousands of miles");
   return db;
}
```

The function `ReadSet` incorporates a try and a catch block. In the try block we prepare the reading from the MS Access file. Then we add a set symbol to the `GAMSDatabase` that is filled with the data from the MS Access file afterwards. The function `ReadParameter` works quite similar.

```
static void ReadSet(OleDbConnection connect, GAMSDatabase db, string strAccessSelect, string setName, int setDim, string
      setExp = "")
{
  try
  {
      OleDbCommand cmd = new OleDbCommand(strAccessSelect, connect);
      connect.Open();
      OleDbDataReader reader = cmd.ExecuteReader();
      if (reader.FieldCount != setDim)
      {
          Console.WriteLine("Number of fields in select statement does not match setDim");
          Environment.Exit(1);
      }
      GAMSSet i = db.AddSet(setName, setDim, setExp);
      string[] keys = new string[setDim];
      while (reader.Read())
      {
          for (int idx = 0; idx < setDim; idx++)
               keys[idx] = reader.GetString(idx);
          i.AddRecord(keys);
```

```
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error: Failed to retrieve the required data from the DataBase.\n{0}", ex.Message);
        Environment.Exit(1);
    }
    finally
    {
        connect.Close();
    }
}
```

Once we read in all the data we can create a `GAMSJob` from the `GAMSDatabase` and run it as usual.

7.8.3.21 How to fill a GAMSDatabase by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from <`GAMS system directory`>\`apifiles\Data\transport.xlsx`. The model is given as string without data like in in many examples before. At first we have to add

```
using Excel = Microsoft.Office.Interop.Excel;
```

to the preamble to be able to use the `Microsoft.Office.Interop.Excel` namespace. Then we define `excelApp` as Excel application using the implicit variable type var, use the aforementioned file as workbook wb and define the Excel range that can represent a cell, a row, a column, a selection of cells containing one or more contiguous blocks of cells, or a 3-D range.

```
...
var excelApp = new Excel.Application();
Excel.Workbook wb = excelApp.Workbooks.Open(Directory.GetCurrentDirectory() + @"\..\..\..\..\Data\transport.xlsx");
Excel.Range range;
...
```

The following lines address the different worksheets and read in the contained data. Afterwards we make an errorcheck to ensure that the number of plants and markets is the same in all worksheets.

```
...
Excel.Worksheet capacity = (Excel.Worksheet)wb.Worksheets.get_Item("capacity");
range = capacity.UsedRange;
Array capacityData = (Array)range.Cells.Value;
int iCount = capacity.UsedRange.Columns.Count;
Excel.Worksheet demand = (Excel.Worksheet)wb.Worksheets.get_Item("demand");
range = demand.UsedRange;
Array demandData = (Array)range.Cells.Value;
int jCount = range.Columns.Count;
Excel.Worksheet distance = (Excel.Worksheet)wb.Worksheets.get_Item("distance");
range = distance.UsedRange;
Array distanceData = (Array)range.Cells.Value;
// number of markets/plants have to be the same in all spreadsheets
Debug.Assert((range.Columns.Count - 1) == jCount && (range.Rows.Count - 1) == iCount,
             "Size of the spreadsheets doesn't match");
wb.Close();
...
```

If you have problems to see through the steps above, adding the following output section right after `wb.Close()` might be helpful to understand where which data is stored.

```
...
for (int ii = 1; ii <= iCount; ii++)
{
  Console.WriteLine("capacityData(1, " + ii + ") = " + capacityData.GetValue(1, ii));
  Console.WriteLine("capacityData(2, " + ii + ") = " + capacityData.GetValue(2, ii));
}
for (int jj = 1; jj <= jCount; jj++)
{
  Console.WriteLine("demandData(1, " + jj + ") = " + demandData.GetValue(1, jj));
  Console.WriteLine("demandData(2, " + jj + ") = " + demandData.GetValue(2, jj));
}
for (int ii = 1; ii <= iCount; ii++)
{
  for (int jj = 1; jj <= jCount; jj++)
      Console.WriteLine("distanceData("+ (ii+1) + "," + (jj+1) +") = " + distanceData.GetValue(ii+1, jj+1));
}
...
```

Now we can create the `GAMSWorkspace` as usual and afterwards create a `GAMSDatabase` and fill it with the workbook data as follows:

```
...
GAMSDatabase db = ws.AddDatabase();
GAMSSet Set1 = db.AddSet("i", 1, "Plants");
GAMSSet j = db.AddSet("j", 1, "Markets");
GAMSParameter capacityParam = db.AddParameter("a", 1, "Capacity");
GAMSParameter demandParam = db.AddParameter("b", 1, "Demand");
GAMSParameter distanceParam = db.AddParameter("d", 2, "Distance");
for (int ic = 1; ic <= iCount; ic++)
{
   Set1.AddRecord((string)capacityData.GetValue(1, ic));
   capacityParam.AddRecord((string)capacityData.GetValue(1, ic)).Value = (double)capacityData.GetValue(2, ic);
}
for (int jc = 1; jc <= jCount; jc++)
{
   j.AddRecord((string)demandData.GetValue(1, jc));
   demandParam.AddRecord((string)demandData.GetValue(1, jc)).Value = (double)demandData.GetValue(2, jc);
   for (int ic = 1; ic <= iCount; ic++)
   {
       distanceParam.AddRecord((string)distanceData.GetValue(ic + 1, 1), (string)distanceData.GetValue(1, jc + 1)).Value =
      (double)distanceData.GetValue(ic + 1, jc + 1);
   }
}
...
```

Note that we can name sets and parameters just like in the database but we don't have to. Now we finally set up the database and can run our `GAMSJob` as usual.

```
...
using (GAMSOptions opt = ws.AddOptions())
{
   GAMSJob t10 = ws.AddJobFromString(GetModelText());
   opt.Defines.Add("gdxincname", db.Name);
   opt.AllModelTypes = "xpress";
   t10.Run(opt, db);
   foreach (GAMSVariableRecord rec in t10.OutDB.GetVariable("x"))
      Console.WriteLine("x(" + rec.Keys[0] + "," + rec.Keys[1] + "): level=" + rec.Level + " marginal=" + rec.Marginal);
}
...
```

7.8.3.22 How to create and use a save/restart file (Transport11)

In `Transport11` we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a `GAMSCheckpoint`.

In the main function we start with the creation of a folder called tmp that internally is denoted as `wDir`. Then we call the function `CreateSaveRestart`.

```
...
string wDir = Path.Combine(".", "tmp");
CreateSaveRestart(Path.Combine(wDir, "tbase"));
...
```

In function `CreateSaveRestart` we choose the path of the file given as argument as working directory, so it should be `wDir`. Then we create a `GAMSJob` from a string. Note that the string given via `GetBaseModelText()` contains the basic definitions of sets without giving them a content (that is what $onempty is used for). Afterwards we specify a `GAMSOption` to only compile the job but do not execute it. Then we create a checkpoint `cp` that is initialized by the following run of the `GAMSJob` and stored in the file given as argument to the function. This becomes possible because the `AddCheckpoint` method accepts identifiers as well as file names as argument.

```
static void CreateSaveRestart(string cpFileName)
{
  GAMSWorkspace ws;
  if (Environment.GetCommandLineArgs().Length > 1)
       ws = new GAMSWorkspace(workingDirectory: Path.GetDirectoryName(cpFileName), systemDirectory:
      Environment.GetCommandLineArgs()[1]);
  else
       ws = new GAMSWorkspace(workingDirectory: Path.GetDirectoryName(cpFileName));
  GAMSJob j1 = ws.AddJobFromString(GetBaseModelText());
  GAMSOptions opt = ws.AddOptions();
  opt.Action = GAMSOptions.EAction.CompileOnly;
  GAMSCheckpoint cp = ws.AddCheckpoint(Path.GetFileName(cpFileName));
  j1.Run(opt, cp);
  opt.Dispose();
}
```

So what you should keep in mind before we return to further explanations of the main function is, that now the file tbase in folder `tmp` which is denoted as `wDir` contains a checkpoint. Now in the main function we define some data using C# data structures as we already did in `Transport4` before we create the `GAMSWorkspace` with working directory `wDir`.

```
...
GAMSWorkspace ws;
if (Environment.GetCommandLineArgs().Length > 1)
   ws = new GAMSWorkspace(workingDirectory: wDir, systemDirectory: Environment.GetCommandLineArgs()[1]);
else
   ws = new GAMSWorkspace(workingDirectory: wDir);
...
```

Afterwards we set up the `GAMSDatabase` like we already did in `Transport4`. Once this is done we run a `GAMSJob` using this data plus the checkpoint stored in file tbase.

```
...
GAMSCheckpoint cpBase = ws.AddCheckpoint("tbase");
using (GAMSOptions opt = ws.AddOptions())
{
   GAMSJob t4 = ws.AddJobFromString(GetModelText(), cpBase);
   opt.Defines.Add("gdxincname", db.Name);
   opt.AllModelTypes = "xpress";
   t4.Run(opt, db);
   foreach (GAMSVariableRecord rec in t4.OutDB.GetVariable("x"))
      Console.WriteLine("x(" + rec.Keys[0] + "," + rec.Keys[1] + "): level=" + rec.Level + " marginal=" + rec.Marginal);
}
...
```

Note that the string from which we create job `t4` is different to the one used to prepare the checkpoint stored in tbase and is only responsible for reading in the data from the `GAMSDatabase` correctly. The entire model definition is delivered by the checkpoint `cpBase` which is equal to the one we saved in `tbase`.

7.9 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS C++ API. It allows the user to start working immediately with the API by providing a set of small examples based on the well-known transportation problem. These examples introduce several API features step by step. The GAMS distribution comes with a pre compiled binary version of the GAMS C++ API on Windows, macOS and Linux. Furthermore, the C++ API itself and the provided examples are published under MIT license and are hosted at the GAMS GitHub organization. Therefore it is possible to compile the library manually (e.g. if a certain compiler is required) The GAMS C++ API is implemented and compatible with C++17.

- Getting Started A quick introduction about how to create and configure a C++ project using Qt Creator
- Important Classes of the API Overview of some fundamental classes of the GAMS C++ API
- How to use API An extensive set of examples how to use API components

7.9.1 Getting Started

This section guides you through the basic steps of compiling and running a program from scratch using either Qt Creator, Microsoft Visual Studio or Xcode. If you want to have a look at the API itself you can directly start with the How to use API section instead. While this tutorial uses Qt Creator, Microsoft Visual Studio and Xcode, other IDE's can be used as well.

Note

When using the pre compiled binary distribution of the GAMS C++ API as done in this section, one needs to use a compiler that is compatible with the one that was used during compilation of the library. Depending on the platform, different versions of the pre build binaries can be found under <GAMS system directory>/apifiles/C++/lib. In GAMS versions before 25.0 the binaries can be found under <GAMS system directory>. The distributed libraries use the following compilers:

- Windows: MSVC2017 and MSVC2019
- MacOS (x86_64): GCC and Clang
- MacOS (ARM64): Clang
- Linux: GCC

Note that the compiler needs at least full C++17 support in order to be able to use the GAMS C++ API.

7.9.1.1 Getting Started using Qt Creator

After opening Qt Creator click on **File** > **New File or Project**. Choose **Non-Qt Project** > **Plain C++ Application** and click on **Choose...**. On the next page fill in the name and the location of the new project and click **Next**. As a build system we recommend using CMake, which should be the default for recent Qt Creator versions. Afterwards, select the kits you want to use for the project and click again on **Next**. If you use the precompiled library on Windows, you need to use MSVC2017 or MSVC2019. Confirm the creation of the new project by clicking on **Finish**.

As soon as the project is created open the file CMakeLists.txt located in the root folder of your project. Add the following lines to the file:

```
set(GAMS_INSTALLATION "C:/GAMS/41")
set(VSVERSION "vs2019" CACHE STRING "Visual Studio version")
include_directories("${GAMS_INSTALLATION}/apifiles/C/api"
                    "${GAMS_INSTALLATION}/apifiles/C++/api")
if (WIN32)
    target_link_libraries(${PROJECT_NAME} "${GAMS_INSTALLATION}/apifiles/C++/lib/${VSVERSION}/gamscpp.lib")
else()
    target_link_libraries(${PROJECT_NAME} "${GAMS_INSTALLATION}/apifiles/C++/lib/libgamscpp.so")
endif()
```

Note

Depending on your operating system and the location of your GAMS system you might need to adjust the path to GAMS and Visual Studio version. Furthermore, all pre build binaries are always located at <GAMS system directory>\apifiles\C++\lib, if GAMS 25.0 (or higher) is used. This location may contain additional sub directories, like <GAMS system directory>\apifiles\C++\lib\vs2019. In case a previous version of GAMS is used the binaries are located at <GAMS system directory>.

Save the changes, right click on the project and choose Run CMake.

The last step of creating a first small example is to write code that makes use of the GAMS C++ API. Open the file main.cpp and replace the content of the file with the following lines of code:

```
#include "gams.h"
#include <iostream>
using namespace gams;
using namespace std;
int main(int argc, char* argv[])
{
    GAMSWorkspace ws;
    ws.gamsLib("trnsport");
    // create a GAMSJob from file and run it with default settings
    GAMSJob t1 = ws.addJobFromFile("trnsport.gms");
    t1.run();
    for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
        cout << "x(" << rec.key(0) << "," << rec.key(1) << "):" << " level=" << rec.level() << " marginal=" <<
      rec.marginal() << endl;
}
```

Make sure that MSVC2019 64bit is chosen as kit using the Release mode.

Click on the play button below to run (and compile) the small example. If everything works, you should see the output of the example displaying results of the solved transportation problem.

7.9.1.2 Getting Started using Visual Studio 2019

After opening Visual Studio click on **File** > **New** > **Project ...**.

Choose **Console App** and fill in the name and the location of the project and click on **Ok**.

As soon as the project is loaded, open the **Configuration Manager**:

Change the **Active solution configuration** to **Release** and the **Active solution platform** to **x64**.

The next step is to configure the project to find the GAMS C++ API. Rigth click on the project and choose **Properties** from the context menue. Go to **Configuration Properties** > **C/C++** > **General** and add <`GAMS system directory`>\`apifiles\C++\api` to **Additional Include Directories**.

Select **Configuration Properties** > **C/C++** > **Precompiled Headers** on the left. On the right open the dropdown of **Precompiled Header** and select `Not Using Precompiled Headers`.

Go to **Configuration Properties** > **Linker** > **General** and add the apifiles path to the matching version of your Visual Studio (e.g. <`GAMS system directory`>\`apifiles\C++\lib\vs2017`) to **Additional Library Directories**:

Note

Please mind that the pre build binaries for older GAMS versions (before GAMS 25.0) are located at <`GAMS system directory`>.

Choose **Input** and add `gamscpp.lib;` to **Additional Dependencies**. Make sure that all previous entries including `%(AdditionalDependencies)` are present as well.

The last required configuration step is to add a post-build event that copies required files next to the generated executable. Click on **Build Events** > **Post-Build Event** and add the following code to **Command Line**. It is helpful to select <**Edit...**> from the dropdown box as there are two command lines. Note that you might need to adjust the used path depending on the location of your GAMS installation:

```
xcopy /Y "C:\gams\36\apifiles\C++\lib\vs2017\gamscpp.dll" "$(OutDir)"
```

Open the file `GAMSApplication.cpp` and replace its content by the following code:

```
#include "gams.h"
#include <iostream>
using namespace gams;
using namespace std;
int main(int argc, char* argv[])
{
    GAMSWorkspace ws;
    ws.gamsLib("trnsport");
    // create a GAMSJob from file and run it with default settings
    GAMSJob t1 = ws.addJobFromFile("trnsport.gms");
    t1.run();
    for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
        cout << "x(" << rec.key(0) << "," << rec.key(1) << "):" << " level=" << rec.level() << " marginal=" <<
      rec.marginal() << endl;
}
```

Click on **Debug** > **Start Without Debugging** in order to compile and run the example. If everything works, you should see the output of the example displaying results of the solved transportation problem.

7.9.1.3 Getting Started using Xcode

After starting Xcode select "Create a new Xcode project" in the welcome screen.

As a platform pick macOS and "Command Line Tool" as Application type.

Give your application a name, an organization identifier and select "C++" as language.

On the next screen pick a location for your new project.

In your project settings go to the "Build Phases" tab add the following entry for "Compile Sources":

- `/Library/Frameworks/GAMS.framework/Versions/Current/Resources/apifiles/C++/api`

Then, under "Link Binary With Libraries" add libgamscpp.dylib. On the system for x86_64, this library is shipped in two versions, a GCC and a Clang version:

- The GCC library can be found in the `/Library/Frameworks/GAMS.framework/Versions/Current/Resources/apif` directory.
- The Clang version is only supported from macOS 10.15 onwards and can be found at `/Library/Frameworks/GAMS.framework/Versions/Current/Resources/apifiles/C++/api/Clang`.

On the system for ARM64, only the Clang version is available and can be found at `/Library/Frameworks/GAMS.framework/`

Open the file `main.cpp` and replace its content by the following code:

```
#include "gams.h"
#include <iostream>
using namespace gams;
using namespace std;
int main(int argc, char* argv[])
{
    GAMSWorkspace ws;
    ws.gamsLib("trnsport");
    // create a GAMSJob from file and run it with default settings
    GAMSJob t1 = ws.addJobFromFile("trnsport.gms");
    t1.run();
    for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
        cout << "x(" << rec.key(0) << "," << rec.key(1) << "):" << " level=" << rec.level() << " marginal=" <<
      rec.marginal() << endl;
}
```

7.9.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the gams namespace. Their usage is demonstrated by an extensive set of examples.

- **gams** Namespace
- **gams::GAMSWorkspace** Class
- **gams::GAMSJob** Class
- **gams::GAMSDatabase** Class
- **gams::GAMSOptions** Class
- **gams::GAMSModelInstance** Class
- **gams::GAMSSymbol** Class

7.9.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the C++ API. <GAMS system directory>\apifiles\C++ contains several Visual Studio solutions for different versions of Visual Studio (e.g. examples-vs2017.sln, examples-vs2019.sln). The code snippets that are explained in this tutorial are taken from these examples. Furthermore there is a CMakeLists.txt file that can be used for building the examples using CMake.

Note

When using CMake on macOS the Clang library is used by default. There is an option to switch to the GCC library: Add -D USE-GCC="ON" to your CMake command.

The well-known transportation problem is discussed step by step and each example introduces new elements of the GAMS C++ API.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- How to choose the GAMS system (Transport1)
- How to export data to GDX (TransportGDX)
- How to import data from GDX (TransportGDX)
- How to run a GAMSJob from file (Transport1)
- How to specify the solver (Transport1)
- How to run a job with a solver option file (Transport1)
- How to use include files (Transport2)
- How to read data from string and export to GDX (Transport3)
- How to run a job using data from GDX (Transport3)
- How to run a job using implicit database communication (Transport3)
- How to define data using C++ data structures (Transport4)
- How to prepare a GAMSDatabase from C++ data structures (Transport4)
- How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)
- How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)
- How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)
- How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)
- How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)
- How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)
- How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)
- How to fill a GAMSDatabase by reading from MS Access (Transport9)
- How to fill a GAMSDatabase by reading from MS Excel (Transport10)
- How to create and use a save/restart file (Transport11)

7.9.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing Transport1.exe with C:/GAMS/46 we use the 64-bit version of GAMS 46.4 to run `Transport1` even if our default GAMS system might be a different one. This is managed by the following code:

```
...
GAMSWorkspaceInfo wsInfo;
if (argc > 1)
    wsInfo.setSystemDirectory(argv[1]);
GAMSWorkspace ws(wsInfo);
...
```

Remember that the bitness of the GAMS system has to match the bitness of your C++ program.

7.9.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented C++ API offers much more than exchanging data between C++ and GDX, a common use case is the export and import of GDX files. The central class for this purpose is GAMSDatabase. We assume that the data to be exported is available in C++ data structures.

```
...
vector<string> plants = {
    "Seattle", "San-Diego"
};
vector<string> markets = {
    "New-York", "Chicago", "Topeka"
};
map<string, double> capacity = {
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
map<string, double> demand = {
    { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
map<tuple<string, string>, double> distance = {
    { make_tuple("Seattle", "New-York"), 2.5 },
    { make_tuple("Seattle", "Chicago"), 1.7 },
    { make_tuple("Seattle", "Topeka"), 1.8 },
    { make_tuple("San-Diego", "New-York"), 2.5 },
    { make_tuple("San-Diego", "Chicago"), 1.8 },
    { make_tuple("San-Diego", "Topeka"), 1.4 }
};
...
```

Different GAMS symbols are represented using different C++ data structures. The data for the GAMS sets is represented using vectors of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by maps (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with C++ data structures.

A new GAMSDatabase instance can be created using GAMSWorkspace.addDatabase.

```
...
// create new GAMSDatabase instance
GAMSDatabase db = ws.addDatabase();
...
```

We start adding GAMS sets using the method `GAMSDatabase.addSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created GAMSSet instance `i` using `GAMSSet.addRecord`.

```
...
// add 1-dimensional set 'i' with explanatory text 'canning plants' to the GAMSDatabase
GAMSSet i = db.addSet("i", 1, "canning plants");
for (string p: plants)
    i.addRecord(p);
...
```

GAMSParameter instances can be added by using the method `GAMSDatabase.addParameter`. It has the same signature as `GAMSDatabase.addSet`. Anyhow, in this example we use an overload of the method

which takes a list of GAMSSet instances instead of the dimension for creating a parameter with domain information.

```
...
// add parameter 'a' with domain 'i'
GAMSParameter a = db.addParameter("a", "capacity of plant i in cases", i);
for (string p: plants)
    a.addRecord(p).setValue(capacity[p]);
...
```

As soon as all data is prepared in the GAMSDatabase, the method GAMSDatabase.doExport can be used to create a GDX file.

```
...
// export the GAMSDatabase to a GDX file with name 'data.gdx' located in the 'workingDirectory' of the GAMSWorkspace
db.doExport("data.gdx");
...
```

7.9.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using `GAMSWorkspace.addDatabaseFromGDX`. The method takes a path to a GDX file and creates a GAMSDatabase instance.

```
...
// add a new GAMSDatabase and initialize it from the GDX file just created
GAMSDatabase db2 = ws.addDatabaseFromGDX("data.gdx");
...
```

Reading the data from the GAMSSet `i` into a vector can be done as follows:

```
...
// read data from symbols into C++ data structures
vector<string> iNew;
for(GAMSSetRecord rec : db2.getSet("i"))
    iNew.push_back(rec.key(0));
...
```

A new vector `iNew` is created. `i` is retrieved by calling `GAMSDatabase.getSet` on `db2`. The returned GAMSSet object can be iterated using a for-loop to access the records of the set. Each record is of type GAMSSetRecord and can be asked for its keys.

You can do the same for GAMSParameters. Instead of creating a vector, we want to have the data in the form of a map. GAMSParameterRecords can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter `a` into a `map<string, double>`.

```
...
map<string, double> aNew;
for(GAMSParameterRecord rec : db2.getParameter("a"))
    aNew[rec.key(0)] = rec.value();
...
```

For multi dimensional symbols, we choose the map keys to be tuples instead of string. We access the individual keys by index and generate a tuple using `make_tuple`.

```
...
map<tuple<string, string>, double> dNew;
for(GAMSParameterRecord rec : db2.getParameter("d"))
    dNew[make_tuple(rec.key(0), rec.key(1))] = rec.value();
...
```

Scalars can be read into a variable of type double by accessing the value of the first and only record.

```
...
double fNew = db2.getParameter("f").firstRecord().value();
...
```

7.9.3.4 How to run a GAMSJob from file (Transport1)

At first we create a `GAMSWorkspace`. Afterwards we load the model `trnsport` from the GAMS Model Library. In doing so it is made available in the current working directory and can be loaded by the `GAMSWorkspace.AddJobFromFile` Method afterwards. Apparently this method also works with any other gms file you might have created on your own as long as it is located in the current working directory. Then the GAMSJob `t1` is defined from that file and run by the `GAMSJob.run` method. The following lines create the solution output and illustrate the usage of the `GAMSJob.outDB` method to get access to the `GAMSDatabase` created by the `run` method. To retrieve the content of variable `x` we use the `GAMSVariableRecord` class and the `GAMSDatabase.getVariable` method.

```
...
GAMSWorkspace ws;
ws.gamsLib("trnsport");
// create a GAMSJob from file and run it with default settings
GAMSJob t1 = ws.addJobFromFile("trnsport.gms");
// Default run
t1.run();
cout << "Ran with Defaults:" << endl;
for (GAMSVariableRecord rec : t1.outDB().getVariable("x"))
    cout << "x(" << rec.key(0) << "," << rec.key(1) << "):" << " level=" << rec.level() << " marginal=" << rec.marginal()
      << endl;
...
```

7.9.3.5 How to specify the solver (Transport1)

The solver can be specified via the GAMSOptions class and the `GAMSWorkspace.addOptions` method. The `GAMSOptions.setAllModelTypes` method sets `xpress` as default solver for all model types which the solver can handle.

```
...
// Run the job again with another solver
GAMSOptions opt = ws.addOptions();
opt.setAllModelTypes("xpress");
t1.run(opt);
...
```

7.9.3.6 How to run a job with a solver option file (Transport1)

At first create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. We choose xpress as solver just like in the preceding example and call `GAMSOptions.setOptFile` in order to tell the solver to look for a solver option file.

```
...
// Run the job with a solver option file
ofstream xpressopt(ws.workingDirectory() + cPathSep + "xpress.opt");
xpressopt << "algorithm=barrier" << endl;
xpressopt.close();
opt.setAllModelTypes("xpress");
opt.setOptFile(1);
t1.run(opt);
...
```

7.9.3.7 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings provided by the methods `getDataText` and `getModelText` are using GAMS syntax.

At first we write an include file `tdata.gms` that contains the data but not the model text:

```
...
ofstream tdata(ws.workingDirectory() + cPathSep + "tdata.gms");
tdata << getDataText();
tdata.close();
...
```

Afterwards we create a GAMSJob using the GAMSWorkspace.addJobFromString method. The GAMSOptions.setDefine method is used like the 'double dash' GAMS parameters, i.e. it corresponds to --incname=tdata on the command line.

```
...
GAMSOptions opt = ws.addOptions();
GAMSJob t2 = ws.addJobFromString(getModelText());
opt.setDefine("incname", "tdata");
t2.run(opt);
...
```

Note that the string provided by getModelText contains the following lines to read in the data.

```
...
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

7.9.3.8 How to read data from string and export to GDX (Transport3)

We read the data from the string provided by getDataText as we did in the preceding example. Note that this contains no solve statement but only data definition in GAMS syntax. By running the corresponding GAMSJob a GAMSDatabase is created that is available via the GAMSJob.outDb method. We can use the GAMSDatabase.doExport method to write the content of this database to a gdx file tdata.gdx.

```
...
// data from a string with GAMS syntax with explicit export to GDX file
GAMSJob t3 = ws.addJobFromString(getDataText());
t3.run();
t3.outDB().doExport(ws.workingDirectory() + cPathSep + "tdata.gdx");
...
```

7.9.3.9 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in Transport2 - How to use include files (Transport2).

```
...
t3 = ws.addJobFromString(getModelText());
GAMSOptions opt = ws.addOptions();
opt.setDefine("gdxincname", "tdata");
opt.setAllModelTypes("xpress");
t3.run(opt);
...
```

Note that there are some changes in getModelText due to the usage of a GDX file instead of an include file.

```
...
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
...
```

7.9.3.10 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two GAMSJobs t3a and t3b where the first one contains only the data and the second one contains only the model without data. After running t3a the corresponding outDB can be read in directly just like a gdx file. Note that the database needs to be passed to the GAMSJob.run method as additional argument.

```
...
GAMSJob t3a = ws.addJobFromString(getDataText());
GAMSJob t3b = ws.addJobFromString(getModelText());
t3a.run();
opt.setDefine("gdxincname", t3a.outDB().name());
t3b.run(opt, t3a.outDB());
...
```

7.9.3.11 How to define data using C++ data structures (Transport4)

We use the `vector<T>` and `map<Key, Value>` to define C++ data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```
...
vector<string> plants = {
    "Seattle", "San-Diego"
};
vector<string> markets = {
    "New-York", "Chicago", "Topeka"
};
map<string, double> capacity = {
    { "Seattle", 350.0 }, { "San-Diego", 600.0 }
};
map<string, double> demand = {
    { "New-York", 325.0 }, { "Chicago", 300.0 }, { "Topeka", 275.0 }
};
map<tuple<string, string>, double> distance = {
    { make_tuple("Seattle", "New-York"), 2.5 },
    { make_tuple("Seattle", "Chicago"), 1.7 },
    { make_tuple("Seattle", "Topeka"), 1.8 },
    { make_tuple("San-Diego", "New-York"), 2.5 },
    { make_tuple("San-Diego", "Chicago"), 1.8 },
    { make_tuple("San-Diego", "Topeka"), 1.4 }
};
...
```

7.9.3.12 How to prepare a GAMSDatabase from C++ data structures (Transport4)

At first we create an empty GAMSDatabase `db` using the `GAMSWorkspace.addDatabase` method. Afterwards we prepare the database. To add a set to the database we use the `GAMSSet` class and the `GAMSDatabase.addSet` method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our C++ data structure and add them by using the `GAMSSet.addRecord` method. We do pretty much the same thing for the parameters.

Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2 and that the scalars can be treated as parameter with dimension 0.

The `GAMSJob` can be run like explained in the preceding example about implicit database communication.

```
...
GAMSDatabase db = ws.addDatabase();
GAMSSet i = db.addSet("i", 1, "canning plants");
for (string p: plants)
    i.addRecord(p);
...
GAMSParameter a = db.addParameter("a", "capacity of plant i in cases", i);
for (string p: plants)
    a.addRecord(p).setValue(capacity[p]);
...
GAMSParameter d = db.addParameter("d", "distance in thousands of miles", i, j);
for (auto t : distance)
    d.addRecord(get<0>(t.first), get<1>(t.first)).setValue(t.second);
GAMSParameter f = db.addParameter("f", "freight in dollars per case per thousand miles");
f.addRecord().setValue(90);
// run a job using data from the created GAMSDatabase
GAMSJob t4 = ws.addJobFromString(getModelText());
GAMSOptions opt = ws.addOptions();
opt.setDefine("gdxincname", db.name());
opt.setAllModelTypes("xpress");
t4.run(opt, db);
...
```

7.9.3.13 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following two lines of code conduct several operations. While the first line simply creates a GAMS checkpoint, the second one uses the `GAMSWorkspace.addJobFromString` method to create a `GAMSJob` containing the model text and data but no solve statement. In contrast to the preceding examples it runs the job immediately using the `GAMSJob.run` method. Furthermore, it passes an additional checkpoint argument to the `run` method. That means the GAMSCheckpoint `cp` captures the state of the `GAMSJob`.

```
...
GAMSCheckpoint cp = ws.addCheckpoint();
...
ws.addJobFromString(getModelText()).run(cp);
...
```

7.9.3.14 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `getModelText()` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```
...
Scalar bmult  demand multiplier /1/;
...
demand(j) ..   sum(i, x(i,j))  =g=  bmult*b(j) ;
...
Scalar ms 'model status', ss 'solve status';
...
```

We create a `vector` with eight different values for this demand multiplier.

```
vector<double> bmultlist = { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
```

For each entry of the vector we create a GAMSJob `t5` using the `GAMSWorkspace.addJobFromString` method. Besides the string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a `GAMSJob` combined from the checkpoint plus the content provided by the string.

We run the `GAMSJob` and echo some interesting data from the `outDB` using the `GAMSDatabase.getParameter` and `GAMSDatabase.getVariable` methods, the `GAMSParameter.findRecord` and `GAMSVariable.findRecord` methods plus the `GAMSParameterRecord.value` and the `GAMSVariableRecord.level` methods.

```
...
for (double b : bmultlist)
{
    GAMSJob t5 = ws.addJobFromString("bmult=" + to_string(b) + "; solve transport min z use lp; ms=transport.modelstat;
      ss=transport.solvestat;", cp);
    t5.run();
    cout << "Scenario bmult=" << b << ":" << endl;
    cout << "  Modelstatus: " << t5.outDB().getParameter("ms").findRecord().value() << endl;
    cout << "  Solvestatus: " << t5.outDB().getParameter("ss").findRecord().value() << endl;
    cout << "  Obj: " << t5.outDB().getVariable("z").findRecord().level() << endl;
}
...
```

Note

> Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

7.9.3.15 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)

This example illustrates how to run the jobs we already know from `Transport5` in parallel. We create a GAMSCheckpoint `cp` and initialize it by running a `GAMSJob`. Furthermore we introduce a demand multiplier as we did before.

```
...
GAMSCheckpoint cp = ws.addCheckpoint();
ws.addJobFromString(getModelText()).run(cp);
vector<double> bmultlist = { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
...
```

Furthermore, we introduce a new object `ioMutex` that will be used to avoid mixed up output from the parallel jobs. For each element `b` from the vector of demand multipliers we create a thread executing the `runScenario` method.

```
...
// run multiple parallel jobs using the created GAMSCheckpoint
mutex ioMutex;
vector<thread> v;
for(double b : bmultlist)
    v.emplace_back([&ws, cp, &ioMutex, b]{runScenario(&ws, cp,&ioMutex,b);});
for (auto& t : v)
    t.join();
...
```

In function `runScenario` a `GAMSJob` is created and run just like in the preceding example of `Transport5`. The output section is also the same except for the fact that it is locked by using a `lock_guard` on the `ioMutex`. That means the threads of `runScenario` are running in parallel but the output block of different threads cannot be executed in parallel since it is locked using the same `ioMutex`.

```
...
void Transport6::runScenario(GAMSWorkspace* ws, const GAMSCheckpoint& cp, mutex* ioMutex, double b)
{
    auto t6 = ws->addJobFromString("bmult=" + to_string(b) + "; solve transport min z use lp; ms=transport.modelstat;
      ss=transport.solvestat;", cp);
    t6.run();
    // we need to make the ouput a critical section to avoid messed up report information
    lock_guard<mutex> lck(*ioMutex);
    cout << "Scenario bmult=" << b << ":" << endl;
    cout << "  Modelstatus: " << t6.outDB().getParameter("ms").findRecord().value() << endl;
    cout << "  Solvestatus: " << t6.outDB().getParameter("ss").findRecord().value() << endl;
    cout << "  Obj: " << t6.outDB().getVariable("z").findRecord().level() << endl;
}
...
```

While the output in `Transport5` is strictly ordered subject to the order of the elements of `bmultlist`, in `Transport6` the output blocks might change their order but the blocks describing one scenario are still appearing together due to the `lock_guard` mechanism.

If you want a further impression of the impact of the `lock_guard`, just rerun `Transport6` but comment out the `lock_guard` as follows and compare the output.

```
...
void Transport6::runScenario(GAMSWorkspace* ws, const GAMSCheckpoint& cp, mutex* ioMutex, double b)
{
    auto t6 = ws->addJobFromString("bmult=" + to_string(b) + "; solve transport min z use lp; ms=transport.modelstat;
      ss=transport.solvestat;", cp);
    t6.run();
    // we need to make the ouput a critical section to avoid messed up report information
    // lock_guard<mutex> lck(*ioMutex);
    cout << "Scenario bmult=" << b << ":" << endl;
    cout << "  Modelstatus: " << t6.outDB().getParameter("ms").findRecord().value() << endl;
    cout << "  Solvestatus: " << t6.outDB().getParameter("ss").findRecord().value() << endl;
    cout << "  Obj: " << t6.outDB().getVariable("z").findRecord().level() << endl;
}
...
```

7.9.3.16 How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)

In `Transport7` the usage of **gams::GAMSModelInstance** is demonstrated.

At first we create a checkpoint `cp` as in the preceding examples. Then we create the GAMSModelInstance `mi` using the `GAMSCheckpoint.addModelInstance` method. Note that the `GAMSJob` again contains no solve statement and the demand multiplier is already included with default value 1.

```
...
GAMSCheckpoint cp = ws.addCheckpoint();
GAMSJob t7 = ws.addJobFromString(getModelText());
t7.run(cp);
GAMSModelInstance mi = cp.addModelInstance();
...
```

7.9.3.17 How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)

A `GAMSModelInstance` uses a `syncDB` to maintain the data. We define `bmult` as `GAMSParameter` using the `GAMSDatabase.addParameter` method and specify `cplex` as solver. Afterwards the `GAMSModelInstance` is instantiated with three arguments: the solve statement, the `GAMSOptions` object `opt` and `bmult`. The `GAMSModifier` means that `bmult` is modifiable while all other parameters, variables and equations of GAMSModelInstance `mi` stay unchanged.

We use the `GAMSParameter.addRecord` method to assign a value to `bmult` that can be varied afterwards using the `GAMSParameter.firstRecord` method to reproduce our well-known example with different demand multipliers.

```
...
```

```
GAMSModelInstance mi = cp.addModelInstance();
GAMSParameter bmult = mi.syncDb().addParameter("bmult", 0, "demand multiplier");
GAMSOptions opt = ws.addOptions();
opt.setAllModelTypes("cplex");
// instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare bmult mutable
mi.instantiate("transport use lp min z", opt, GAMSModifier(bmult));
bmult.addRecord().setValue(1.0);
vector<double> bmultlist = { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
for (double b : bmultlist) {
    bmult.firstRecord().setValue(b);
    mi.solve();
    cout << "Scenario bmult=" << b << ":" << endl;
    cout << "  Modelstatus: " << mi.modelStatusAsString() << endl;
    cout << "  Solvestatus: " << mi.solveStatusAsString() << endl;
    cout << "  Obj: " << mi.syncDb().getVariable("z").findRecord().level() << endl;
}
...
```

7.9.3.18 How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)

We create a `GAMSModelInstance` and define `x` as `GAMSVariable` and its upper bound as `GAMSParameter` `xup`. At the following instantiate method `GAMSModifier` has three arguments. The first one says that `x` is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the `GAMSParameter` that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```
...
mi = cp.addModelInstance();
GAMSVariable x = mi.syncDb().addVariable("x", 2, GAMSEnum::VarType::Positive, "");
GAMSParameter xup = mi.syncDb().addParameter("xup", 2, "upper bound on x");
// instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare upper bound of X mutable
mi.instantiate("transport use lp min z", GAMSModifier (x, GAMSEnum::SymbolUpdateAction::Upper, xup));
for (GAMSSetRecord i : t7.outDB().getSet("i")) {
    for (GAMSSetRecord j : t7.outDB().getSet("j")) {
        xup.clear();
        xup.addRecord(i.key(0), j.key(0)).setValue(0);
        mi.solve();
        cout << "Scenario link blocked=" << i.key(0) << "-" << j.key(0) << endl;
        cout << "  Modelstatus: " << mi.modelStatusAsString() << endl;
        cout << "  Solvestatus: " << mi.solveStatusAsString() << endl;
        cout << "  Obj: " << mi.syncDb().getVariable("z").findRecord().level() << endl;
    }
}
...
```

7.9.3.19 How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)

We initialize a GAMSCheckpoint `cp` from a `GAMSJob`. Then we define a vector that represents the different values of the demand multiplier. While `ioMutex` is used to avoid messed up output, `vectorMutex` synchronizes the access to the `vector`. Then we start multiple thread executing the method `scenSolve`. The number of parallel executed threads is specified by `nrThreads`. All trheads share the same `vector` that provides the different values for `bmult`.

```
...
GAMSCheckpoint cp = ws.addCheckpoint();
ws.addJobFromString(getModelText()).run(cp);
vector<double> bmultVector = { 1.3, 1.2, 1.1, 1.0, 0.9, 0.8, 0.7, 0.6 };
int nrThreads = 2;
std::mutex vectorMutex;
std::mutex ioMutex;
vector<thread> v;
for (int i = 0; i < nrThreads; i++)
    v.emplace_back([&ws, &cp, &bmultVector, &vectorMutex, &ioMutex] {scenSolve(&ws, &cp, &bmultVector, &vectorMutex,
      &ioMutex);});
for (auto& t : v)
    t.join();
...
```

In scenSolve we create and instantiate a GAMSModelInstance as in the preceding examples and make bmult modifiable. As long as bmultVector is not empty, an elements is taken and removed from the vector. The value is used for the scalar bmult. Note that we chose cplex as solver because it is thread safe (gurobi would also be possible). Once the vector is empty the loop terminates.

```
void Transport8::scenSolve(GAMSWorkspace* ws, GAMSCheckpoint* cp, vector<double>* bmultVector, std::mutex* vectorMutex,
      std::mutex* ioMutex)
{
    unique_lock<mutex> vectorLock(*vectorMutex);
    GAMSModelInstance mi = cp->addModelInstance();
    vectorLock.unlock();
    GAMSParameter bmult = mi.syncDb().addParameter("bmult", "demand multiplier");
    GAMSOptions opt = ws->addOptions();
    opt.setAllModelTypes("cplex");
    // instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare bmult mutable
    mi.instantiate("transport use lp min z", opt, GAMSModifier(bmult));
    bmult.addRecord().setValue(1.0);
    while (true)
    {
        double b;
        // dynamically get a bmult value from the vector instead of passing it to the different threads at creation time
        vectorLock.lock();
        if (bmultVector->empty())
            return;
        b = bmultVector->back();
        bmultVector->pop_back();
        vectorLock.unlock();
        bmult.firstRecord().setValue(b);
        mi.solve();
        // we need to make the output a critical section to avoid messed up report informations
        unique_lock<mutex> ioLock(*ioMutex);
        cout << "Scenario bmult=" << b << ":" << endl;
        cout << "  Modelstatus: " << mi.modelStatus() << endl;
        cout << "  Solvestatus: " << mi.solveStatus() << endl;
        cout << "  Obj: " << mi.syncDb().getVariable("z").findRecord().level() << endl;
        ioLock.unlock();
    }
}
```

7.9.3.20 How to fill a GAMSDatabase by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a GAMSDatabase. It can only be run on Windows since it makes use of the Microsoft Access Driver. Furthermore the example makes use of the Qt SQL module. Note that you need to have Microsoft Access installed and that its bitness needs to match the bitness of your GAMS version. In the example we call a function readFromAccess that finally returns a GAMSDatabase as shown below.

```
...
GAMSDatabase db = readFromAccess(ws);
...
```

The data we are going to read can be found in <GAMS system directory>\apifiles\Data\transport.accdb. It might be helpful to open this file for a better understanding. The function begins with the creation of an empty GAMSDatabase. Afterwards we create a QSqlDatabase, specify the connection string and open the connection to the database. If opening the connection was successful, data is read from the database into the GAMSDatabase using the methods readSet and readParameter.

```
GAMSDatabase readFromAccess(GAMSWorkspace ws)
{
    GAMSDatabase db = ws.addDatabase();
    QSqlDatabase sqlDb = QSqlDatabase::addDatabase("QODBC", "readConnection");
    QString strAccessConn = ("Driver={Microsoft Access Driver (*.mdb, *.accdb)};DSN='';DBQ=" + ws.systemDirectory() \
                            + cPathSep + "apifiles" + cPathSep + "Data" + cPathSep + "transport.accdb").c_str();
    sqlDb.setDatabaseName(strAccessConn);
    if(sqlDb.open())
    {
        // read GAMS sets
        readSet(sqlDb, db, "SELECT Plant FROM Plant", "i", 1, "canning plants");
        readSet(sqlDb, db, "SELECT Market FROM Market", "j", 1, "markets");
        // read GAMS parameters
        readParameter(sqlDb, db, "SELECT Plant,Capacity FROM Plant", "a", 1, "capacity of plant i in cases");
        readParameter(sqlDb, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases");
        readParameter(sqlDb, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in thousands of miles");
        sqlDb.close();
    }
    else
    {
        cout << "Error: Failed to create a database connection. " << sqlDb.lastError().text().toStdString() << endl;
        exit(1);
```

```
    }
    return db;
}
```

The function readSet creates a QSqlQuery that reads data from a table of the Access database. A GAMSSet is created and populated with the data.

```
void readSet(QSqlDatabase sqlDb, GAMSDatabase db, string strAccessSelect, string setName, int setDim, string setExp = "")
{
    QSqlQuery query(sqlDb);
    if (!query.exec(strAccessSelect.c_str()))
    {
        cout << "Error executing query on set '" << setName << "'" << endl;
        cout << query.lastError().text().toStdString() << endl;
        exit(1);
    }
    if (query.size() && (query.record().count() != setDim))
    {
        cout << "Number of fields in select statement does not match setDim" << endl;
        exit(1);
    }
    GAMSSet i = db.addSet(setName, setDim, setExp);
    vector<string> keys = vector<string>(setDim);
    while (query.next())
    {
        for (int idx = 0; idx < setDim; idx++)
            keys[idx] = query.value(idx).toString().toStdString();
        i.addRecord(keys);
    }
}
```

Once we read in all the data we can create a GAMSJob from the GAMSDatabase and run it as usual. Finally the results are written to transport.accdb.

7.9.3.21 How to fill a GAMSDatabase by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from <GAMS system directory>\apifiles\Data\transport.xlsx. It can only be run on Windows. Note that you need to have Microsoft Excel installed and that its bitness needs to match the bitness of your GAMS version. The model is given as string without data like in in many examples before. At first we have to add

```
#include <QAxObject>
#include <Windows.h>
```

to be able to use the QAxObject class, which serves as a wrapper for COM objects. Furthermore we include Windows.h to access the functions CoInitialize and CoUninitialize. In the main function we open the aforementioned file to get access to its content. Note that we need to call CoInitialize(0) first in order to initialize ActiveX. We create an instance of the class QAxObject and use the querySubObject method multiple times to get to the sheets of the workbook.

```
...
QAxObject* excel = new QAxObject( "Excel.Application", 0 );
QAxObject* workbooks = excel->querySubObject( "Workbooks" );
QAxObject* workbook = workbooks->querySubObject( "Open(const QString&)", fileName );
QAxObject* sheets = workbook->querySubObject( "Worksheets" );
...
```

The method sheetToParameter is used in order to transfer data from the workbook into GAMSParameter instances. In order to determine from which sheet the actual data needs to be read, the method takes the QAxObject instance referring to the sheets and the actual sheet name. A new GAMSParameter is added to the GAMSDatabase object db using set1 and set2 as domains. The next step is to determine the number of columns and the number of rows used. The for loop reads the actual data, adds the records to the new GAMSParameter and sets their values.

```
...
GAMSParameter sheetToParameter(QAxObject* sheets, string sheetName, GAMSDatabase db, string paramName, string paramText,
      GAMSSet set1, GAMSSet set2)
{
    QAxObject* sheet = sheets->querySubObject( "Item( string )", sheetName.c_str() );
    vector<GAMSDomain> sets {set1, set2};
    GAMSParameter param = db.addParameter(paramName, paramText, sets);
    QAxObject* usedrange = sheet->querySubObject( "UsedRange");
    QAxObject * columns = usedrange->querySubObject("Columns");
    int intCols = columns->property("Count").toInt();
    QAxObject * rows = usedrange->querySubObject("Rows");
    int intRows = rows->property("Count").toInt();
```

```
    for (int j = 2; j <= intCols; j++) {
        string namej = sheet->querySubObject("Cells( int, int )", 1, j)->dynamicCall("Value()").toString().toStdString();
        for (int i = 2; i <= intRows; ++i) {
            string namei = sheet->querySubObject("Cells( int, int )", i,
      1)->dynamicCall("Value()").toString().toStdString();
            GAMSParameterRecord rec = param.addRecord(namei, namej);
            double value = sheet->querySubObject("Cells( int, int )", i, j)->dynamicCall("Value()").toDouble();
            rec.setValue(value);
        }
    }
    return param;
}
...
```

After all required data was read from the workbook, we need to close it and quit the `Excel` application.

```
...
workbook->dynamicCall("Close()");
excel->dynamicCall("Quit()");
...
```

Now we can create und run the `GAMSJob` using the created `GAMSDatabase` as usual.

```
...
GAMSOptions opt = ws.addOptions();
GAMSJob t10 = ws.addJobFromString(getModelText());
opt.setDefine("gdxincname", db.name());
opt.setAllModelTypes("xpress");
t10.run(opt, db);
for (GAMSVariableRecord record : t10.outDB().getVariable("x"))
    cout << "x(" << record.key(0) << "," << record.key(1) << "): level=" << record.level() <<
            " marginal=" << record.marginal() << endl;
...
```

Finally we need to call 'CoUninitialize' in order to close the `COM` library.

```
...
::CoUninitialize();
...
```

7.9.3.22 How to create and use a save/restart file (Transport11)

In `Transport11` we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a `GAMSCheckpoint`. We start by creating the save/restart file by calling `createSaveRestart`.

```
...
std::string cpName = "tbase";
createSaveRestart(argc, argv, cpName);
...
```

In function `createSaveRestart` we choose the `checkpointName` as relative path to the current directory in order to create a `GAMSWorkspace`. Then we create a `GAMSJob` from a string. Note that the string given via `getBaseModelText()` contains the basic definitions of sets without giving them a content (that is what $onempty is used for). Afterwards we specify a `GAMSOption` to only compile the job without executing it. Then we create a checkpoint `cp` that is initialized by the following run of the `GAMSJob` and stored in the file given as argument to the function. This becomes possible because the `addCheckpoint` method accepts identifiers as well as file names as argument.

```
void createSaveRestart(int argc, char* argv[], const string &checkpointName)
{
    GAMSWorkspaceInfo wsInfo;
    if (argc > 1)
        wsInfo.setSystemDirectory(argv[1]);
    wsInfo.setWorkingDirectory("." +(cPathSep+ checkpointName));
    GAMSWorkspace ws(wsInfo);
    GAMSJob j1 = ws.addJobFromString(getBaseModelText());
    GAMSOptions opt = ws.addOptions();
    opt.setAction(GAMSOptions::EAction::CompileOnly);
    auto checkpoint = ws.workingDirectory() + cPathSep + checkpointName;
    GAMSCheckpoint cp = ws.addCheckpoint(checkpoint);
    j1.run(opt, cp);
}
```

So what you should keep in mind before we return to further explanations of the main function is, that now the file tbase in folder `tbase` contains a checkpoint. Now in the main function we define some data using C++ data structures as we already did in `Transport4` before we create another `GAMSWorkspace`.

```
...
GAMSWorkspaceInfo wsInfo;
if (argc > 1)
    wsInfo.setSystemDirectory(argv[1]);
wsInfo.setWorkingDirectory("." +(cPathSep+ cpName));
GAMSWorkspace ws(wsInfo);
...
```

Afterwards we set up the `GAMSDatabase` like we already did in `Transport4`. Once this is done we run a `GAMSJob` using this data plus the checkpoint stored in file tbase.

```
...
GAMSCheckpoint cpBase = ws.addCheckpoint("tbase");
GAMSOptions opt = ws.addOptions();
GAMSJob t4 = ws.addJobFromString(getModelText(), cpBase);
opt.setDefine("gdxincname", db.name());
opt.setAllModelTypes("xpress");
t4.run(opt, db);
for (auto record : t4.outDB().getVariable("x"))
    cout << "x(" << record.key(0) << "," << record.key(1) << "): level=" << record.level() <<
            " marginal=" << record.marginal() << endl;
...
```

Note that the string from which we create job `t4` is different to the one used to prepare the checkpoint stored in tbase and is only responsible for reading in the data from the `GAMSDatabase` correctly. The entire model definition is delivered by the checkpoint `cpBase` which is equal to the one we saved in `tbase`.

7.9.4 How to use the pre configured example projects

Depending on the operating system, the GAMS C++ API comes with example project configurations for CMake and Visual Studio. They provide an easy way of building and running the distributed GAMS C++ examples and are located in <`GAMS system directory`>\`apifiles`\`C++`.

7.9.4.1 CMake

Windows:

Create build directory

```
cd <GAMS system directory>\apifiles\C++
mkdir build && cd build
```

Run CMake and build project

```
cmake -G "Visual Studio 16 2019" -DCMAKE_BUILD_TYPE=Release -DVSVERSION:STRING=vs2019 ..
msbuild.exe examples.sln /p:Configuration=Release
```

Note

For other Visual Studio versions, the generator-name (-G) needs to be adjusted (see cmake documentation).

All generated executables can now be found in the CMake build directory. The exact path depends on the project name and the config setting. Assuming the solution was build with `-config Release`, a `transport1.exe` executable is located in <`GAMS system directory`>\`apifiles`\`C++`\`build`\`transport1`\`Release`. Therefore you can use the following commands to execute the transport1 example:In order to execute an example (e.g. transport1) run the following commands:

```
cd transport1\Release
set PATH=%PATH%;<GAMS system directory>\apifiles\C++\lib\vs2019
transport1.exe
```

Linux:

Create build directory

```
cd <GAMS system directory>/apifiles/C++
mkdir build && cd build
```

Run CMake and build project

```
cmake ..
make
```

All generated executables can now be found in the CMake build directory. For every project a subdirectory is created. Assuming you want to execute the transport1 example, use the following commands:

```
cd transport1
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<GAMS system directory>/apifiles/C++/lib
./transport1
```

macOS:

Create build directory

```
cd <GAMS system directory>/apifiles/C++
mkdir build && cd build
```

Run CMake and build project.

Note

> By default, Clang is used as a compiler. If GCC was used to build the gamscpp library, then use `-D USE-GCC="ON"` to select the correct gamscpp library from your GAMS folder. Otherwise, the build will fail.

```
cmake ..
make
```

All generated executables can now be found in the CMake build directory. For every project a subdirectory is created. Assuming you want to execute the transport1 example, use the following commands:

```
cd transport1
export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:<GAMS system directory>/apifiles/C++/lib
./transport1
```

7.9.4.2 Microsoft Visual Studio

The Visual Studio solutions are available for Windows only. Open the solution file that matches the version of your Visual Studio installation (e.g. <GAMS system directory>\apifiles\C++\examples-vs2017.sln).

As soon as the solution is loaded, click on **Build** > **Configuration Manager** and adjust the `Active solution configuration` and the `Active solution platform`. The solution configuration needs to be set to `Release` for using the distributed binaries of the GAMS C++ API. Note that the platform needs to match the version of your GAMS installation.

The examples `Transport9` and `Transport10` require a Qt installation. If you do not have one, you need to disable these projects. Right click on a project in the Solution Explorer and choose **Unload Project** from the context menu.

Click on **Build** > **Build Solution** in order to compile and link the examples.

Note

In some cases the build might return with `xcopy` errors. Just build the project again. Make sure that you don't do a rebuild.

In order to run an example, click on **Debug** > **Start Without Debugging**. This will execute the current StartUp project. Changing the StartUp project can be achieved by right clicking on a project in the Solution Explorer and choosing **Set as StartUp Project**.

7.10 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS Java API. It allows the user to start immediately working with the API by providing a set of small examples based on the well-known **transportation problem**. These examples introduce several API features step by step.

- Getting started A quick introduction about how to compile and run a program from command line or an IDE
- Important Classes of the API Overview of some fundamental classes of the GAMS Java API
- How to use API An extensive set of examples describing how to use API components

7.10.1 Getting started

Since GAMS 44.1.0, a Java program that uses GAMS Java API requires at least Java SE 11 to compile and run. For a Java program that still requires Java SE 8 to compile and run, it is possible to use a separate version of GAMS Java API that targets Java SE 8, but the new or updated functionalities that are released after GAMS 43 are not avaiable in this Java API version.

For all platforms, assume GAMS system has been installed at `[PathToGAMS]` directory, called GAMS directory. The directory `[PathToGAMS]` denotes the path setting according to your GAMS installation on targeted platforms. For instance

- on Windows-based platforms with GAMS distribution 46.4 installed, `[PathToGAMS]` denotes `C:\GAMS\win64\46.4`

- on Unix-based platforms with GAMS distribution 46.4 installed, `[PathToGAMS]` denotes `/usr/gams/gams46.4_linux_x64_64_sfx`

All GAMS Java API classes are contained within one single jar file `GAMSJavaAPI.jar` (or `GAMSJavaAPI-8.jar` for Java SE 8) with a namespace `com.gams.api`. The jar files are distributed with the current GAMS distribution and located at

- on Windows-based platforms:

  `[PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar`

- on Unix-based platforms:

  `[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar`

Note

Since GAMS 44.1.0, GAMS Java API has an additional dependency on JSON.simple, a simple Java library for JSON processing, read and write JSON data. Compiling and running a program that uses `GAMSJavaAPI.jar` requires `json-simple-1.1.1.jar` to be in the same directory as `GAMSJavaAPI.jar` (This is not the case for `GAMSJavaAPI-8.jar`). Missing `json-simple-1.1.1.jar` from the same directory might raise runtime exception when using related functionality in the GAMS Java API.

There are two different approaches of how to use the GAMS Java API. Either the Java source files can be edited with any common editor and compiled from command line or a Java IDE can be used. The following sections give a quick introduction about the different possibilities.

7.10.1.1 Compiling a Program

To compile a Java program, one usually provides the following information to Java compiler:

- the directory(ies) containing all required Java class files
- the directory to place the generated class files
- the name of the Java source file

A Java program that uses GAMS Java API requires class files that are contained in `GAMSJavaAPI.jar` for compilation (or in `GAMSJavaAPI-8.jar` for Java SE 8).

Compiling a Program from Command Line

To compile a Java program that uses GAMS Java API, for instance `HelloAPI.java`, at the command line:

```
javac -cp [CLASSPATH] -d [TARGETDIR] HelloAPI.java
```

where

```
[CLASSPATH]  is 'GAMSJavaAPI.jar' (or 'GAMSJavaAPI-8.jar' for Java SE 8) with its location,
             as it contains class files that are used by 'HelloAPI.java'.
             On Windows-based platform, [CLASSPATH] denotes
                 [PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar
             On Unix-based platforms, [CLASSPATH] denotes
                 [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar

[TARGETDIR]  is the directory to place the generated classe file.
```

Note

- it is possible to add [CLASSPATH] to your CLASSPATH environment variable of the operating system as an alternative to specifying -cp [CLASSPATH].
- -d [TARGETDIR] is optional. In case -d [TARGETDIR] is omited and the compilation is successful, the class file will be generated and located under the current directory.
- GAMSJavaAPI.jar has an additional dependency on json-simple-1.1.1.jar (not the case for GAMSJavaAPI-8.jar). Though it is not necessary to add json-simple-1.1.1.jar to [CLASSPATH] as it already been added into the GAMSJavaAPI.jar's Classpath. It is important, however, for json-simple-1.1.1.jar to be in the same directory as GAMSJavaAPI.jar.

To compile other programs, change the arguments accordingly. See also How to compile and run examples from the GAMS

Compiling a Program from Java IDE

In case of compiling a program under a Java IDE (for instance, Eclipse, NetBeans, or IntelliJ), the location of the jar file GAMSJavaAPI.jar (or GAMSJavaAPI-8.jar for Java SE 8) shall be added into the Java build path of the IDE's project properties.

For an Eclipse user, see How to import an Eclipse Java Project from the GAMS system directory.

7.10.1.2 Running a Program

To run a Java program, one usually provide the following information to Java Virtual Machine:

- the directory(ies) containing all required Java classes
- the directory(ies) containing all required shared libraries
- the name of an entry point class (in most case containing main method)

A Java program that uses GAMS Java API requires a number of shared libraries for establishing a connection with GAMS software components during run time. These shared libraries are platform dependent and they are located at [PathToGAMS]/apifiles/Java/api directory.

Running a Program from Command Line

To run a Java program that uses GAMS Java API, for instance HelloAPI.class containing in [TARGETDIR] directory, at the command line type:

```
java -cp [CLASSPATH]  HelloAPI
```

where

```
[CLASSPATH]  is the list of two paths, a path of 'GAMSJavaAPI.jar' (or 'GAMSJavaAPI-8.jar' for Java S
             and a path of targeted directory containing 'HelloAPI.class', separated by path separato
             on the targeted platform.
             On Windows-based platform, [CLASSESPATHS] denotes
                 [PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar;[TARGETDIR]
             On Unix-based platform, [CLASSESPATHS] denotes
                 [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar:[TARGETDIR]

HelloAPI     is the name of an entry point class containing main method
```

Please note that it is possible to add [CLASSPATH] to your CLASSPATH environment variable of the operating system as an alternative to specify -cp [CLASSPATH].

Attention

> Most operations performed by GAMS Java API such as gdx or options operations require shared libraries (located in the folder `[PathToGAMS]/apifiles/Java/api` by default) when running a program. It is recommended to specify the path to shared libraries when running a program. In case no `java.library.path` has been specified when running a program, the Java API will determine the path of the shared libraries from the java class path containing `GAMSJavaAPI.jar`.

It is possible to specify the path to shared libraries by passing the following argument to Java Virtual Machine:

```
-Djava.library.path=[LIBRARYPATH]
```

where `[LIBRARYPATH]` describes the list of paths containing all required shared libraries (as previously mentioned, it is `[PathToGAMS]/apifiles/Java/api` by default). In such case, the Java API will give priority to the java library path when loading all required JNI libraries during run time. The java API will raise an exception when either the java virtual machine fails to load the required libraries from the specified java library path or there is a version conflict of shared libraries during run time.

To run other programs, change the arguments accordingly. See also How to compile and run examples from the GAMS syst

Running a Program from Java IDE

In case of running a program under a Java IDE (for instance, Eclipse, NetBeans, or IntelliJ), it is possible to set the nessary arguments via the properties of the IDE project.

For an Eclipse user, see How to import an Eclipse Java Project from the GAMS system directory.

7.10.1.3 Setting Up Your Environment

Since 24.3, the GAMS system directory can be specified within the program during run time. Therefore the setting up of execution environment before running a program is no longer required (see also Release Notes 24.3).

Nevertheless, it is still possible to preconfigure the GAMS system directory before running a program and use the default workspace setting within a program. In the default setting, GAMS system directory is not specified by a user within the program. The directory will be searched during run time from environment variable in the following order (depends on the target platform):

- On Windows-based platform: first from `PATH` environment variable. If not found, from the platform windows registry `gams.location`,
- On macOS-based platform: first from `PATH` environment variable. If not found, from `DYLD_LIBRARY_PATH`,
- On other Unix-based platfrom: from `PATH` environment variable. If not found, from `LD_LIBRARY_PATH`.

The following examples illustrate how to add `[PathToGAMS]` into the environment variable `PATH` on different platforms:

- On Windows 2000, XP, Vista, or Windows 7 desktop:

```
> right-click on 'My Computer'
> choose 'Properties' (alternatively, click on 'System' icon in the control panel)
> click on 'Advanced' tab (or 'Advance system settings')
> click on 'Environment Variables'
> edit 'PATH' by adding [PathToGAMS] to the variable using a semi-colon as a separator.
```

- On Window-based platform via command line terminal:

```
set PATH=[PathToGAMS];%PATH%
```

- On Unix-based platform via command line terminal using Bourne shell and its derivatives:

```
export PATH=[PathToGAMS]:${PATH}
```

- On Unix-based platform via command line terminal using C Shell:

```
setenv PATH [PathToGAMS]:${PATH}
```

7.10.1.4 How to compile and run examples from the GAMS system directory

GAMS provides several examples to demonstrate how to use GAMS Java API. These examples are located in the `[PathToGAMS]/apifiles/Java` directory containing various examples based on the different optimization problems. The following explanations guide you through the compilation and execution process of one example based on the well-known **transportation problem**, assuming that Java Runtime System is already installed on your machine.

For instance, `Transport1.java` under the directory `[PathToGAMS]/apifiles/Java/transport` demonstrates how to retrieve GAMS transport model from GAMS Model Library, execute the model with various GAMS options, and extract results after execution.

To compile `Transport1.java` at the command line under the directory `[PathToGAMS]/apifiles/Java/transport`:

```
javac -cp [CLASSPATH] -d [TARGETDIR] Transport1.java
```

where

```
[CLASSPATH]  is a location of 'GAMSJavaAPI.jar' (or 'GAMSJavaAPI-8.jar' for Java SE 8),
             as it contains classes files that are used by 'Transport1.java'.
             On Windows-based platform, [CLASSPATH] denotes
                [PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar;
             On Unix-based platforms, [CLASSESPATHS] denotes
                [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar

[TARGETDIR]  is the destination directory to place the generated classed file.
```

As `Transport1.java` is declared under package `com.gams.examples.transport`, the output class file will be located in `[TARGETDIR]` under the directory structure corresponding to package information. This means the compiled output file `Transport1.class` is located under:

- `[TARGETDIR]\com\gams\examples\transport` on Windows-based platform, or
- `[TARGETDIR]/com/gams/examples/transport` on Unix-based platforms.

To run `Transport1.class` from the command line:

```
java -cp [CLASSPATH] com.gams.examples.transport.Transport1
```

where

```
[CLASSPATH]  is the list of two paths, 'GAMSJavaAPI.jar' (or 'GAMSJavaAPI-8.jar'
             for Java SE 8) with its path and a directory containing 'Transport1.class'
             and (in this case [TARGETDIR] from the compliation step), separated by
             path separator on the targeted platform.
             On Windows-based platform, [CLASSESPATHS] denotes
                [PathToGAMS]\apifiles\Java\api\GAMSJavaAPI.jar;[TARGEDIR]
             On Unix-based platforms, [CLASSESPATHS] denotes
                [PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar:[TARGEDIR]
```

Please note that the path `[TARGETDIR]` can be either absolute or relative path representing the directory that contains `Transport1.class`.

To compile and run other examples under the directory `[PathToGAMS]/apifiles/Java`, adjust the arguments accordingly.

See How to use API for detailed explanations on the series of **transportation problem** examples located in the folder `[PathToGAMS]/apifiles/Java/transport`.

7.10.1.5 How to import an Eclipse Java Project from the GAMS system directory

Since GAMS version 24.2 there is an prepared Java project that can be imported into Eclipse. The project is located in folder `[PathToGAMS]/apifiles/Java/Eclipse` and contains the examples based on the **transportation problem**. The following explanations guide you through the import and preparation process, assuming that the Eclipse Java IDE is already installed on your machine.

To import the project located in folder `[PathToGAMS]/apifiles/Java/Eclipse` into the workspace:

1. open the Eclipse IDE with the chosen workspace location, click on **File** menu and choose **Import**.
2. In the **Import** window, choose **General** > **Existing Projects into Workspace** then click **Next**.
3. In the next page of **Import** window, click **Browse..** and select the folder with the prepared project (`[PathToGAMS]/apifiles/Java/Eclipse`) as root directory.

   We recommend to check for option **Copy projects into workspace**. Click **Finish** to finish the **Import** window.
4. Now the project is imported, and appears in the **Package Explorer** window on the left hand side of of the IDE (shown below with its elements expanded).
5. As every example in the project requires GAMS Java API classes files to compile (all classes are packaged in `GAMSJavaAPI.jar` located in the folder `[PathToGAMS]/apifiles/Java/api`), you need to tell eclipse where to find `GAMSJavaAPI.jar`. To do this, either click on the project then choose **Project** > **Properties** or right click on the project name and choose **Properties** .
6. The **Properties** window of the proejct appears. Click **Java Build Path** on the left of the window, choose **Libraries** tab, choose **Add External JARs...**.

   select `GAMSJavaAPI.jar` located in the folder `[PathToGAMS]/apifiles/Java/api` to be opened.

   and click **OK** to finish addding `[PathToGAMS]/apifiles/Java/api/GAMSJavaAPI.jar` to **Java Build Path**.

7. Now all the java files should be successfully compiled without errors except for `Transport10` (unsuccessful compilation is denoted with red x mark in front of the file name). This is because `Transport10` requires an additional `jexcelapi` JAR file `jxl.jar` from jexcelapi. It can be downloaded and unzipped the downloaded archive into a local directory e.g. `C:\tools`. Then the jar file `C:\tools\jexcelapi\jxl.jar` can be added to the the Java Build Path of the project in a similar way as explained in 6.

8. To run a transport example, for example `Transport1`, you need to create a `Run Configuration`.

   To do this, either click **Run** menu and choose **Run Configurations...**

   or open the drop down menu next to the run button and click on **Run Configurations...**.

   Then create a new run configuration by either press the **New** button in the left hand side of the window or right click at **Java Application** in the left hand side of the window and choose **New**.

9. The run configuration for `Transport1` appears.

   For each run configuration it is possible to configure the execution of a program, such as the arguments of the virtual machine, class paths, environment variables and so on.

   For instance, to specify the `java.library.path` for `Transport1` run configuration, choose **Arguments** tab in the run configuration and specify `-Djava.library.path=[PathToGAMS]/apifiles/Java/api` as one of the **VM arguments**. Then Click **Apply** and **Run**.

Note that the run configuration described in 8. and 9. is needed to be set for every single example. Once this is done you should be able to use the run configuration to repeat the execution of all the transport examples.

For a more detailed explanations on each transport examples see How to use API.

7.10.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS Namespace. Their usage is demonstrated by an extensive set of examples. All GAMS Java API classes are contained within one single jar file `GAMSJavaAPI.jar` (or `GAMSJavaAPI-8.jar` for Java SE 8) with a namespace `com.gams.api`. It provides objects to interact with the General Algebraic Modeling System (`GAMS`). Objects in this namespace allow convenient exchange of input data and model results (**GAMSDatabase**) and help to create and run GAMS models (**GAMSJob**), that can be customized by GAMS options (**GAMSOptions**). Furthermore, it introduces a way to solve a sequence of closely related models in the most efficient way (**GAMSModelInstance**).

Other classes are **GAMSWorkspace**, **GAMSOptions**, **GAMSSymbol**, and **GAMSException**.

7.10.3 How to use API

In the GAMS system directory there are some examples provided that illustrate the usage of the Java API. `[PathToGAMS]/apifiles/Java/transport` contains multiple examples dealing with the well-known **transportation problem**. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- How to choose the GAMS system (Transport1)
- How to export data to GDX (TransportGDX)

- How to import data from GDX (TransportGDX)
- How to run a GAMSJob from file (Transport1)
- How to retrieve a solution from an output database (Transport1)
- How to specify the solver using GAMSOptions (Transport1)
- How to run a job with a solver option file and capture its log output (Transport1)
- How to use include files (Transport2)
- How to set a non-default working directory (Transport3)
- How to read data from string and export to GDX (Transport3)
- How to run a job using data from GDX (Transport3)
- How to run a job using implicit database communication (Transport3)
- How to define data using Java data structures (Transport4)
- How to prepare a GAMSDatabase from Java data structures (Transport4)
- How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)
- How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)
- How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)
- How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)
- How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)
- How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)
- How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)
- How to fill a GAMSDatabase by reading from MS Access (Transport9)
- How to fill a GAMSDatabase by reading from MS Excel (Transport10)
- How to create and use a save/restart file (Transport11)

7.10.3.1 How to choose the GAMS system (Transport1)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing `Transport1` with C:/GAMS/win64/46.4 we use the 64-bit version of GAMS 46.4 to run `Transport1` even if our default GAMS system might be a different one. This is managed by the following code:

```
...
  GAMSWorkspaceInfo  wsInfo  = new GAMSWorkspaceInfo();
  if (args.length > 0)
     wsInfo.setSystemDirectory( args[0] );
  GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
...
```

Remember that the bitness of the GAMS system has to match the bitness of your Java Runtime Environment.

7.10.3.2 How to export data to GDX (TransportGDX)

Although the Object-oriented Java API offers much more than exchanging data between Java and GDX, a common use case is the export and import of GDX files. The central class for this purpose is GAMSDatabase. We assume that the data to be exported is available in Java data structures.

```
...
  List<String> plants = Arrays.asList("Seattle", "San-Diego");
  List<String> markets = Arrays.asList("New-York", "Chicago", "Topeka");
  Map<String, Double> capacity = new HashMap<String, Double>();
  {
    capacity.put("Seattle", Double.valueOf(350.0));
    capacity.put("San-Diego", Double.valueOf(600.0));
  }
  Map<String, Double> demand = new HashMap<String, Double>();
  {
    demand.put("New-York", Double.valueOf(325.0));
    demand.put("Chicago", Double.valueOf(300.0));
    demand.put("Topeka", Double.valueOf(275.0));
  }
  Map<Vector<String>, Double> distance = new HashMap<Vector<String>, Double>();
  {
    distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "New-York"}) ), Double.valueOf(2.5));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "Chicago"}) ),  Double.valueOf(1.7));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "Topeka"}) ),  Double.valueOf(1.8));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "New-York"}) ), Double.valueOf(2.5));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "Chicago"}) ),  Double.valueOf(1.8));
    distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "Topeka"}) ),   Double.valueOf(1.4));
  }
...
```

Different type of GAMS symbols are represented using different Java data structures. The data for the GAMS sets is represented using List of Strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by Map (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses Vectors for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with Java data structures.

A new GAMSDatabase instance can be created using GAMSWorkspace.addDatabase.

```
...
  // create new GAMSDatabase instance
  GAMSDatabase db = ws.addDatabase();
...
```

We start adding GAMS sets using the method `GAMSDatabase.addSet` which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created GAMSSet instance `i` using `GAMSSet.addRecord`.

```
...
  GAMSSet i = db.addSet("i", 1, "canning plants");
  for(String p : plants)
      i.addRecord(p);
...
```

GAMSParameter instances can be added by using the method `GAMSDatabase.addParameter`. In this example we use the overloaded method which takes a list of GAMSSet instances instead of the dimension for creating a parameter with domain information.

```
...
  GAMSParameter a = db.addParameter("a", "capacity of plant i in cases", i);
  for (String p : plants)
      a.addRecord(p).setValue( capacity.get(p) );
...
```

As soon as all data is prepared in the GAMSDatabase, the method `GAMSDatabase.export` can be used to create a GDX file.

```
...
  db.export("data.gdx");
...
```

7.10.3.3 How to import data from GDX (TransportGDX)

Data can be imported from a GDX file using `GAMSWorkspace.addDatabaseFromGDX`. The method takes a path to a GDX file and creates a GAMSDatabase instance.

```
...
  GAMSDatabase gdxdb = ws.addDatabaseFromGDX("data.gdx");
...
```

Reading the data from the GAMSSet `i` into a List of Strings can be done as follows:

```
...
  List<String> gdxPlants = new ArrayList<String>();
  for(GAMSSetRecord rec : gdxdb.getSet("i"))
      gdxPlants.add(rec.getKey(0));
...
```

A new List `gdxPlants` is created. `i` is retrieved by calling `GAMSDatabase.getSet` on `gdxdb`. The returned GAMSSet object can be iterated using a for-loop to access the records of the set. Each record is of type GAMSSetRecord and can be asked for its keys.

You can do the same for GAMSParameter. Instead of creating a List, we want to have the data in the form of a Map. GAMSParameterRecord can not only be asked for its keys, but also for its value. The following code snippet shows how to read the one dimensional parameter `a` into a `Map<String, Double>`.

```
...
  Map<String, Double> gdxCapacity = new HashMap<String, Double>();
  for(GAMSParameterRecord rec : gdxdb.getParameter("a"))
     gdxCapacity.put(rec.getKey(0), rec.getValue());
...
```

For a key of multi dimensional symbol, we choose Vector of String instead of String.

```
...
  Map<Vector<String>, Double> gdxDistance = new HashMap<Vector<String>, Double>();
  for(GAMSParameterRecord rec : gdxdb.getParameter("d"))
      gdxDistance.put( new Vector<String>( Arrays.asList(new String[]{rec.getKey(0), rec.getKey(1)}) ), rec.getValue());
...
```

Scalar can be read into a variable of type double by accessing the value of the first and only record.

```
...
  double gdxFreight = gdxdb.getParameter("f").getFirstRecord().getValue();
...
```

7.10.3.4 How to run a GAMSJob from file (Transport1)

At first we create our workspace using **GAMSWorkspace** `ws = new GAMSWorkspace();`. Afterward we can create a **GAMSJob** `t1` using the **addJobFromGamsLib** method and run it.

Apparently you can create a **GAMSJob** with any other gms file you might have created on your own as long as it is located in the current working directory. Then the **GAMSJob** `t1` can be defined using the **GAMSJob.addJobFromFile** method.

```
...
  // create GAMSWorkspace "ws" with default working directory
  // (the directory named with current date and time under System.getProperty("java.io.tmpdir"))
  GAMSWorkspace ws = new GAMSWorkspace();
  // create GAMSJob "t1" from "trnsport" model in GAMS Model Libraries
  GAMSJob t1 = ws.addJobFromGamsLib("trnsport");
  // run GAMSJob "t1"
  t1.run();
...
```

See also **Transport1.java**.

7.10.3.5 How to retrieve a solution from an output database (Transport1)

The following lines create the solution output and illustrate the usage of the **GAMSJob.OutDB** property to get access to the **GAMSDatabase** created by the **run** method. To retrieve the content of variable `x` we use the **getVariable** method and the **GAMSVariableRecord** class.

```
...
  // retrieve GAMSVariable "x" from GAMSJob's output databases
  System.out.println("Ran with Default:");
  GAMSVariable x = t1.OutDB().getVariable("x");
  for (GAMSVariableRecord rec : x)
  {
      System.out.print("x(" + rec.getKeys()[0] + ", " + rec.getKeys()[1] + "):");
      System.out.print(", level = " + rec.getLevel());
      System.out.println(", marginal = " + rec.getMarginal());
  }
...
```

See also **Transport1.java**.

7.10.3.6 How to specify the solver using GAMSOptions (Transport1)

The solver can be specified via the **GAMSOptions** class and the **GAMSWorkspace.addOptions** method. The `GAMSOptions.setAllModelTypes` property sets `xpress` as default solver for all model types which the solver can handle. Then we run our **GAMSJob** `t1` with the new **GAMSOptions**.

```
...
  // create GAMSOptions "opt1"
  GAMSOptions opt1 = ws.addOptions();
  // set all model types of "opt1" for "xpress"
  opt1.setAllModelTypes("xpress");
  // run GAMSJob "t1" with GAMSOptions "opt1"
  t1.run(opt1);
...
```

See also **Transport1.java**.

7.10.3.7 How to run a job with a solver option file and capture its log output (Transport1)

At first we create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. Afterward we use a **GAMSOptions** just like in the preceding example and `GAMSOptions.setOptFile` property to 1 to tell the solver to look for a solver option file. We specify the argument `output` in order to stream the log of the GAMSJob into the file `transport1_xpress.log`. When the `output` argument is omitted then the log will be written to standard output.

```
...
  // write file "xpress.opt" under GAMSWorkspace's working directory
  try {
      BufferedWriter optFile = new BufferedWriter(new FileWriter(
                                    ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "xpress.opt"
                               ));
      optFile.write("algorithm=barrier");
      optFile.close();
  } catch(IOException e) {
      e.printStackTrace();
      System.exit(-1);
  }
  // create GAMSOptions "opt2"
  GAMSOptions opt2 = ws.addOptions();
  // set all model types of "opt2" for "xpress"
  opt2.setAllModelTypes( "xpress" );
  // for "opt2", use "xpress.opt" as solver's option file
  opt2.setOptFile(1);
  try {
      // run GAMSJob "t2" with GAMSOptions "opt2" and capture log into "transport1_xpress.log".
      PrintStream output = new PrintStream(new File(ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR
      +"transport1_xpress.log"));
      t1.run(opt2, output);
  } catch (FileNotFoundException e) {
      // run GAMSJob "t2" with GAMSOptions "opt2" and log is written to standard output
      t1.run(opt2);
  }
...
```

See also **Transport1.java**.

7.10.3.8 How to use include files (Transport2)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings data and model are using GAMS syntax.

At first we write an include file `tdata.gms` that contains the data but not the model text:

```
...
  try {
      BufferedWriter file = new BufferedWriter(new FileWriter(
                                ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "tdata.gms"
                            ));
      file.write(data);
      file.close();
  } catch(IOException e) {
      e.printStackTrace();
      System.exit(-1);
  }
...
```

Afterwards we create a **GAMSJob** using the **GAMSWorkspace.addJobFromString** method. `GAMSOptions.defines` is used like the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line where incname is used as name for the include file in the `model` string.

```
...
  // create GAMSJob "t2" from the "model"
  GAMSJob t2 = ws.addJobFromString(model);
  // create GAMSOption "opt" and define "incname" as "tdata"
  GAMSOptions opt = ws.addOptions();
  opt.defines("incname", "tdata");
  // run GAMSJob "t2" with GAMSOptions "opt"
  t2.run(opt);
...
```

The string model contains the following lines to read in the data.

```
...
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

See also **Transport2.java**.

7.10.3.9 How to set a non-default working directory (Transport3)

At first we create a new directory. Once this is done we can use this directory when creating the **GAMSWorkspace** and make it the working directory.

```
...
  // create a directory
  File workingDirectory = new File(System.getProperty("user.dir"), "Transport3");
  workingDirectory.mkdir();
  // create a workspace
  GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
  wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());
  GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
...
```

See also **Transport3.java**.

7.10.3.10 How to read data from string and export to GDX (Transport3)

We read the data from the string data. Note that this contains no model but only data definition in GAMS syntax. By running the corresponding **GAMSJob** a **GAMSDatabase** is created that is available via the **GAMSJob.OutDB** property. We can use the **GAMSDatabase.export** method to write the content of this database to a gdx file `tdata.gdx`.

```
...
  // Create and run a job from a data file, then explicitly export to a GDX file
  GAMSJob t3 = ws.addJobFromString(data);
  t3.run();
  t3.OutDB().export( ws.workingDirectory() + GAMSGlobals.FILE_SEPARATOR + "tdata.gdx" );
...
```

See also **Transport3.java**.

7.10.3.11 How to run a job using data from GDX (Transport3)

This works quite similar to the usage of an include file explained in `Transport2` - How to use include files (Transport2)

```
.
...
  // run a job using an instance of GAMSOptions that defines the data include file
  t3 = ws.addJobFromString(model);
  GAMSOptions opt = ws.addOptions();
  opt.defines("gdxincname", "tdata");
  t3.run(opt);
...
```

Note that there are some minor changes in model due to the usage of a gdx instead of an include file.

```
...
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
...
```

See also **Transport3.java**.

7.10.3.12 How to run a job using implicit database communication (Transport3)

This example does basically the same as the two preceding examples together. We create two **GAMSJob**s `t3a` and `t3b` where the first one contains only the data and the second one contains only the model without data. After running `t3a` the corresponding **OutDB** can be read in directly just like a gdx file. Note that the database needs to be passed to the **GAMSJob.run** method as additional argument.

```
...
  GAMSVariable x = t3.OutDB().getVariable("x");
  for (GAMSVariableRecord rec : x)
      System.out.println("x(" + rec.getKeys()[0] + ", " +rec.getKeys()[1] + "): level=" + rec.getLevel() + " marginal=" +
      rec.getMarginal());
  System.out.println();
  // similar to the previous run but without exporting database into a file
  GAMSJob t3a = ws.addJobFromString(data);
  GAMSJob t3b = ws.addJobFromString(model);
  opt = ws.addOptions();
  t3a.run();
  opt.defines("gdxincname", t3a.OutDB().getName());
  t3b.run(opt, t3a.OutDB());
...
```

See also **Transport3.java**.

7.10.3.13 How to define data using Java data structures (Transport4)

We use the `List<E>` class and the `Map<Key,Value>`to define Java data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```
...
  // prepare input data
  List<String> plants = Arrays.asList("Seattle", "San-Diego");
  List<String> markets = Arrays.asList("New-York", "Chicago","Topeka");
  Map<String, Double> capacity = new HashMap<String, Double>();
  {
     capacity.put("Seattle", Double.valueOf(350.0));
     capacity.put("San-Diego", Double.valueOf(600.0));
  }
  Map<String, Double> demand = new HashMap<String, Double>();
  {
     demand.put("New-York", Double.valueOf(325.0));
     demand.put("Chicago", Double.valueOf(300.0));
     demand.put("Topeka", Double.valueOf(275.0));
  }
  Map<Vector<String>, Double> distance = new HashMap<Vector<String>, Double>();
  {
     distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "New-York"}) ), Double.valueOf(2.5));
     distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "Chicago"}) ), Double.valueOf(1.7));
     distance.put( new Vector<String>( Arrays.asList(new String[]{"Seattle", "Topeka"}) ), Double.valueOf(1.8));
     distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "New-York"}) ), Double.valueOf(2.5));
     distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "Chicago"}) ), Double.valueOf(1.8));
     distance.put( new Vector<String>( Arrays.asList(new String[]{"San-Diego", "Topeka"}) ), Double.valueOf(1.4));
  }
...
```

See also **Transport4.java**.

7.10.3.14 How to prepare a GAMSDatabase from Java data structures (Transport4)

At first we create an empty **GAMSDatabase** `db` using the **GAMSWorkspace.addDatabase** method. Afterwards we prepare the database. To add a set to the database we use the **GAMSSet** class and the **GAMSDatabase.addSet** method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our Java data structure and add them by using the **GAMSSet.addRecord** method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2 and that scalars can be treated as parameter with dimension 0.

The **GAMSJob** can be run like explained in the preceding example about implicit database communication.

```
...
  // add a database and add input data into the database
  GAMSDatabase db = ws.addDatabase();
  GAMSSet i = db.addSet("i", 1, "canning plants");
  for(String p : plants)
      i.addRecord(p);
  GAMSSet j = db.addSet("j", 1, "markets");
  for(String m : markets)
      j.addRecord(m);
  GAMSParameter a = db.addParameter("a", 1, "capacity of plant i in cases");
  for (String p : plants)
      a.addRecord(p).setValue( capacity.get(p) );
  GAMSParameter b = db.addParameter("b", 1, "demand at market j in cases");
  for(String m : markets)
      b.addRecord(m).setValue( demand.get(m) );
  GAMSParameter d = db.addParameter("d", 2, "distance in thousands of miles");
  for(Vector<String> vd : distance.keySet())
      d.addRecord(vd).setValue( distance.get(vd).doubleValue() );
  GAMSParameter f = db.addParameter("f", 0, "freight in dollars per case per thousand miles");
  f.addRecord().setValue( 90 );
  // create and run a job from the model and read gdx include file from the database
  GAMSJob t4 = ws.addJobFromString(model);
  GAMSOptions opt = ws.addOptions();
  opt.defines("gdxincname", db.getName());
  t4.run(opt, db);
...
```

See also **Transport4.java**.

7.10.3.15 How to initialize a GAMSCheckpoint by running a GAMSJob (Transport5)

The following lines of code conduct several operations. While the first line simply creates a **GAMSCheckpoint**, the second one uses the **GAMSWorkspace.addJobFromString** method to create a **GAMSJob** containing the model text and data but no solve statement. Afterwards the run method gets the **GAMSCheckpoint** as argument. That means the **GAMSCheckpoint** `cp` captures the state of the **GAMSJob**.

```
...
  GAMSCheckpoint cp = ws.addCheckpoint();
  GAMSJob t5 = ws.addJobFromString(model);
  t5.run(cp);
...
```

See also **Transport5.java**.

7.10.3.16 How to initialize a GAMSJob from a GAMSCheckpoint (Transport5)

Note that the string returned from function `model` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```
...
Scalar bmult demand multiplier /1/;
...
demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ;
...
```

```
Scalar ms 'model status', ss 'solve status';
...
```

In **Transport5** we create a list with eight different values for this demand multiplier.

```
...
  double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
...
```

For each entry of that list we create a **GAMSJob** `t5` using the **GAMSWorkspace.addJobFromString** method. Besides another string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint cp as additional argument. This results in a **GAMSJob** combined from the checkpoint plus the content provided by the string. We run the **GAMSJob** and echo some interesting data from the **OutDB** using the **GAMSDatabase.getParameter** and **GAMSDatabase.getVariable** methods, the **GAMSParameter.findRecord** and **GAMSVariable.findRecord** methods plus the **GAMSParameterRecord.getValue** property and the **GAMSVariableRecord.getLevel** method.

```
...
// create a new GAMSJob that is initialized from the GAMSCheckpoint
for(double b : bmultlist)
{
   t5 = ws.addJobFromString(
         "bmult=" + b + "; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;",
         cp
        );
   t5.run();

   System.out.println("Scenario bmult=" + b + ":");
   System.out.println(
         " Modelstatus: " + GAMSGlobals.ModelStat.lookup(
                              (int) t5.OutDB().getParameter("ms").findRecord().getValue()
                            )
    );
   System.out.println(
         " Solvestatus: " + GAMSGlobals.SolveStat.lookup(
                              (int)t5.OutDB().getParameter("ss").findRecord().getValue()
                            )
   );
   System.out.println(" Obj: " + t5.OutDB().getVariable("z").findRecord().getLevel());
}
...
```

Note

> Some of demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

See also **Transport5.java**.

7.10.3.17 How to run multiple GAMSJobs in parallel using a GAMSCheckpoint (Transport6)

This example illustrates how to run the jobs known from **Transport5** in parallel. We initialize the **GAMSCheckpoint** cp and introduce a demand multiplier as we did before :

```
...
  GAMSJob t6 = ws.addJobFromString(model);
  t6.run(cp);
  double[] bmultlist = new double[] { 0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3 };
...
```

Furthermore, we introduce a new object `lockObject` that will be used to avoid mixed up output from the parallel jobs. We create one scenario for each entry of `bmultlist` and cause a thread to begin execution.

```
...
  // run multiple parallel jobs using the created checkpoint
  Object lockObject = new Object();
  Scenario[] scenarios = new Scenario[bmultlist.length];
  for (int i=0; i<bmultlist.length; i++)
  {
     scenarios[i] = new Scenario(ws, cp, lockObject, bmultlist[i]);
     scenarios[i].start();
```

```
  }
...
```

In class `Scenario` a **GAMSJob** `t6` is created and run just like in the preceding example of **Transport5**. The output section is also the same except for the fact that it is 'locked' by the object `lockObject` which means that the output section cannot be executed simultaneously for multiple demand multipliers.

```
...
static class Scenario extends Thread
{
   GAMSWorkspace workspace;
   GAMSCheckpoint checkpoint;
   Object lockObject;
   double bmult;
   public Scenario(GAMSWorkspace ws, GAMSCheckpoint cp, Object lockObj, double b)
   {
      workspace = ws;
      checkpoint = cp;
      lockObject = lockObj;
      bmult = b;
   }
   public void run()
   {
      GAMSJob t6 = workspace.addJobFromString(
                     "bmult=" + bmult +"; solve transport min z use lp; ms=transport.modelstat;ss=transport.solvestat;",
                     checkpoint
                  );
      t6.run();
      // we need to make the output a critical section to avoid messed up report information
      synchronized (lockObject)
      {
          System.out.println("Scenario bmult=" + bmult + ":");
          System.out.println(
                " Modelstatus: " + GAMSGlobals.ModelStat.lookup(
                                      (int) t6.OutDB().getParameter("ms").findRecord().getValue()
                                    )
          );
          System.out.println(
              " Solvestatus: " + GAMSGlobals.SolveStat.lookup(
                                      (int)t6.OutDB().getParameter("ss").findRecord().getValue()
                                   )
          );
          System.out.println(" Obj: " + t6.OutDB().getVariable("z").findRecord().getLevel());
       }
    }
}
...
```

While the output in **Transport5** is strictly ordered subject to the order of the elements of `bmultlist` in **Transport6** the output blocks might change their order but the blocks describing one scenario are still appearing together due to the `lockObject`.

If you want a further impression of the impact of the `lockObject`, just rerun **Transport6** but comment out the lock as follows and compare the output.

```
...
      // synchronized (lockObject)
      // {
          System.out.println("Scenario bmult=" + bmult + ":");
          System.out.println(
              " Modelstatus: " + GAMSGlobals.ModelStat.lookup(
                                      (int) t6.OutDB().getParameter("ms").findRecord().getValue()
                                   )
          );
          System.out.println(
              " Solvestatus: " + GAMSGlobals.SolveStat.lookup(
                                    (int)t6.OutDB().getParameter("ss").findRecord().getValue()
                                   )
          );
          System.out.println(" Obj: " + t6.OutDB().getVariable("z").findRecord().getLevel());
       // }
...
```

See also **Transport6.java**.

7.10.3.18 How to create a GAMSModelInstance from a GAMSCheckpoint (Transport7)

In **Transport7** the usage of **com::gams::api::GAMSModelInstance** is demonstrated.

At first checkpoint `cp` is created as in the preceding examples. Then we create the **GAMSModelInstance** `mi` using the **GAMSCheckpoint.addModelInstance** method. Note that the **GAMSJob** again contains no solve statement and the demand multiplier is already included with default value 1.

```
...
  GAMSCheckpoint cp = ws.addCheckpoint();
  // initialize a checkpoint by running a job
  GAMSJob t7 = ws.addJobFromString(model);
  t7.run(cp);
  GAMSModelInstance mi = cp.addModelInstance();
  ...
```

See also **Transport7.java**.

7.10.3.19 How to modify a parameter of a GAMSModelInstance using GAMSModifier (Transport7)

A **GAMSModelInstance** uses a `SyncDB` to maintain the data. We define `bmult` as **GAMSParameter** using the **GAMSParameter** method and specify gurobi as solver. Afterwards the **GAMSModelInstance** is instantiated with 3 arguments, the solve statement, **GAMSOptions** `opt` and **GAMSModifier** `bmult`. The **GAMSModifier** means that `bmult` is modifiable while all other parameters, variables and equations of ModelInstance `mi` stay unchanged. We use the **GAMSParameter.addRecord** method and the **setValue** function to assign a value to `bmult`. That value can be varied afterwards using the **GAMSParameter.getFirstRecord** method to reproduce our well-known example with different demand multipliers.

```
...
  GAMSParameter bmult = mi.SyncDB().addParameter("bmult", 0,"demand multiplier");
  GAMSOptions opt = ws.addOptions();
  opt.setAllModelTypes("gurobi");
  // instantiate the ModelInstance and pass a model definition and Modifier to declare bmult mutable
  mi.instantiate("transport use lp min z", opt, new GAMSModifier(bmult));
  bmult.addRecord().setValue( 1.0 );
  double[] bmultlist = new double[] { 0.6, 0.7 , 0.8, 0.9, 1.0,1.1, 1.2, 1.3 };
  for (double b : bmultlist)
  {
     bmult.getFirstRecord().setValue( b );
     mi.solve();
     System.out.println("Scenario bmult=" + b + ":");
     System.out.println(" Modelstatus: " + mi.getModelStatus());
     System.out.println(" Solvestatus: " + mi.getSolveStatus());
     System.out.println(" Obj: " + mi.SyncDB().getVariable("z").findRecord().getLevel());
  }
...
```

See also **Transport7.java**.

7.10.3.20 How to modify a variable of a GAMSModelInstance using GAMSModifier (Transport7)

We create a **GAMSModelInstance** just like in the next to last example. We define `x` as **GAMSVariable** and its upper bound as **GAMSParameter** `xup`. At the following **instantiate** method **GAMSModifier** has 3 arguments. The first one says that `x` is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the **GAMSParameter** that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```
...
  mi = cp.addModelInstance();
  GAMSVariable x = mi.SyncDB().addVariable("x", 2, GAMSGlobals.VarType.POSITIVE, "");
  GAMSParameter xup = mi.SyncDB().addParameter("xup", 2, "upper bound on x");
  // instantiate the ModelInstance and pass a model definition and Modifier to declare upper bound of X mutable
  mi.instantiate("transport use lp min z", new GAMSModifier(x, GAMSGlobals.UpdateAction.UPPER, xup));
  for (GAMSSetRecord i : t7.OutDB().getSet("i"))
  {
     for (GAMSSetRecord j : t7.OutDB().getSet("j"))
     {
```

```
        xup.clear();
        String[] keys = { i.getKeys()[0], j.getKeys()[0] };
        xup.addRecord(keys).setValue(0);
        mi.solve();
        System.out.println("Scenario link blocked: " + i.getKeys()[0] + " - " + j.getKeys()[0]);
        System.out.println(" Modelstatus: " + mi.getModelStatus());
        System.out.println(" Solvestatus: " + mi.getSolveStatus());
        System.out.println(" Obj: " + mi.SyncDB().getVariable("z").findRecord().getLevel());
      }
  }
...
```

See also **Transport7.java**.

7.10.3.21 How to use a queue to solve multiple GAMSModelInstances in parallel (Transport8)

We initialize a **GAMSCheckpoint** `cp` from a **GAMSJob**. Then we define a queue that represents the different values of the demand multiplier. A queue follows the first-in-first-out principle The object `IOLockObject` is used later to avoid messed up output. Then we call `Scenarios` multiple times in parallel. The number of parallel calls is specified by `numberOfWorkers`.

```
...
  GAMSJob t8 = ws.addJobFromString(model);
  t8.run(cp);
  Queue<Double> bmultQueue = new LinkedList<Double>(
                    Arrays.asList( Double.valueOf(0.6), Double.valueOf(0.7),
                    Double.valueOf(0.8), Double.valueOf(0.9),
                    Double.valueOf(1.0), Double.valueOf(1.1),
                    Double.valueOf(1.2), Double.valueOf(1.3) )
  );
  // solve multiple model instances in parallel
  Object IOLockObject = new Object();
  int numberOfWorkers = 2;
  Scenarios[] scenarios = new Scenarios[numberOfWorkers];
  for (int i=0; i<numberOfWorkers; i++)
  {
     scenarios[i] = new Scenarios( ws, cp, bmultQueue, IOLockObject, i );
     scenarios[i].start();
  }
...
```

In class `Scenarios` we create and instantiate a **GAMSModelInstance** as in the preceding examples and make parameter `bmult` modifiable. Note that we chose `cplex` as solver because it is thread safe (gurobi would also be possible). Once the queue is empty the loop terminates.

```
...
static class Scenarios extends Thread
{
   GAMSWorkspace workspace;
   GAMSCheckpoint checkpoint;
   Object IOLockObject;
   Queue<Double> bmultQueue;
   int workerNumber;
   public Scenarios(GAMSWorkspace ws, GAMSCheckpoint cp, Queue<Double> que, Object IOLockObj, int i)
   {
        workspace = ws;
        checkpoint = cp;
        IOLockObject = IOLockObj;
        bmultQueue = que;
        workerNumber = i;
   }
   public void run()
   {
       GAMSModelInstance mi = checkpoint.addModelInstance();
       {
          GAMSParameter bmult = mi.SyncDB().addParameter("bmult", 0, "demand multiplier");
          GAMSOptions opt = workspace.addOptions();
          opt.setAllModelTypes("cplex");
          // instantiate the GAMSModelInstance and pass a model definition and GAMSModifier to declare bmult mutable
          mi.instantiate("transport use lp min z", opt, new
          GAMSModifier(bmult));
          bmult.addRecord().setValue( 1.0 );
          while (true)
              double b = 0.0;
              // dynamically get a bmult value from the queue instead of passing it to the different threads at creation
      time
              synchronized (bmultQueue)
                  if (bmultQueue.isEmpty())
                      break;
```

```
                else
                    b = bmultQueue.remove();
            }
            bmult.getFirstRecord().setValue(b);
            mi.solve();
            // we need to make the output a critical section to avoid messed up report informations
            synchronized (IOLockObject)
            {
               System.out.println("#"+workerNumber+":Scenario bmult=" + b + ":");
               System.out.println(" Modelstatus: " + mi.getModelStatus());
               System.out.println(" Solvestatus: " + mi.getSolveStatus());
               System.out.println(" Obj: " + mi.SyncDB().getVariable("z").findRecord().getLevel());
            }
         }
      }
   }
}
...
```

See also **Transport8.java**.

7.10.3.22 How to fill a GAMSDatabase by reading from MS Access (Transport9)

This example illustrates how to import data from Microsoft Access to a **GAMSDatabase** (on Windows platform only). There are a few prerequisites required to run **Transport9** successfully.

- Install a Microsoft access driver on your machine (if none is available) as the driver must be loaded at runtime to connect to a data source file.
- To access the data source file go to Control `Panel\System` and `Security\Administrative Tools\Data Sources` (ODBC) and set up a data source name (DSN) as "transportdsn", leave out user ID and password, and select `[PathToGAMS]/apfiles/Data/tansport.accdb` as the data source file.
- Note that an architecture mismatch between the Driver and Application might cause problems.

We call a function `readDataFromAccess` that finally returns a **GAMSDatabase** as shown below.

```
...
  GAMSDatabase db = readDataFromAccess(ws);
...
```

The function `readDataFromAccess` begins with the creation of an empty database. Afterwards we set up a connection to the MS Access database by specifying the aforementioned data source name (DSN). To finally read in GAMS sets and parameters we call the functions `readSet` and `readParameter`.

```
static GAMSDatabase readDataFromAccess(GAMSWorkspace ws)
{
    GAMSDatabase db = ws.addDatabase();
    try {
         // loading the jdbc odbc driver
         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
         // creating connection to database
         Connection c = DriverManager.getConnection("jdbc:odbc:transportdsn","","");
         // read GAMS sets
         readSet(c, db, "SELECT Plant FROM Plant", "i", 1, "canning plants");
         readSet(c, db, "SELECT Market FROM Market", "j", 1, "markets");
         // read GAMS parameters
         readParameter(c, db, "SELECT Plant, Capacity FROM Plant", "a", 1, "capacity of plant i in cases");
         readParameter(c, db, "SELECT Market,Demand FROM Market", "b", 1, "demand at market j in cases");
         readParameter(c, db, "SELECT Plant,Market,Distance FROM Distance", "d", 2, "distance in thousands of miles");
         c.close();
    } catch (ClassNotFoundException e) {
         System.err.println("Error: Failed to find a driver for the database.");
         e.printStackTrace();
         System.exit(-1);
    } catch (SQLException e) {
         System.err.println("Error: Failed to retrieve data from the database.");
         e.printStackTrace();
         System.exit(-1);
    }
    return db;
}
```

The function `readSet` adds a set to the **GAMSDatabase** that is filled with the data from the MS Access file afterwards. The function `readParameter` works quite similar.

```
static void readSet(Connection c, GAMSDatabase db, String queryString, String setName, int setDimension, String
      setExplanatoryText) throws SQLException
{
    Statement st = c.createStatement();
    ResultSet rs = st.executeQuery(queryString);
    ResultSetMetaData rsmd = rs.getMetaData();
    if (rsmd.getColumnCount() != setDimension)
    {
        System.err.println("Error: Number of fields in select statement does not match setDimemsion.");
        c.close();
        System.exit(-1);
    }
    GAMSSet set = db.addSet(setName, setDimension, setExplanatoryText);
    String[] keys = new String[setDimension];
    while (rs.next())
    {
        for (int idx=0; idx < setDimension; idx++)
            keys[idx] = rs.getString(idx+1);
        set.addRecord( keys );
    }
    st.close();
}
```

Once we read in all the data we can create a **GAMSJob** from the **GAMSDatabase** and run it as usual.

See also **Transport9.java**.

7.10.3.23 How to fill a GAMSDatabase by reading from MS Excel (Transport10)

This example illustrates how to read data from Excel, or to be more specific, from `[PathToGAMS]\apifiles\Data\transport`

At first we have to add an additional jar file to the `[CLASSESPATH]` since we use the Java Excel API which can be donwnloaded from jexcelapi. We unzipped the Java Excel API folder in `C:\tools`. To compile and run on a Windows platform from the command line go to `C:\GAMS\win64\24.1\apifiles\Java\transport` and use:

```
javac -cp C:\GAMS\win64\GAMS_VERSION\apifiles\Java\api\GAMSJavaAPI.jar;C:\tools\jexcelapi\jxl.jar;. t
```

and

```
java -cp C:\GAMS\win64\GAMS_VERSION\apifiles\Java\api\GAMSJavaAPI.jar;C:\tools\jexcelapi\jxl.jar;. c
```

If you are using an IDE like Eclipse add `C:\tools\jexcelapi\jxl.jar` to the class path as explained in the Compiling a Program from Java IDE and Running a Program from Java IDE section.

The model is given as string without data like in many examples before and the Excel file `transport.xlsx` is located at `[gamsdir]\apifiles\Data`.

At first we define the input string and create the corresponding input file.

```
...
  String input = gamsdir + "apifiles" +
  GAMSGlobals.FILE_SEPARATOR + "Data" + GAMSGlobals.FILE_SEPARATOR + "transport.xlsx";
  File inputFile = new File(input);
...
```

The following lines address the different worksheets and read in the contained data.

```
...
  int iCount = 0;
  int jCount = 0;
  String[][] capacityData = null;
  String[][] demandData = null;
  String[][] distanceData = null;
  Workbook w;
```

```
  try {
      w = Workbook.getWorkbook(inputFile);
      Sheet capacity = w.getSheet("capacity");
      capacityData = new String[capacity.getRows()][capacity.getColumns()];
      iCount = capacity.getColumns();
      for (int j = 0; j < capacity.getColumns(); j++)
         for (int i = 0; i < capacity.getRows(); i++)
            capacityData[i][j] = capacity.getCell(j, i).getContents();
      Sheet demand = w.getSheet("demand");
      demandData = new String[demand.getRows()][demand.getColumns()];
      jCount = demand.getColumns();
      for (int j = 0; j < demand.getColumns(); j++)
         for (int i = 0; i < demand.getRows(); i++)
          demandData[i][j] = demand.getCell(j, i).getContents();
      Sheet distance = w.getSheet("distance");
      distanceData = new String[distance.getRows()][distance.getColumns()];
      for (int j = 0; j < distance.getColumns(); j++)
         for (int i = 0; i < distance.getRows(); i++)
           distanceData[i][j] = distance.getCell(j, i).getContents();
      w.close();
  } catch (IOException e) {
     e.printStackTrace();
  } catch (BiffException e) {
     e.printStackTrace();
  }
...
```

Now we can create the **GAMSWorkspace** as usual and afterwards create a **GAMSDatabase** and fill it with the workbook data as follows:

```
...
  GAMSDatabase db = ws.addDatabase();
  GAMSSet i = db.addSet("i", 1, "Plants");
  GAMSSet j = db.addSet("j", 1, "Markets");
  GAMSParameter capacityParam = db.addParameter("a", 1, "Capacity");
  GAMSParameter demandParam = db.addParameter("b", 1, "Demand");
  GAMSParameter distanceParam = db.addParameter("d", 2, "Distance");
  for (int ic = 0; ic < iCount; ic++)
  {
    i.addRecord( capacityData[0][ic] );
    capacityParam.addRecord( capacityData[0][ic] ).setValue( Double.valueOf(capacityData[1][ic]).doubleValue() );
  }
  for (int jc = 0; jc < jCount; jc++)
  {
     j.addRecord( demandData[0][jc] );
     demandParam.addRecord( demandData[0][jc] ).setValue( Double.valueOf(demandData[1][jc]).doubleValue() );
     String[] data = null;
     for (int ic = 0; ic < iCount; ic++)
     {
        data =  new String[] { distanceData[ic+1][0], distanceData[0][jc+1] };
        distanceParam.addRecord( data ).setValue( Double.valueOf(distanceData[ic+1][jc+1]) );
     }
  }
```

Note that we can name sets and parameters just like in the database but we don't have to. Now we can run our **GAMSJob** as usual.

```
...
  GAMSOptions opt = ws.addOptions();
  GAMSJob t10 = ws.addJobFromString(model);
  opt.defines("gdxincname", db.getName());
  opt.setAllModelTypes("xpress");
  t10.run(opt, db);
  for (GAMSVariableRecord rec : t10.OutDB().getVariable("x"))
     System.out.println("x(" + rec.getKeys()[0] + "," + rec.getKeys()[1] + "): level=" + rec.getLevel() + " marginal=" +
      rec.getMarginal());
...
```

See also **Transport10.java**.

7.10.3.24 How to create and use a save/restart file (Transport11)

In **Transport11** we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a **GAMSCheckpoint**. From the main function we call the function `CreateSaveRestart` giving it the current working directory and the desired file name as arguments.

```
...
  CreateSaveRestart( workingDirectory, "tbase" );
...
```

In function `CreateSaveRestart` create a workspace with the given working directory. Then we create a **GAMSJob** from a string. Note that the string given via `baseModel` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a **GAMSOptions** to only compile the job but do not execute it. Then we create a checkpoint `cp` that is initialized by the following run of the **GAMSJob** and stored in the file given as argument to the function, in our case `tbase`. This becomes possible because the **addCheckpoint** method accepts identifiers as well as file names as argument.

```
static void CreateSaveRestart(File workingDirectory, String cpFileName)
{
   GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
   wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());
   GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
   GAMSJob j1 = ws.addJobFromString(baseModel);
   GAMSOptions opt = ws.addOptions();
   opt.setAction( GAMSOptions.EAction.CompileOnly );
   GAMSCheckpoint cp = ws.addCheckpoint(cpFileName);
   j1.run(opt, cp);
   opt.dispose();
}
```

So what you should keep in mind before we return to further explanations of the main function is, that the file tbase is now in the current working directory and contains a checkpoint. Now in the main function we define some data using Java data structures as we already did in **Transport4** before we create the **GAMSWorkspace** and a **GAMSDatabase**.

```
...
  GAMSWorkspaceInfo wsInfo = new GAMSWorkspaceInfo();
  wsInfo.setWorkingDirectory(workingDirectory.getAbsolutePath());
  GAMSWorkspace ws = new GAMSWorkspace(wsInfo);
  GAMSDatabase db = ws.addDatabase();
...
```

Afterwards we set up the **GAMSDatabase** like we already did in **Transport4**. Once this is done we run a **GAMSJob** using this data plus the checkpoint stored in file tbase.

```
...
  GAMSCheckpoint cpBase = ws.addCheckpoint("tbase");
  GAMSOptions opt = ws.addOptions();
  GAMSJob t11 = ws.addJobFromString(model, cpBase);
  opt.defines("gdxincname", db.getName());
  opt.setAllModelTypes("xpress");
  t11.run(opt, db);
...
```

Note that the string from which we create job `t11` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the **GAMSDatabase** correctly. The entire model definition is delivered by the checkpoint `cpBase` which is equal to the one we saved in `tbase`.

See also **Transport11.java**.

7.11 Control

GAMS `control` is a sub-module of the Python API that allows for full control over the GAMS system (data, model instances, and solving). It can be used in conjunction with other Python API sub-modules (`numpy` and `transfer`) to boost performance when pushing/pulling data to/from a GAMS model. The `gams.control` package provides objects to interact with the General Algebraic Modeling System (GAMS). Objects in this package allow convenient exchange of input data and model results (`GamsDatabase`), help to create and run GAMS models (`GamsJob`), that can be customized by GAMS options (`GamsOptions`). Furthermore, it introduces a way to solve a sequence of closely related model instances in the most efficient way (`GamsModelInstance`).

A GAMS program can include other source files (e.g. $include), load data from GDX files (e.g. $GDXIN or execute_load), and create PUT files. All these files can be specified with a (relative) path and therefore an anchor into the file system is required. The base class GamsWorkspace manages the anchor to the file system. If external file communication is not an issue in a particular Python application, temporary directories and files will be managed by objects in the namespace.

With the exception of GamsWorkspace the objects in the `gams.control` package cannot be accessed across different threads unless the instance is locked. The classes themself are thread safe and multiple objects of the class can be used from different threads (see below for restrictions on solvers that are not thread safe within the GamsModelInstance class).

Note

If you use multiple instances of the GamsWorkspace in parallel, you should avoid using the same working directory. Otherwise you may end up with conflicting file names.

The GAMS `control` Python API lacks support for the following GAMS components: Acronyms, support for GAMS compilation/execution errors (GamsJob.run just throws an exception), structured access to listing file, and proper support for solver options.

Currently only Cplex, Gurobi, and SoPlex fully utilize the power of solving GamsModelInstances. Some solvers will not even work in a multi-threaded application using GamsModelInstances. For some solvers this is unavoidable because the solver library is not thread safe (e.g. MINOS), other solvers are in principle thread safe but the GAMS link is not (e.g. SNOPT). Moreover, GamsModelInstances are not available for quadratic model types (QCP, MIQCP, RMIQCP).

7.11.1 Recommended Import

GAMS `control` is available with the following import statement once the API has been installed:

Other sub-modules must be imported with separate import statements.

7.11.1.1 Specifying a GAMS System Directory

There are several ways to specify which system directory should be used. On all platforms, the system directory can be specified in the GamsWorkspace constructor. If no system directory is specified by the user, The API tries to find one automatically:

- Windows: Try to find a system directory in the Windows registry.
- Linux: Try to find a system directory in the `PATH` first. If none was found, search `LD_LIBRARY_PATH`.
- OS X: Try to find a system directory in the `PATH` first. If none was found, search `DYLD_LIBRARY_PATH`.

The environment variable `PATH` can be set as follows on Linux and macOS:

```
export PATH=<Path/To/GAMS>:$PATH
```

Note

On Linux and macOS it is recommended to specify the `PATH` only instead of `(DY)LD_LIBRARY_PATH` since this might cause problems loading the correct version of certain modules (e.g. gdx).

7.11.2 Important Classes of the API

This section provides a quick overview of some fundamental classes of the GAMS `control` API. Their usage is demonstrated by an extensive set of examples in the How to use the API section.

- **gams::control::workspace::GamsWorkspace** Class
- **gams::control::execution::GamsJob** Class
- **gams::control::database::GamsDatabase** Class
- **gams::control::options::GamsOptions** Class
- **gams::control::execution::GamsModelInstance** Class

7.11.3 How to use the API

The GAMS distribution provides several examples that illustrate the usage of the API. `[GAMSDIR]\api\python\examples\c` contains multiple examples dealing with the well-known `transportation problem`. In further course of this tutorial we discuss these examples step by step and introduce new elements of the API in detail.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- How to import packages/modules from the GAMS control API (transport1.py)
- How to choose the GAMS system (transport1.py)
- How to export data to GDX (transport_gdx.py)
- How to import data from GDX (transport_gdx.py)
- How to run a GamsJob from file (transport1.py)
- How to retrieve a solution from an output database (transport1.py)
- How to specify the solver using GamsOptions (transport1.py)
- How to run a job with a solver option file and capture its log output (transport1.py)
- How to use include files (transport2.py)
- How to read data from string and export to GDX (transport3.py)
- How to run a job using data from GDX (transport3.py)
- How to run a job using implicit database communication (transport3.py)
- How to define data using Python data structures (transport4.py)
- How to prepare a GamsDatabase from Python data structures (transport4.py)
- How to initialize a GamsCheckpoint by running a GamsJob (transport5.py)
- How to initialize a GamsJob from a GamsCheckpoint (transport5.py)
- How to run multiple GamsJobs in parallel using a GamsCheckpoint (transport6.py)
- How to create a GamsModelInstance from a GamsCheckpoint (transport7.py)
- How to modify a parameter of a GamsModelInstance using GamsModifier (transport7.py)
- How to modify a variable of a GamsModelInstance using GamsModifier (transport7.py)
- How to use a queue to solve multiple GamsModelInstances in parallel (transport8.py)
- How to fill a GamsDatabase by reading from MS Access (transport9.py)
- How to fill a GamsDatabase by reading from MS Excel (transport10.py)
- How to create and use a save/restart file (transport11.py)

7.11.3.1 How to import packages/modules from the GAMS control API (transport1.py)

Before we can start using the GAMS `control` API, it needs to be installed by following the instructions from the Getting Started section. Afterwards we can use the API by importing the `GamsWorkspace` class like this:

```
from gams import GamsWorkspace
```

Conventional Python packages/modules can by imported like that:

```
import os
import sys
```

7.11.3.2 How to choose the GAMS system (transport1.py)

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. If we type `python transport1.py` C:/GAMS/42 we use `GAMS 42` to run `transport1.py` even if our default GAMS system might be a different one. This is managed by the following code:

```
...
sys_dir = sys.argv[1] if len(sys.argv) > 1 else None
ws = GamsWorkspace(system_directory=sys_dir)
...
```

7.11.3.3 How to export data to GDX (transport_gdx.py)

Although the GAMS `control` Python API offers much more than exchanging data between Python and GDX, a common use case is the export and import of GDX files. The central class for this purpose is GamsDatabase. We assume that the data to be exported is available in Python data structures.

```
...
    plants = ["Seattle", "San-Diego"]
    markets = ["New-York", "Chicago", "Topeka"]
    capacity = {"Seattle": 350.0, "San-Diego": 600.0}
    demand = {"New-York": 325.0, "Chicago": 300.0, "Topeka": 275.0}
    distance = {
        ("Seattle", "New-York"): 2.5,
        ("Seattle", "Chicago"): 1.7,
        ("Seattle", "Topeka"): 1.8,
        ("San-Diego", "New-York"): 2.5,
        ("San-Diego", "Chicago"): 1.8,
        ("San-Diego", "Topeka"): 1.4,
    }
...
```

Different GAMS symbols are represented using different Python data structures. The data for the GAMS sets is represented using Python lists of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by Python dictionaries (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses Python tuples for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with standard Python data structures.

A new GamsDatabase instance can be created using GamsWorkspace.add_database.

```
...
    # create new GamsDatabase instance
    db = ws.add_database()
...
```

We start adding GAMS sets using the method GamsDatabase.add_set which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created GamsSet instance `i` using GamsSet.add_record.

```
...
    # add 1-dimensional set 'i' with explanatory text 'canning plants' to the GamsDatabase
    i = db.add_set("i", 1, "canning plants")
    for p in plants:
        i.add_record(p)
...
```

GamsParameter instances can be added by using the method GamsDatabase.add_parameter. It has the same signature as GamsDatabase.add_set. Anyhow, in this example we use GamsDatabase.add_parameter_dc instead which takes a list of GamsSet instances instead of the dimension for creating a parameter with domain information.

```
...
    # add parameter 'a' with domain 'i'
    a = db.add_parameter_dc("a", [i], "capacity of plant i in cases")
    for p in plants:
        a.add_record(p).value = capacity[p]
...
```

As soon as all data is prepared in the GamsDatabase, the method GamsDatabase.export can be used to create a GDX file.

```
...
    # export the GamsDatabase to a GDX file with name 'data.gdx' located in the 'working_directory' of the GamsWorkspace
    db.export("data.gdx")
...
```

7.11.3.4 How to import data from GDX (transport_gdx.py)

Data can be imported from a GDX file using GamsWorkspace.add_database_from_gdx. The method takes a path to a GDX file and creates a GamsDatabase instance.

```
...
    # add a new GamsDatabase and initialize it from the GDX file just created
    db2 = ws.add_database_from_gdx("data.gdx")
...
```

Reading the data from the GamsSet `i` into a list can be done as follows:

```
...
    # read data from symbols into Python data structures
    i = [rec.keys[0] for rec in db2["i"]]
...
```

A Python list is created using list comprehensions. `i` is retrieved by querying the GamsDatabase `db2`. The returned GamsSet object can be iterated using a for-loop to access the records of the set. Each record is of type GamsSetRecord and can be asked for its keys.

You can do the same for GamsParameters. Instead of creating a Python list, we want to have the data in the form of a Python dictionary. GamsParameterRecords can not only be asked for their keys, but also for their value. The following code snippet shows how to read the one dimensional parameter `a` into a Python dictionary using dict comprehensions.

```
...
    a = {rec.keys[0]: rec.value for rec in db2["a"]}
...
```

For multi dimensional symbols, we choose the Python dictionary keys to be tuples instead of string. We access the keys as usual, but do not address a specific key. Instead, we take the whole list of keys and turn it into a tuple.

```
...
    d = {tuple(rec.keys): rec.value for rec in db2["d"]}
...
```

Scalars can be read into a Python identifier by accessing the value of the first and only record.

```
...
    f = db2["f"].first_record().value
...
```

7.11.3.5 How to run a GamsJob from file (transport1.py)

At first we create our workspace using `ws = GamsWorkspace()`. Afterwards we load the `trnsport` model from the GAMS model library which puts the corresponding gms file in our working directory. Note that you can create a GamsJob with any other gms file you might have created on your own as long as it is located in the current working directory. Then the GamsJob `job` can be defined using the add_job_from_file method and afterwards we run the job.

```
...
    ws.gamslib("trnsport")
    job = ws.add_job_from_file("trnsport.gms")
    job.run()
...
```

7.11.3.6 How to retrieve a solution from an output database (transport1.py)

The following lines create the solution output and illustrate the usage of the GamsJob.out_db property to get access to the GamsDatabase created by the run method. To retrieve the content of variable `x` we use squared brackets that internally call the get_symbol method.

```
...
    for rec in job.out_db["x"]:
        print(
            f"x({rec.key(0)},{rec.key(1)}): level={rec.level} marginal={rec.marginal}"
        )
...
```

Note that instead of using the squared brackets we could also use

```
...
    for rec in job.out_db.get_symbol("x"):
...
```

7.11.3.7 How to specify the solver using GamsOptions (transport1.py)

The solver can be specified via the GamsOptions class and the GamsWorkspace.add_options method. The GamsOptions.all_model_types property sets xpress as default solver for all model types that can be handled by the solver. Then we run our GamsJob `job` with the new GamsOption.

```
...
    opt = ws.add_options()
    opt.all_model_types = "xpress"
    job.run(opt)
...
```

7.11.3.8 How to run a job with a solver option file and capture its log output (transport1.py)

At first we create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. Afterwards we use a GamsOption just like in the preceding example and set GamsOption.optfile property to 1 to tell the solver to look for a solver option file. In addition, we specify the argument `output` in order to write the log of the GamsJob into the file `transport1_xpress.log`.

```
...
    with open(os.path.join(ws.working_directory, "xpress.opt"), "w") as file:
        file.write("algorithm=barrier")
    opt.optfile = 1
    with open("transport1_xpress.log", "w") as log:
        job.run(opt, output=log)
...
```

Instead of writing the log output to a file, any object that provides the functions `write` and `flush` can be used. In order to write the log directly to `stdout`, we can use the following code:

```
...
    job.run(opt, output=sys.stdout)
...
```

7.11.3.9 How to use include files (transport2.py)

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings accessed via `GAMS_DATA` and `GAMS_MODEL` are using GAMS syntax. At first we write an include file `tdata.gms` that contains the data but not the model text and save it in our current working directory.

```
...
    with open(os.path.join(ws.working_directory, "tdata.gms"), "w") as file:
        file.write(GAMS_DATA)
...
```

Afterwards we create a GamsJob using the GamsWorkspace.add_job_from_string method. GamsOptions.defines is used like the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line where `incname` is used as name for the include file in `GAMS_MODEL` as shown below.

```
...
    job = ws.add_job_from_string(GAMS_MODEL)
    opt = ws.add_options()
    opt.defines["incname"] = "tdata"
    job.run(opt)
...
```

The string `GAMS_MODEL` contains the following lines to read in the data.

```
...
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
...
```

7.11.3.10 How to read data from string and export to GDX (transport3.py)

We read the data from the string `GAMS_DATA`. Note that this contains no model but only data definition in GAMS syntax. By running the corresponding GamsJob a GamsDatabase is created that is available via the GamsJob.out_db property. We can use the GamsDatabase.export method to write the content of this database to a GDX file `tdata.gdx` in the current working directory.

```
...
    job = ws.add_job_from_string(GAMS_DATA)
    job.run()
    job.out_db.export(os.path.join(ws.working_directory, "tdata.gdx"))
...
```

7.11.3.11 How to run a job using data from GDX (transport3.py)

This works quite similar to the usage of an include file explained in How to use include files (transport2.py).

```
...
    job = ws.add_job_from_string(GAMS_MODEL)
    opt = ws.add_options()
    opt.defines["gdxincname"] = "tdata"
    opt.all_model_types = "xpress"
    job.run(opt)
...
```

Note that there are some minor changes in `GAMS_MODEL` compared to preceding examples due to the usage of a GDX instead of an include file.

```
...
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxIn %gdxincname%
$load i j a b d f
$gdxIn
...
```

7.11.3.12 How to run a job using implicit database communication (transport3.py)

This example does basically the same as the two preceding examples together. We create two GamsJobs `job_data` and `job_model` where the first one contains only the data and the second one contains only the model without data. After running `job_data` the corresponding `out_db` can be read in directly just like a GDX file. Note that the database needs to be passed to the GamsJob.run method as additional argument.

```
...
    job_data = ws.add_job_from_string(GAMS_DATA)
    job_model = ws.add_job_from_string(GAMS_MODEL)
    job_data.run()
    opt.defines["gdxincname"] = job_data.out_db.name
    job_model.run(opt, databases=job_data.out_db)
...
```

7.11.3.13 How to define data using Python data structures (transport4.py)

We use Python lists to define the sets and Python dictionaries for the parameter definition.

```
...
    plants = ["Seattle", "San-Diego"]
    markets = ["New-York", "Chicago", "Topeka"]
    capacity = {"Seattle": 350.0, "San-Diego": 600.0}
    demand = {"New-York": 325.0, "Chicago": 300.0, "Topeka": 275.0}
    distance = {
        ("Seattle", "New-York"): 2.5,
        ("Seattle", "Chicago"): 1.7,
        ("Seattle", "Topeka"): 1.8,
        ("San-Diego", "New-York"): 2.5,
        ("San-Diego", "Chicago"): 1.8,
        ("San-Diego", "Topeka"): 1.4,
    }
...
```

7.11.3.14 How to prepare a GamsDatabase from Python data structures (transport4.py)

At first we create an empty GamsDatabase db using the GamsWorkspace.add_database method. Afterwards we prepare the database. To add a set to the database we use the GamsSet class and the GamsDatabase.add_set method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our Python data structure and add them by using the GamsSet.add_record method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2.

The GamsJob can be run like explained in the preceding example How to run a job using implicit database communication

```
...
    db = ws.add_database()
    i = db.add_set("i", 1, "canning plants")
    for p in plants:
        i.add_record(p)
    j = db.add_set("j", 1, "markets")
    for m in markets:
        j.add_record(m)
    a = db.add_parameter_dc("a", [i], "capacity of plant i in cases")
    for p in plants:
        a.add_record(p).value = capacity[p]
    b = db.add_parameter_dc("b", [j], "demand at market j in cases")
    for m in markets:
        b.add_record(m).value = demand[m]
    d = db.add_parameter_dc("d", [i, j], "distance in thousands of miles")
    for k, v in distance.items():
        d.add_record(k).value = v
    f = db.add_parameter("f", 0, "freight in dollars per case per thousand miles")
    f.add_record().value = 90
    job = ws.add_job_from_string(GAMS_MODEL)
    opt = ws.add_options()
    opt.defines["gdxincname"] = db.name
    opt.all_model_types = "xpress"
    job.run(opt, databases=db)
...
```

7.11.3.15 How to initialize a GamsCheckpoint by running a GamsJob (transport5.py)

The following lines of code conduct several operations. While the first line simply creates a GamsCheckpoint, the second one uses the GamsWorkspace.add_job_from_string method to create a GamsJob containing the model text and data but no solve statement. Afterwards the run method gets the GamsCheckpoint as argument. That means the GamsCheckpoint `cp` captures the state of the GamsJob.

```
...
    cp = ws.add_checkpoint()
    job = ws.add_job_from_string(GAMS_MODEL)
    job.run(checkpoint=cp)
...
```

7.11.3.16 How to initialize a GamsJob from a GamsCheckpoint (transport5.py)

Note that the string `GAMS_MODEL` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```
...
   bmult 'demand multiplier' / 1 /;
...
   demand(j) 'satisfy demand at market j';
...
Scalar ms 'model status', ss 'solve status';
...
```

In transport5.py we create a list with eight different values for this demand multiplier.

```
...
    bmult = [0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3]
...
```

For each entry of that list we create a GamsJob using the GamsWorkspace.add_job_from_string method. Besides another string which resets the demand multiplier `bmult`, specifies the solve statement and assigns

values to the scalars `ms` and `ss` we pass the checkpoint `cp` as additional argument. This results in a GamsJob combined from the checkpoint plus the content provided by the string.

We run the GamsJob and print some interesting data from the out_db.

```
...
    for b in bmult:
        job = ws.add_job_from_string(
            f"bmult={b}; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;",
            cp,
        )
        job.run()
        print(f"Scenario bmult={b}:")
        print(f"  Modelstatus: {job.out_db['ms'].find_record().value}")
        print(f"  Solvestatus: {job.out_db['ss'].find_record().value}")
        print(f"  Obj: {job.out_db['z'].find_record().level}")
...
```

NOTE: Some of the demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

7.11.3.17 How to run multiple GamsJobs in parallel using a GamsCheckpoint (transport6.py)

With the exception of GamsWorkspace the objects in the `gams.control` package cannot be accessed across different threads unless the instance is locked. The classes themselves are thread safe and multiple objects of the class can be used from different threads (see below for restrictions on solvers that are not thread safe within the `GamsModelInstance` class).

Note

> If you use multiple instances of the `GamsWorkspace` in parallel, you should avoid using the same working directory. Otherwise you may end up with conflicting file names.

Currently only Cplex, Gurobi, and SoPlex fully utilize the power of solving GamsModelInstances. Some solvers will not even work in a multi-threaded application using GamsModelInstances. For some solvers this is unavoidable because the solver library is not thread safe (e.g. MINOS), other solvers are in principle thread safe but the GAMS link is not (e.g. SNOPT). Moreover, GamsModelInstances are not available for quadratic model types (QCP, MIQCP, RMIQCP).

This example illustrates how to run the jobs known from transport5.py in parallel. We initialize the GamsCheckpoint `cp` and introduce a demand multiplier as we did before:

```
...
    cp = ws.add_checkpoint()
    job = ws.add_job_from_string(GAMS_MODEL)
    job.run(checkpoint=cp)
    bmult = [0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3]
...
```

Furthermore, we introduce a lock object `io_lock` that will be used to avoid mixed up output from the parallel jobs. We create one scenario for each entry of `bmultlist` and cause a thread to begin execution.

```
...
    io_lock = Lock()
    threads = {}
    for b in bmult:
        threads[b] = Thread(target=run_scenario, args=(ws, cp, io_lock, b))
        threads[b].start()
    for b in bmult:
        threads[b].join()
...
```

In function `run_scenario` a GamsJob is created and run just like in the preceding example of transport5.py. The output section is also the same except for the fact that it is 'locked' by the object `io_lock` which means that the output section cannot be executed simultaneously for multiple demand multipliers.

```
...
def run_scenario(workspace, checkpoint, io_lock, b):
```

```
    job = workspace.add_job_from_string(
        f"bmult={b}; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;",
        checkpoint,
    )
    job.run()
    # we need to make the ouput a critical section to avoid messed up report informations
    io_lock.acquire()
    print(f"Scenario bmult={b}:")
    print(f"  Modelstatus: {job.out_db['ms'].first_record().value}")
    print(f"  Solvestatus: {job.out_db['ss'].first_record().value}")
    print(f"  Obj: {job.out_db['z'].first_record().level}")
    io_lock.release()
...
```

While the output in transport5.py is strictly ordered subject to the order of the elements of `bmult` in transport6.py the output blocks might change their order but the blocks describing one scenario are still appearing together due to the `io_lock` object.

7.11.3.18 How to create a GamsModelInstance from a GamsCheckpoint (transport7.py)

In transport7.py the usage of GamsModelInstance is demonstrated.

At first checkpoint `cp` is created as in the preceding examples. Note that the GamsJob `job` again contains no solve statement and the demand multiplier is already included with default value 1. We create the GamsModelInstance `mi` using the GamsCheckpoint.add_modelinstance method.

```
...
    cp = ws.add_checkpoint()
    job = ws.add_job_from_string(GAMS_MODEL)
    job.run(checkpoint=cp)
    mi = cp.add_modelinstance()
...
```

7.11.3.19 How to modify a parameter of a GamsModelInstance using GamsModifier (transport7.py)

A GamsModelInstance uses a sync_db to maintain the data. We define `bmult` as GamsParameter using the GamsDatabase.add_parameter method and specify gurobi as solver. Afterwards the GamsModelInstance is instantiated with 3 arguments, the solve statement, GamsModifier `bmult` and GamsOptions `opt`. The GamsModifier means that `bmult` is modifiable while all other parameters, variables and equations of GamsModelInstance `mi` stay unchanged. We use the GamsParameter.add_record method to assign a value to `bmult`. That value can be varied afterwards using the GamsParameter.first_record method to reproduce our well-known example with different demand multipliers.

```
...
    bmult = mi.sync_db.add_parameter("bmult", 0, "demand multiplier")
    opt = ws.add_options()
    opt.all_model_types = "cplex"
    mi.instantiate("transport use lp min z", GamsModifier(bmult), opt)
    bmult.add_record().value = 1.0
    bmult_list = [0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3]
    for b in bmult_list:
        bmult.first_record().value = b
        mi.solve()
        print(f"Scenario bmult={b}:")
        print(f"  Modelstatus: {mi.model_status}")
        print(f"  Solvestatus: {mi.solver_status}")
        print(f"  Obj: {mi.sync_db['z'].first_record().level}")
...
```

7.11.3.20 How to modify a variable of a GamsModelInstance using GamsModifier (transport7.py)

We create a GamsModelInstance using the GamsCheckpoint.add_modelinstance method. Afterwards we define `x` as GamsVariable and a GamsParameter `xup` that will be used as upper bound for `x`. At the following instantiate method GamsModifier has 3 arguments. The first one says that `x` is modifieable, the

second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the GamsParameter that holds the new value, in this case `xup`.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```
...
    mi = cp.add_modelinstance()
    x = mi.sync_db.add_variable("x", 2, VarType.Positive)
    xup = mi.sync_db.add_parameter("xup", 2, "upper bound on x")
    # instantiate the GamsModelInstance and pass a model definition and GamsModifier to declare upper bound of x mutable
    mi.instantiate("transport use lp min z", GamsModifier(x, UpdateAction.Upper, xup))
    mi.solve()
    for i in job.out_db["i"]:
        for j in job.out_db["j"]:
            xup.clear()
            xup.add_record((i.key(0), j.key(0))).value = 0
            mi.solve()
            print(f"Scenario link blocked: {i.key(0)} - {j.key(0)}")
            print(f"  Modelstatus: {mi.model_status}")
            print(f"  Solvestatus: {mi.solver_status}")
            print(f"  Obj: {mi.sync_db['z'].find_record().level}")
...
```

7.11.3.21 How to use a queue to solve multiple GamsModelInstances in parallel (transport8.py)

We initialize a GamsCheckpoint `cp` from a GamsJob. Then we define a list that represents the different values of the demand multiplier. That list will be used like a queue where we extract the last element first. The objects `list_lock` and `io_lock` are used later to avoid multiple reading of one demand multiplier and messed up output. Then we call function `scen_solve` multiple times in parallel. The number of parallel calls is specified by `nr_workers`.

```
...
    cp = ws.add_checkpoint()
    job = ws.add_job_from_string(GAMS_MODEL)
    job.run(checkpoint=cp)
    bmult_list = [1.3, 1.2, 1.1, 1.0, 0.9, 0.8, 0.7, 0.6]
    list_lock = Lock()
    io_lock = Lock()
    # start 2 threads
    nr_workers = 2
    threads = {}
    for i in range(nr_workers):
        threads[i] = Thread(
            target=scen_solve, args=(cp, bmult_list, list_lock, io_lock)
        )
        threads[i].start()
    for i in range(nr_workers):
        threads[i].join()
```

In function `scen_solve` we create and instantiate a GamsModelInstance as in the preceding examples and make parameter `bmult` modifiable. Note that we choose `cplex` as solver because it is thread safe (gurobi would also be possible).

We have two critical sections that are locked by the objects `list_lock` and `io_lock`. Note that the pop method removes and returns the last element from the list and deletes it. Once the list is empty the loop terminates.

```
...
def scen_solve(checkpoint, bmult_list, list_lock, io_lock):
    list_lock.acquire()
    mi = checkpoint.add_modelinstance()
    list_lock.release()
    bmult = mi.sync_db.add_parameter("bmult", 0, "demand multiplier")
    opt = ws.add_options()
    opt.all_model_types = "cplex"
    # instantiate the GamsModelInstance and pass a model definition and GamsModifier to declare bmult mutable
    mi.instantiate("transport use lp min z", GamsModifier(bmult), opt)
    bmult.add_record().value = 1.0
    while True:
        # dynamically get a bmult value from the queue instead of passing it to the different threads at creation time
        list_lock.acquire()
        if not bmult_list:
            list_lock.release()
            return
```

```
        b = bmult_list.pop()
        list_lock.release()
        bmult.first_record().value = b
        mi.solve()
        # we need to make the ouput a critical section to avoid messed up report informations
        io_lock.acquire()
        print(f"Scenario bmult={b}:")
        print(f"  Modelstatus: {mi.model_status}")
        print(f"  Solvestatus: {mi.solver_status}")
        print(f"  Obj: {mi.sync_db['z'].first_record().level}")
        io_lock.release()
...
```

7.11.3.22 How to fill a GamsDatabase by reading from MS Access (transport9.py)

This example illustrates how to import data from Microsoft Access to a GamsDatabase. There are a few prerequisites required to run transport9.py successfully.

- We import `pyodbc`.
- Note that an architecture mismatch might cause problems. The bitness of your MS Access, Python, pyodbc and GAMS should be identical (64 bit).

We call a function `read_data_from_access` that finally returns a GamsDatabase as shown below.

```
...
    db = read_from_access(ws)
...
```

The function `read_from_access` begins with the creation of an empty database. Afterwards we set up a connection to the MS Access database `transport.accdb` which can be found in `[GAMSDIR]\apifiles\Data`. To finally read in GAMS sets and parameters we call the functions `read_set` and `read_parameter` that are explained down below.

```
...
def read_from_access(ws):
    db = ws.add_database()
    # connect to database
    str_access_conn = r"DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};DBQ=..\..\..\..\apifiles\Data\transport.accdb"
    try:
        connection = pyodbc.connect(str_access_conn)
    except Exception as e:
        raise Exception(f"Error: Failed to create a database connection.\n{e}")
    # read GAMS sets
    read_set(connection, db, "SELECT Plant FROM plant", "i", 1, "canning plants")
    read_set(connection, db, "SELECT Market FROM Market", "j", 1, "markets")
    # read GAMS parameters
    read_parameter(
        connection,
        db,
        "SELECT Plant,Capacity FROM Plant",
        "a",
        1,
        "capacity of plant i in cases",
    )
    read_parameter(
        connection,
        db,
        "SELECT Market,Demand FROM Market",
        "b",
        1,
        "demand at market j in cases",
    )
    read_parameter(
        connection,
        db,
        "SELECT Plant,Market,Distance FROM Distance",
        "d",
        2,
        "distance in thousands of miles",
    )
    connection.close()
    return db
...
```

The function `read_set` adds a set to the GamsDatabase that is filled with the data from the MS Access file afterwards. The function `read_parameter` works quite similar.

```
...
def read_set(connection, db, query_string, set_name, set_dim, set_exp=""):
    try:
        cursor = connection.cursor()
        cursor.execute(query_string)
        data = cursor.fetchall()
        if len(data[0]) != set_dim:
            raise Exception(
                "Number of fields in select statement does not match setDim"
            )
        i = db.add_set(set_name, set_dim, set_exp)
        for row in data:
            keys = []
            for key in row:
                keys.append(str(key))
            i.add_record(keys)
    except Exception as ex:
        raise Exception(
            "Error: Failed to retrieve the required data from the database.\n{0}".format(
                ex
            )
        )
    finally:
        cursor.close()
...
```

Once we read in all the data we can create a GamsJob from the GamsDatabase and run it as usual.

7.11.3.23 How to fill a GamsDatabase by reading from MS Excel (transport10.py)

This example illustrates how to read data from Excel, or to be more specific, from `[GAMSDIR]\apifiles\Data\transport.xl`

At first you have to download the `openpyxl` package:

```
pip install openpyxl
```

Now you should be able to run transport10.py.

In transport10.py the model is given as string without data like in many examples before and the Excel file `transport.xlsx` is located at `[GAMSDIR]\apifiles\Data`. At first we define the workbook to read from and the different sheet names. To ensure to have the same number of markets and plants in all spreadsheets, we conduct a little test that checks for the number of rows and columns. Our workspace is only created if this test yields no errors.

```
...
    wb = load_workbook(
        os.path.join(*[os.pardir] * 4, "apifiles", "Data", "transport.xlsx")
    )
    capacity = wb["capacity"]
    demand = wb["demand"]
    distance = wb["distance"]
    # number of markets/plants have to be the same in all spreadsheets
    if (
        distance.max_column - 1 != demand.max_column
        or distance.max_row - 1 != capacity.max_row
    ):
        raise Exception("Size of the spreadsheets doesn't match")
...
```

Now we can create a GamsDatabase and read in the data contained in the different worksheets. We iterate over the columns and read in the set names and the corresponding parameter values.

```
...
    db = ws.add_database()
    i = db.add_set("i", 1, "Plants")
    j = db.add_set("j", 1, "Markets")
    capacity_param = db.add_parameter_dc("a", [i], "Capacity")
    demand_param = db.add_parameter_dc("b", [j], "Demand")
    distance_param = db.add_parameter_dc("d", [i, j], "Distance")
    for c in capacity.iter_cols():
        key = c[0].value
        i.add_record(key)
        capacity_param.add_record(key).value = c[1].value
    for c in demand.iter_cols():
        key = c[0].value
        j.add_record(key)
        demand_param.add_record(key).value = c[1].value
```

```
    for c in range(2, distance.max_column + 1):
        for r in range(2, distance.max_row + 1):
            keys = (
                distance.cell(row=r, column=1).value,
                distance.cell(row=1, column=c).value,
            )
            v = distance.cell(row=r, column=c).value
            distance_param.add_record(keys).value = v
...
```

Note that we can name sets and parameters just like in the database but we don't have to. Now we can run our GamsJob as usual.

```
...
    job = ws.add_job_from_string(GAMS_MODEL)
    opt = ws.add_options()
    opt.defines["gdxincname"] = db.name
    opt.all_model_types = "xpress"
    job.run(opt, databases=db)
    for rec in job.out_db["x"]:
        print(
            f"x({rec.key(0)},{rec.key(1)}): level={rec.level} marginal={rec.marginal}"
        )
...
```

7.11.3.24 How to create and use a save/restart file (transport11.py)

In transport11.py we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a GamsCheckpoint.

We create a directory `tmp` with internal identifier `w_dir` in the directory we are currently in. This file will be used as working directory later. From the main function we call the function `create_save_restart` giving it directory `tmp` and the desired name for the save/restart file (`tbase`) as arguments.

```
...
    working_dir = os.path.join(os.curdir, "tmp")
    create_save_restart(sys_dir, os.path.join(working_dir, "tbase"))
...
```

In function `create_save_restart` we create a workspace with the given working directory (`w_dir` refers to `tmp`). Then we create a GamsJob from a string. Note that the string given via `get_base_model_text` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a GamsOption to only compile the job but do not execute it. Then we create a checkpoint `cp` that is initialized by the following run of the GamsJob and stored in the file given as argument to the function, in our case `tbase`. This becomes possible because the add_checkpoint method accepts identifiers as well as file names as argument.

```
...
def create_save_restart(sys_dir, cp_file_name):
    ws = GamsWorkspace(os.path.dirname(cp_file_name), sys_dir)
    job_1 = ws.add_job_from_string(GAMS_BASE_MODEL)
    opt = ws.add_options()
    opt.action = Action.CompileOnly
    cp = ws.add_checkpoint(os.path.basename(cp_file_name))
    job_1.run(opt, cp)
...
```

So what you should keep in mind before we return to further explanations of the main function is, that the file `tbase` is now in the current working directory and contains a checkpoint that will work exactly like a restart file.

In the main function we define some data using Python data structures as we already did in [transport4.py] (How to define data using Python data structures (transport4.py)) before we create the GamsWorkspace and a GamsDatabase.

```
...
    sys_dir = sys.argv[1] if len(sys.argv) > 1 else None
    working_dir = os.path.join(os.curdir, "tmp")
    ws = GamsWorkspace(working_dir, sys_dir)
    db = ws.add_database()
...
```

Afterwards we set up the GamsDatabase like we already did in [transport4.py] (How to prepare a GamsDatabase from Pyth
Once this is done we run a GamsJob using this data plus the checkpoint stored in file `tbase`.

```
...
    cp_base = ws.add_checkpoint("tbase")
    job = ws.add_job_from_string(GAMS_MODEL, cp_base)
    opt = ws.add_options()
    opt.defines["gdxincname"] = db.name
    opt.all_model_types = "xpress"
    job.run(opt, databases=db)
...
```

Note that the string from which we create `job` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the GamsDatabase correctly. The entire model definition is delivered by the checkpoint `cp_base` which is equal to the one we saved in `tbase`.

7.12 Tutorial

The goal of this tutorial is to provide a compact overview of the basic functionality of the GAMS Matlab Control API. It allows the user to start immediately working with the API by providing a set of examples based on the well-known **transportation problem**. Those examples are also part of the GAMS system directory, see `[PathToGAMS]/api/matlab/examples/control`. These examples introduce several API features step by step.

We recommend to open the aforementioned files to gain a complete overview of the examples. Down below we explain the examples with the help of selected code snippets.

- Choose the GAMS system
- Export data to GDX
- Import data from GDX
- Run a Job from file
- Retrieve a solution from an output database
- Specify solver using Options
- Run Job with solver option file and capture log
- Use include files
- Set non-default working directory
- Read data from string and export to GDX
- Run Job using data from GDX
- Run Job using implicit database communication
- Define data using Matlab data structures
- Prepare Database from Matlab data structures
- Initialize Checkpoint by running Job
- Initialize Job from Checkpoint
- Create ModelInstance from Checkpoint
- Modify parameter of ModelInstance using Modifier
- Modify variable of ModelInstance using Modifier
- Create and use save/restart file

7.12.1 Choose the GAMS system

Example: `transport1`

By default the GAMS system is determined automatically. In case of having multiple GAMS systems on your machine, the desired system can be specified via an additional argument when the workspace is created. When running the examples, we can provide an additional command line argument in order to define the GAMS system directory that should be used. By executing `transport1` with C:/GAMS/46 we use GAMS 46.4 to run `transport1` even if our default GAMS system might be a different one. This is managed by the following code:

```
wsInfo = gams.control.WorkspaceInfo();
if nargin > 0
  wsInfo.systemDirectory = varargin{1};
end
ws = gams.control.Workspace(wsInfo);
```

Note

The API can detect GAMS automatically from the `PATH` environment variable. Please note that this is not the `MATLABPATH`. You can inspect the `PATH` with `getenv("PATH")`.

In Matlab you can import the GAMS Control package by `import gams.control.*`. Then, you don't need to call the GAMS classes with the preceding `gams.control.`.

7.12.2 Export data to GDX

Example: `transport_gdx`

Although the Matlab Control API offers much more than exchanging data between Matlab and GDX, a common use case is the export and import of GDX files. The central class for this purpose is **Database**. We assume that the data to be exported is available in Matlab data structures.

```
plants = {'Seattle', 'San-Diego'};
markets = {'New-York', 'Chicago', 'Topeka'};
capacity = containers.Map();
capacity('Seattle') = 350;
capacity('San-Diego') = 600;
demand = containers.Map();
demand('New-York') = 325;
demand('Chicago') = 300;
demand('Topeka') = 275;
distance = containers.Map();
distance('Seattle.New-York') = 2.5;
distance('Seattle.Chicago') = 1.7;
distance('Seattle.Topeka') = 1.8;
distance('San-Diego.New-York') = 2.5;
distance('San-Diego.Chicago') = 1.8;
distance('San-Diego.Topeka') = 1.4;
```

Different type of GAMS symbols are represented using different Matlab data structures. The data for the GAMS sets is represented using a cell of strings (e.g. `plants` and `markets`). On the other hand, GAMS parameters are represented by a `containers.Map` (e.g. `capacity` and `demand`). Note that the representation of the two dimensional parameter `distance` uses a dot notation for storing the keys. The choice of data structures can also be different, but the used structures in this example fit well for representing GAMS data with Matlab data structures.

A new **Database** instance can be created using **Workspace.addDatabase**.

```
db = ws.addDatabase();
```

We start adding GAMS sets using the method **Database.addSet** which takes the name and the dimension as arguments. The third argument is an optional explanatory text. A for-loop iterates through `plants` and adds new records to the recently created **Set** instance `i` using **Set.addRecord**.

```
i = db.addSet('i', 1, 'canning plants');
for p = plants
    i.addRecord(p{1});
end
```

Parameter instances can be added by using the method **Database.addParameter**. In this example we use the overloaded method which takes a list of **Set** instances instead of the dimension for creating a parameter with domain information.

```
a = db.addParameter('a', 'capacity of plant i in cases', i);
for p = plants
    rec = a.addRecord(p{1});
    rec.value = capacity(p{1});
end
```

As soon as all data is prepared in the **Database**, the method **Database.export** can be used to create a GDX file.

```
db.export('data.gdx');
```

7.12.3 Import data from GDX

Example: `transport_gdx`

Data can be imported from a GDX file using **Workspace.addDatabaseFromGDX**. The method takes a path to a GDX file and creates a **Database** instance.

```
gdxdb = ws.addDatabaseFromGDX('data.gdx');
```

Reading the data from the **Set** `i` into a cell of strings can be done as follows:

```
gdxPlantsRecords = gdxdb.getSet('i').records;
gdxPlants = cell(size(gdxPlantsRecords));
for i = 1:numel(gdxPlants)
  gdxPlants{i} = gdxPlantsRecords{i}.key(1);
end
```

`i` is retrieved by calling **Database.getSet** on `gdxdb`. The returned **Set** object has an attribute **records** with an cell array of **SetRecords**. Each record can be asked for its keys.

You can do the same for **Parameter**. Instead of creating a cell, we want to have the data in the form of a `containers.Map`. **ParameterRecord** can not only be asked for its keys, but also for its value. The following code snippet shows how to read the one dimensional parameter `a` into a map.

```
gdxCapacity = containers.Map();
for rec = gdxdb.getParameter('a').records
  gdxCapacity(rec{1}.key(1)) = rec{1}.value;
end
```

For a key of multi dimensional symbol, we choose a dot based concatenation of keys.

```
gdxDistance = containers.Map();
for rec = gdxdb.getParameter('d').records
  gdxCapacity([rec{1}.key(1), '.', rec{1}.key(2)]) = rec{1}.value;
end
```

Scalar can be read into a variable of type double by accessing the value of the first and only record.

```
gdxFreight = gdxdb.getParameter('f').record.value;
```

7.12.4 Run a Job from file

Example: `transport1`

At first we create our workspace using **Workspace** `ws = gams.control.Workspace();`. Afterwards, we can create a **Job** `t1` using the **Workspace.addJobFromGamsLib** method and run it.

Apparently you can create a **Job** with any other gms file you might have created on your own as long as it is located in the current working directory. Then the **Job** `t1` can be defined using the **Workspace.addJobFromFile** method.

```
% create Workspace "ws" with default working directory
ws = gams.control.Workspace();
% create Job "t1" from "trnsport" model in GAMS Model Libraries
t1 = ws.addJobFromGamsLib('trnsport');
% run Job "t1"
t1.run();
```

7.12.5 Retrieve a solution from an output database

Example: `transport1`

The following lines create the solution output and illustrate the usage of the **Job.outDB** property to get access to the **Database** created by the **Job.run** method. To retrieve the content of variable `x` we use the **Database.getVariable** method and the **VariableRecord** class.

```
% retrieve Variable "x" from Job's output databases
fprintf('Ran with Default:\n');
for x = t1.outDB.getVariable('x').records
    fprintf('x(%s,%s): level=%g marginal=%g\n', x{1}.keys{:}, x{1}.level, x{1}.marginal);
end
```

7.12.6 Specify solver using Options

Example: `transport1`

The solver can be specified via the **Options** class and the **Workspace.addOptions** method. The **Options.setAllModelTypes** property sets `xpress` as default solver for all model types which the solver can handle. Then we run our **Job** `t1` with the new **Options**.

```
% create Options 'opt1'
opt1 = ws.addOptions();
% set all model types of 'opt1' for 'xpress'
opt1.setAllModelTypes('xpress');
% run Job 't1' with Options 'opt1'
t1.run(opt1);
```

7.12.7 Run Job with solver option file and capture log

Example: `transport1`

At first we create the file `xpress.opt` with content `algorithm=barrier` which will be used as solver option file and is stored in the current working directory. Afterward we use **Options** just like in the preceding example and **Options.optFile** property to 1 to tell the solver to look for a solver option file. We specify the argument `output` in order to stream the log of the **Job** into the file `transport1_xpress.log`. When the `output` argument is omitted then the log will be written to standard output.

```
% write file 'xpress.opt' under Workspace's working directory
fid = fopen(fullfile(ws.workingDirectory, 'xpress.opt'), 'w');
fprintf(fid, 'algorithm=barrier');
fclose(fid);
% create Options 'opt2'
opt2 = ws.addOptions();
% set all model types of 'opt2' for 'xpress'
opt2.setAllModelTypes('xpress');
% for 'opt2', use 'xpress.opt' as solver's option file
opt2.optFile = 1;
% run Job 't2' with Options 'opt2' and capture log into 'transport1_xpress.log'.
output = gams.control.PrintStream(fullfile(ws.workingDirectory, 'transport1_xpress.log'));
t1.run(opt2, output);
```

7.12.8 Use include files

Example: `transport2`

In this example, as in many succeeding, the data text and the model text are separated into two different strings. Note that these strings `data` and `model` are using GAMS syntax.

At first we write an include file `tdata.gms` that contains the data but not the model text:

```
% write 'data' into file 'tdata.gms' under Workspace's working directory
fid = fopen(fullfile(ws.workingDirectory, 'tdata.gms'), 'w');
fprintf(fid, data);
fclose(fid);
```

Afterwards we create a **Job** using the **Workspace.addJobFromString** method. **Options.defines** is used like the the 'double dash' GAMS parameters, i.e. it corresponds to `--incname=tdata` on the command line where incname is used as name for the include file in the `model` string.

```
% create Job 't2' from the 'model' string variable
t2 = ws.addJobFromString(model);
% create Options 'opt' and define 'incname' as 'tdata'
opt = ws.addOptions();
opt.defines('incname', 'tdata');
% run Job 't2' with Options 'opt'
t2.run(opt);
```

The string model contains the following lines to read in the data.

```
$if not set incname $abort 'no include file name for data file provided'
$include %incname%
```

7.12.9 Set non-default working directory

Example: `transport3`

At first we create a new directory. Once this is done we can use this directory when creating the **Workspace** and make it the working directory.

```
% create a directory
workingDirectory = fullfile(pwd, 'transport3');
mkdir(workingDirectory);
% create a workspace
wsInfo = gams.control.WorkspaceInfo();
wsInfo.workingDirectory = workingDirectory;
ws = gams.control.Workspace(wsInfo);
```

7.12.10 Read data from string and export to GDX

Example: `transport3`

We read the data from the string data. Note that this contains no model but only data definition in GAMS syntax. By running the corresponding **Job** a **Database** is created that is available via the **Job.outDB** property. We can use the **Database.export** method to write the content of this database to a gdx file `tdata.gdx`.

```
% create and run a job from a data file, then explicitly export to a GDX file
t3 = ws.addJobFromString(data);
t3.run();
t3.outDB.export(fullfile(ws.workingDirectory, 'tdata.gdx'));
```

7.12.11 Run Job using data from GDX

Example: `transport3`

This works quite similar to the usage of an include file explained in `transport2` - Use include files .

```
% run a job using an instance of Options that defines the data include file
t3 = ws.addJobFromString(model);
opt = ws.addOptions();
opt.defines('gdxincname', 'tdata');
t3.run(opt);
```

Note that there are some minor changes in the model due to the usage of a gdx instead of an include file.

```
$if not set gdxincname $abort 'no include file name for data file provided'
$gdxin %gdxincname%
$load i j a b d f
$gdxin
```

7.12.12 Run Job using implicit database communication

Example: `transport3`

This example does basically the same as the two preceding examples together. We create two **Jobs** `t3a` and `t3b` where the first one contains only the data and the second one contains only the model without data. After running `t3a` the corresponding **Job.outDB** can be read in directly just like a gdx file. Note that the database needs to be passed to the **Job.run** method as additional argument.

```
t3a = ws.addJobFromString(data);
t3b = ws.addJobFromString(model);
opt = ws.addOptions();
t3a.run();
opt.defines('gdxincname', t3a.outDB.name);
t3b.run(opt, t3a.outDB);
```

7.12.13 Define data using Matlab data structures

Example: `transport4`

We use cell arrays and `containers.Map` to define Matlab data structures that correspond to the sets, parameters and tables used for the data definition in GAMS.

```
plants = {'Seattle', 'San-Diego'};
markets = {'New-York', 'Chicago', 'Topeka'};
capacity = containers.Map();
capacity('Seattle') = 350;
capacity('San-Diego') = 600;
demand = containers.Map();
demand('New-York') = 325;
demand('Chicago') = 300;
demand('Topeka') = 275;
distance = containers.Map();
distance('Seattle.New-York') = 2.5;
distance('Seattle.Chicago') = 1.7;
distance('Seattle.Topeka') = 1.8;
distance('San-Diego.New-York') = 2.5;
distance('San-Diego.Chicago') = 1.8;
distance('San-Diego.Topeka') = 1.4;
```

7.12.14 Prepare Database from Matlab data structures

Example: `transport4`

At first we create an empty **Database** `db` using the **Workspace.addDatabase** method. Afterwards we prepare the database. To add a set to the database we use the **Set** class and the **Database.addSet** method with arguments describing the identifier, dimension and explanatory text. To add the records to the database we iterate over the elements of our Matlab data structure and add them by using the **Set.addRecord** method.

For parameters the procedure is pretty much the same. Note that the table that specifies the distances in GAMS can be treated as parameter with dimension 2 and that scalars can be treated as parameter with dimension 0.

The **Job** can be run like explained in the preceding example about implicit database communication.

```
db = ws.addDatabase();
i = db.addSet('i', 1, 'canning plants');
for p = plants
  i.addRecord(p{1});
end
j = db.addSet('j', 1, 'markets');
for m = markets
  j.addRecord(m{1});
end
a = db.addParameter('a', 'capacity of plant i in cases', i);
for p = plants
  rec = a.addRecord(p{1});
  rec.value = capacity(p{1});
```

```
end
b = db.addParameter('b', 'demand at market j in cases', j);
for m = markets
  rec = b.addRecord(m{1});
  rec.value = demand(m{1});
end
d = db.addParameter('d', 'distance in thousands of miles', i, j);
for p = plants
  for m = markets
    rec = d.addRecord(p{1}, m{1});
    rec.value = distance([p{1}, '.', m{1}]);
  end
end
f = db.addParameter('f', 'freight in dollars per case per thousand miles');
rec = f.addRecord();
rec.value = 90;
% create and run a job from the model and read gdx include file from the database
t4 = ws.addJobFromString(model);
opt = ws.addOptions();
opt.defines('gdxincname', db.name);
t4.run(opt, db);
```

7.12.15 Initialize Checkpoint by running Job

Example: `transport5`

The following lines of code conduct several operations. While the first line simply creates a **Checkpoint**, the second one uses the **Workspace.addJobFromString** method to create a **Job** containing the model text and data but no solve statement. Afterwards the run method gets the **Checkpoint** as argument. That means the **Checkpoint** `cp` captures the state of the **Job**.

```
cp = ws.addCheckpoint();
t5 = ws.addJobFromString(model);
t5.run(cp);
```

7.12.16 Initialize Job from Checkpoint

Example: `transport5`

Note that the string returned from function `model` contains the entire model and data definition plus an additional demand multiplier and scalars for model and solve status but no solve statement:

```
Scalar bmult demand multiplier /1/;
...
demand(j) .. sum(i, x(i,j)) =g= bmult*b(j) ;
...
Scalar ms 'model status', ss 'solve status';
```

In `transport5` we create a list with eight different values for this demand multiplier.

```
bmultlist = [0.6, 0.7, 0.8, 0.9, 1.0, 1.1, 1.2, 1.3];
```

For each entry of that list we create a **Job** `t5` using the **Workspace.addJobFromString** method. Besides another string which resets the demand multiplier `bmult`, specifies the solve statement and assigns values to the scalars `ms` and `ss` we pass the checkpoint cp as additional argument. This results in a **Job** combined from the checkpoint plus the content provided by the string. We run the **Job** and echo some interesting data from the **Job.outDB** using the **Database.getParameter** and **Database.getVariable** methods and the **Symbol.record** attribute plus the **ParameterRecord.value** and the **VariableRecord.level** properties.

```
for i = 1:numel(bmultlist)
  job = sprintf('bmult=%f; solve transport min z use lp; ms=transport.modelstat; ss=transport.solvestat;', bmultlist(i));
  t5 = ws.addJobFromString(job, cp);
  t5.run();
  fprintf('Scenario bmult=%f:\n', bmultlist(i));
  fprintf('  Modelstatus: %s\n', gams.control.globals.ModelStat(t5.outDB.getParameter('ms').record.value).select);
  fprintf('  Solvestatus: %s\n', gams.control.globals.SolveStat(t5.outDB.getParameter('ss').record.value).select);
  fprintf('  Obj: %f\n', t5.outDB.getVariable('z').record.level);
end
```

Note

> Some of the demand multipliers cause infeasibility. Nevertheless, GAMS keeps the incumbent objective function value. Therefore the model status and the solve status provide important information for a correct solution interpretation.

7.12.17 Create ModelInstance from Checkpoint

Example: transport7

In transport7 the usage of **matlab::gams::control::ModelInstance** is demonstrated.

At first **Checkpoint** cp is created as in the preceding examples. Then we create the **ModelInstance** mi using the **Checkpoint.addModelInstance** method. Note that the **Job** again contains no solve statement and the demand multiplier is already included with default value 1.

```
cp = ws.addCheckpoint();
% initialize a checkpoint by running a job
t7 = ws.addJobFromString(model);
t7.run(cp);
% create a ModelInstance and solve it multiple times with different scalar bmult
mi = cp.addModelInstance();
```

7.12.18 Modify parameter of ModelInstance using Modifier

Example: transport7

A **ModelInstance** uses a **matlab::gams::control::ModelInstance::syncDB** ModelInstance.syncDB to maintain the data. We define bmult as **Parameter** using the **Parameter** method and specify cplex as solver. Afterwards the **ModelInstance** is instantiated with 3 arguments, the solve statement, **Options** opt and **Modifier** bmult. The **Modifier** means that bmult is modifiable while all other parameters, variables and equations of **ModelInstance** mi stay unchanged. We use the **Parameter.addRecord** method and the **ParameterRecord.value** property to assign a value to bmult. That value can be varied afterwards using the **Symbol.record** property to reproduce our well-known example with different demand multipliers.

```
bmult = mi.syncDB.addParameter('bmult', 'demand multiplier');
opt = ws.addOptions();
opt.setAllModelTypes('cplex');
% instantiate the ModelInstance and pass a model definition and Modifier to declare bmult mutable
mi.instantiate('transport use lp min z', opt, gams.control.Modifier(bmult));
rec = bmult.addRecord();
rec.value = 1.0;
bmultlist = [0.6, 0.7 , 0.8, 0.9, 1.0, 1.1, 1.2, 1.3];
for i = 1:numel(bmultlist)
  rec.value = bmultlist(i);
  mi.solve();
  fprintf('Scenario bmult=%f:\n', bmultlist(i));
  fprintf('  Modelstatus: %s\n', mi.modelStatus);
  fprintf('  Solvestatus: %s\n', mi.solveStatus);
  fprintf('  Obj: %f\n', mi.syncDB.getVariable('z').record.level);
end
```

7.12.19 Modify variable of ModelInstance using Modifier

Example: transport7

We create a **ModelInstance** just like in the next to last example. We define x as **Variable** and its upper bound as **Parameter** xup. At the following **ModelInstance.instantiate** method **Modifier** has 3 arguments. The first one says that x is modifiable, the second determines which part of the variable (lower bound, upper bound or level) can be modified and the third specifies the **Parameter** that holds the new value.

In the following loops we set the upper bound of one link of the network to zero, which means that no transportation between the corresponding plant and market is possible, and solve the modified transportation problem.

```
mi = cp.addModelInstance();
x = mi.syncDB.addVariable('x', 2, gams.control.globals.VarType.POSITIVE, '');
xup = mi.syncDB.addParameter('xup', 2, 'upper bound on x');
% instantiate the ModelInstance and pass a model definition and Modifier to declare upper bound of X mutable
mi.instantiate('transport use lp min z', gams.control.Modifier(x, gams.control.globals.UpdateAction.UPPER, xup));
for i = t7.outDB.getSet('i').records
```

```
  for j = t7.outDB.getSet('j').records
    xup.clear();
    keys = {i{1}.key(1), j{1}.key(1)};
    rec = xup.addRecord(keys);
    rec.value = 0;
    mi.solve();
    fprintf('Scenario link blocked: %s - %s\n', keys{:});
    fprintf('  Modelstatus: %s\n', mi.modelStatus);
    fprintf('  Solvestatus: %s\n', mi.solveStatus);
    fprintf('  Obj: %f\n', mi.syncDB.getVariable('z').record.level);
  end
end
```

7.12.20 Create and use save/restart file

Example: `transport11`

In `transport11` we demonstrate how to create and use a save/restart file. Usually such a file should be supplied by an application provider but in this example we create one for demonstration purpose. Note that the restart is launched from a **Checkpoint**. From the main function we call the function `create_save_restart` giving it the current workspace settings and the desired file name as arguments.

```
create_save_restart(ws.workingDirectory, ws.systemDirectory, ws.debugLevel, 'tbase');
```

In function `create_save_restart` we create a workspace with the given workspace settings. Then we create a **Job** from a string. Note that the string given via `basemodel` contains the basic definitions of sets without giving them a content (that is what `$onempty` is used for). Afterwards we specify a **Options** to only compile the job but do not execute it. Then we create a **Checkpoint** `cp` that is initialized by the following run of the **Job** and stored in the file given as argument to the function, in our case `tbase`. This becomes possible because the **Workspace.addCheckpoint** method accepts identifiers as well as file names as argument.

```
ws = gams.control.Workspace(workdir, systemdir, debuglevel);
j1 = ws.addJobFromString(basemodel);
opt = ws.addOptions();
opt.action = gams.control.options.Action.CompileOnly;
cp = ws.addCheckpoint('tbase');
j1.run(opt, cp);
opt.dispose();
j1.outDB.dispose();
```

So what you should keep in mind before we return to further explanations of the main function is, that the file tbase is now in the current working directory and contains a checkpoint. Now in the main function we define some data using Matlab data structures as we already did in `transport4` before we create the **Workspace** and a **Database**.

```
ws = gams.control.Workspace(wsInfo);
db = ws.addDatabase();
```

Afterwards we set up the **Database** like we already did in `transport4`. Once this is done we run a **Job** using this data plus the checkpoint stored in file tbase.

```
cpBase = ws.addCheckpoint('tbase');
opt = ws.addOptions();
t11 = ws.addJobFromString(model, cpBase);
opt.defines('gdxincname', db.name);
opt.setAllModelTypes('xpress');
t11.run(opt, db);
```

Note that the string from which we create **Job** `t11` is different to the one used to prepare the checkpoint stored in `tbase` and is only responsible for reading in the data from the **Database** correctly. The entire model definition is delivered by the **Checkpoint** `cpBase` which is equal to the one we saved in `tbase`.

7.13 Release Notes

7.13.1 44.1.0 (July 2023)

- Added method `runEngine` to `GAMSJob` class to run jobs on GAMS Engine.

- since GAMS 44.1.0
  - all classes in `GAMSJavaAPI.jar` require Java SE 11 or later to run. `GAMSJavaAPI.jar` has additional dependency on `json-simple-1.1.1.jar`. `GAMSJavaAPI.jar` and `json-simple-1.1.1.jar` are required to be in the same directory to run.
  - all classes that work with Java SE 8 are in `GAMSJavaAPI-8.jar` but without updated functionalities that are released after GAMS 43. Only corrective maintenance support will be provided for this version in the future.
  - all jar files are located under the `[Path/To/GAMS]/apifiles/Java/api/` directory.

7.13.2 43.1.0 (April 2023)

- **GAMSModelInstance** of this version is no longer compatible with GAMS 33 or older.

7.13.3 39.1.0 (April 2022)

- Added `GAMSSet.getSetType` returning property to distinguish between `multi` and `singleton` sets.
- Added `GAMSDatabase.addSet` allowing to set type of Set.
- Fixed defined enumerated value for options **FDOpt** and **DumpOpt**.

7.13.4 35.1.0 (April 2021)

- Fixed `GAMSOptions.setOutput` parameter being ignored in `GAMSJob.run()`.

7.13.5 32.1.0 (July 2020)

- On windows platform, change of window registry configuration used for finding the installed GAMS directory from: `[HKEY_CURRENT_USER|HKEY_CLASSES_ROOT]\gams.location` to: `[HKEY_CURRENT_USER|HKEY_LOCAL_MACHINE]\Software\Classes\gams.location`, effective only when the installed GAMS directory is not found from "PATH" environment variable.

7.13.6 29.1.0 (November 2019)

- Increase the minimum version requirement of the Java Runtime Environment to Java SE 8.
- The remove operation **GAMSDatabaseIterator.remove** is no longer supported. The method now throws an instance of `java.lang.UnsupportedOperationException` and performs no other actions. (To remove all records of the current iterating symbol, use the method **GAMSSymbol.clear** instead.)
- Fixed the behavior of the remove operation **GAMSSymbolIterator.remove**. The method now removes all records of the last GAMSSymbol element returned by an instance of **GAMSSymbolIterator** and can be called only once per call to **GAMSSymbolIterator.next**. The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling the method.

7.13.7 28.2.0 (August 2019)

- Added new method **GAMSSymbolRecord.dispose** for on-demand release of external resources hold by non-java library.

7.13.8 28.1.0 (August 2019)

- Announced a plan to increase the minimum version requirement of the Java Runtime Environment to Java SE 8 with the next major release.

7.13.9 25.1.1 (May 2018)

- Changed the naming scheme of a temporary working directory to be created from `yyyyMMdd_HHmmss` to the prefixed `gams_` (defined by `GAMSGlobals.workingDirectoryPrefix`), in case no working directory has been specified.
- Fixed the behavior when GAMSDatabase is added with the name that already exists, now GAMSException will be raised. (see **GAMSWorkspace.addDatabase(String databaseName)** and **GAMSWorkspace.addDatabaseFromGDX(String gdxFileName, String databaseName)**)
- Calls on `GAMSWorkspace.finalize` and `GAMSSymbolIterator.finalize` are no longer available. As calling a finalizer method can arbitrarily delay the reclamation of object instances and potentially create unpredictable outcome. Whenever the object is no longer needed it is recommened to explicitly dispose the object rather than to rely on the Java garbage collector to do the job. See **GAMSDatabase.dispose**, **GAMSModelInstance.dispose**, and GAMSOptions.dispose.
- All Java native interfaces to Expert-Level APIs are now included in the distributed `GAMSJavaAPI.jar`, located under the `[Path/To/GAMS]/apifiles/Java/api/` directory.

7.13.10 25.0.1 (January 2018)

- The minimum version requirement of Java Runtime Environment for using with GAMS Java API is now Java SE 7.
- New `TransportGDX` example to demonstrate how to import and export GDX files.
- Removed `GAMSSymbol.compact`, deprecated since 24.8.1 (December 2016).
- Changed equivalence behavior of `GAMSSymbol` and `GAMSSymbolRecord` objects. As a result, two symbol objects with the same internal reference are now equivalent, similar to symbol record objects:
  - Two symbols are equivalent if and only if they have the same internal reference.
  - Two symbol records are equivalent if and only if they have the same internal reference.

  The behavior of operator `==` remains unchanged. The following exmaple illustrates the new equivalence behavior:

```
GAMSVariable x1 = db.getVariable("x");
GAMSVariable x2 = db.getVariable("x");
GAMSVariable x3 = x1;
assertTrue(x1.equals(x2));   // true,  previously false
assertFalse(x1 == x2);       // false, previously false
assertTrue(x1.equals(x3));   // true,  previously true
assertTrue(x1 == x3);        // true,  previously true
```

7.13.11 24.8.1 (December 2016)

- Deprecated `GAMSSymbol.compact` and the method will be removed in the next major release
- an exception or an error thrown by GAMSWorkspace.finalize and GAMSSymbolIterator.finalize now must be caught or declared to be thrown. Note that an explicit call on one of these two methods is not recommended unless it is necessary to do so.

7.13.12 24.7.4 (September 2016)

- **GAMSModelInstance.instantiate**: Skip creation of GDX file, which was unreachable from within the API anyway

7.13.13 24.7.1 (March 2016)

- Fixed a bug with the property GAMSOptions.defines : When too many entries were added, all of them were ignored.

7.13.14 24.5.1 (August 2015)

- New examples
  - **Clad** at `[Path/To/GAMS]/apifiles/Java/clad/Clad.java`
  - **SpecialValues** at `[Path/To/GAMS]/apifiles/Java/specialvalues/SpecialValues.java`
- New functions **GAMSWorkspace.addJobFromApiLib** and **GAMSWorkspace.addJobFromNoaLib** that can be used to retrieve models from **GAMS API Library** and **Nonlinear Optimization Applications Library**.

7.13.15 24.4.2 (March 2015)

- New property **GAMSSymbol.getDomainsAsStrings**: get domains of symbol, each element is a string
  Note: If the domain is an alias in GAMS, this call will return the name of the alias, not the name of the aliased set.
- Disable unwanted debug output from Couenne when running with **GAMSModelInstance**
- Change naming scheme of gdx oputput scratch file to sequence number.

7.13.16 24.4.1 (December 2014)

- Modify the handling of GAMS Aliases in the object oriented APIs:
  - If we ask for the number of **GAMSSymbol** in a **GAMSDatabase**, the Aliases will be excluded.
  - If we iterate over all **GAMSSymbol** in a **GAMSDatabase**, Aliases will be skipped.
  - If we ask explicitly for an Alias in a **GAMSDatabase** (**GAMSDatabase.getSet**("a") with a being an Alias) we will get a reference to the **GAMSSet** referenced by the Alias, not the Alias itself.
  - Note:
    - Aliases can appear in a **GAMSDatabase** only, if it was initialized by a GDX file containing an Alias.
    - The new example **Alias** at `[Path/To/GAMS]/apifiles/Java/alias/Alias.java` demonstrates this new behavior.

7.13.17 24.3.3 (September 2014)

Fixed

- a location of listing file when creating a job from (full-path) file without giving a job name

7.13.18 24.3.2 (August 2014)

- Make more GAMS options available through the **GAMSOptions** class:
  - GAMSOptions.EAppendExpand (**enum**, `getter`, and `setter`): Expand file append option
  - GAMSOptions.EAppendOut (**enum**, `getter`, and `setter`): Output file append option
  - GAMSOptions.EDumpOpt (**enum**, `getter`, and `setter`): Writes preprocessed input to the file input.dmp
  - GAMSOptions.EDumpParms (**enum**, `getter`, and `setter`): GAMS parameter logging
  - GAMSOptions.EErrMsg (**enum**, `getter`, and `setter`): Placing of compilation error messages
  - GAMSOptions.EAppendExpand (`getter`, and `setter`): Expanded (include) input file name
  - GAMSOptions.FErr (`getter`, and `setter`): Alternative error message file
  - GAMSOptions.JobTrace (`getter`, and sette): Job trace string to be written to the trace file at the end of a Gams job
  - GAMSOptions.LimCol (`getter`, and `setter`): Maximum number of columns listed in one variable block
  - GAMSOptions.LimRow (`getter`, and `setter`): Maximum number of rows listed in one equation block
  - GAMSOptions.ELogLine (enum, `getter`, and `setter`): Amount of line tracing to the log file
  - GAMSOptions.EOn115 (**enum**, `getter`, and `setter`): Generate errors for unknown unique element in an equation
  - GAMSOptions.Output (`getter`, and `setter`): Output file
  - GAMSOptions.EPageContr (**enum**, `getter`, and `setter`): Output file page control option
  - GAMSOptions.PageSize (`getter`, and `setter`): Output file page size (`=0` no paging)
  - GAMSOptions.PageWidth (`getter`, and `setter`): Output file page width
  - GAMSOptions.Reference (`getter`, and `setter`): Symbol reference file
  - GAMSOptions.ScriptExit (`getter`, and `setter`): Program or script to be executed at the end of a GAMS run
  - GAMSOptions.ESuppress (enum, `getter`, and `setter`): Compiler listing option
  - GAMSOptions.Symbol (`getter`, and `setter`): Symbol table file
  - GAMSOptions.TraceLevel (`getter`, and `setter`): Solvestat threshold used in conjunction with `a=GT`

7.13.19 24.3.1 (July 2014)

New

- **Transport14** example at `[Path/To/GAMS]/apifiles/Java/transport/Transport14.java`.
- **GAMSSymbolRecord.getKey** method: to retrieve key of **GAMSWorkspace** on specified position index.

Changed

- no longer necessary to set up environment variable before running a program
  - possible to specify the GAMS system directory during run time using **GAMSWorkspaceInfo.setSystemDirectory(String directory)** and **GAMSWorkspace(GAMSWorkspaceInfo info)**
- no longer necessary to specify `-Djava.library.path` when running a program
  - if `java.library.path` is specified, the shared libraries will be loaded from `java.library.path`
  - otherwise the shared libraries will be loaded from the class path that contains `GAMSJavaAPI.jar`.
- when exporting a database to GDX: a symbol with real domains from now on will be registered
- in **GAMSJob.run** : always creates output database `OutDB` even if **GAMSExecutionException** has been raised.
- in **GAMSSymbol.copySymbol** when copy into the Universe symbol (**GAMSDatabase.getSet**("∗")) of a **GAMSDatabase** : merge operation will be performed.
- in **GAMSWorkspace** default constructor: API will apply the default setting, that is, finding GAMS system directory from environment variable in the following order (depends on the target platform):
  - Windows: first from `PATH` environment variable. If not found, from the platform windows registry `gams.location`,
  - macOS: first from `PATH` environment variable. If not found, from `DYLD_LIBRARY_PATH`,
  - Unix: from `PATH` environment variable. If not found, from `LD_LIBRARY_PATH`.
- in non-default **GAMSWorkspace** constructor:
  - user can specify a GAMS system directory during run time, API will not search for a GAMS system directory from an environment variable.
  - in case the specified system directory is null or user speicify other workspace attributes but a system directory, API will apply the default setting (see changed in **GAMSWorkspace** default constructor above).

Fixed

- API memory leak issue in **GAMSDatabaseIterator**.
- a bug when reading an option file on Unix platform with non-standard locale

Removed

- all deprecated classes and methods since 24.1.

7.13.20 24.2.3 (May 2014)

Fixed

- API memory leak issue in **GAMSDatabase**.
- a bug in **GAMSDatabase.getDatabaseDomainViolations**.

7.13.21 24.2.2 (March 2014)

Changed

- null string is treated as an invalid key for all record operations of **GAMSSymbol**.

Fixed

- a bug when creating **GAMSDatabase** from source database.
- a bug when initializing a variable type in **GAMSVariable**.
- a bug in **GAMSSymbol**: methods **getVarType** and **getEquType**.
- a bug in **GAMSWorkspaceInfo** : method **getSystemDirectory**.
- exit codes in **GAMSGlobals.ExitCodeMessage**. Improved
- **GAMSModelInstance.instantiate**: make sure that the method does not solve the model, but only prepares everything required for the following **GAMSModelInstance.solve**.

7.13.22 24.2.1 (December 2013)

New

- tutorial: (`GAMS_java_Tutorial.pdf`) under <`Path/To/GAMS`>`/docs/API`. From the GAMSIDE this document can be accessed at Help -> Docs -> API.
- method in **GAMSGlobals.SpecialValues** : **setValue**, to set a GAMS special value
- method in **GAMSOptions**: export, to write an option into a parameter file
- methods in **GAMSWorkspace** : **addOptions** to create a **GAMSOptions** object from either another object or an option file
- methods in **GAMSWorkspace**: **getAPIVersion**, **getAPIMajorReleaseNumber**, **getAPIMinorReleaseNumber**, and **getAPIGoldReleaseNumber**, to retrieve API version number

Changed

- default value of **GAMSGlobals.SpecialValues**: **NAN**, **PLUS_INF**, **MINUS_INF**, and **EPS**
- deprecated `GAMSGlobals.ModelStat.NONOPTIMAL_INTERMED`, replaced by **GAMSGlobals.ModelStat.FEASIBLE**
- deprecated `GAMSOptions.PoolFree4`, replaced by **GAMSOptions.EIntVarUp**
- deprecated the type of `GAMSOptions.NoNewVarEqu`, replaced by enum **GAMSOptions.ENoNewVarEqu**

Fixed

- a bug when running a job with an input directory `IDir` added into **GAMSOptions** object.

7.13.23 24.1.3 (July 2013)

Improve data iterator

- New classes:
  - **GAMSDatabaseIterator** implements `java.util.Iterator`
  - **GAMSSymbolIterator** implements `java.util.Iterator`
- Deprecated Interface:
  - `GAMSSymbolIterable`, replaced by **GAMSDatabaseIterator** and **GAMSSymbolIterator**
- Changes in **GAMSDatabase** :
  - **GAMSDatabase** implements **GAMSDatabaseIterator** instead of `GAMSSymbolIterable`
  - deprecated the implemented methods of `GAMSSymbolIterable`: next(), hasnext(), and remove()
- Changes in **GAMSSymbol**:
  - **GAMSSymbol** implements **GAMSSymbolIterator** instead of `GAMSSymbolIterable`
  - deprecated the implemented methods of `GAMSSymbolIterable`: next(), hasnext(), and remove()
- New methods in **GAMSSymbolRecord**:
  - **moveNext**: to iterate to the next record using the current data iterator criterion
  - **movePrevious** : to iterate to the previous record using the current data iterator criterion

Update example:

- **Transport12** at `[Path/To/GAMS]/apifiles/Java/transport/Transport12.java`

7.13.24 24.1.1 (May 2013)

Changes in **GAMSDatabase**:

- deprecates the `compact` method, as it has no effect anymore.

Changes in **GAMSGlobals**:

- the default value of working directory has been changed from `System.getProperty("user.dir")` to `System.getProperty("java.io.tmpdir")`.

Changes in **GAMSModelInstance** :

- deprecates the instantiate(GAMSOptions options), instantiate(GAMSModifier[]), and instantiate(GAMSOptions, GAMSModifier[]) methods and replaced by **instantiate**(String, GAMSModifier ...) and **instantiate**(String, GAMSOptions, GAMSModifier ...) methods

Changes in **GAMSWorkspace** and **GAMSWorkspaceInfo**:

- deprecates boolean `debug` flag and replaced by a debug level flag (type of a new class **GAMSGlobals.DebugLevel**).
- allows an override of debug level flag from an environment variable `GAMSOOAPIDEBUG`

New methods in **GAMSDatabase**:

- **GAMSDatabase.addEquation**, **addParameter**, **addSet**, and **addVariable**: to add symbols with domain information.
- **checkDomains**: to check whether or not all records of all symbols are within the specified domain of the symbols.
- **getDatabaseDomainViolations**: to retrieve a domain violation information as a list of **GAMSDatabaseDomainViolation** objects.
- **isAutoDomainCheckingSuppressed** and **suppressAutoDomainChecking**: to control whether domain checking will be called when exporting a database.

New class **GAMSDatabaseDomainViolation**:

- contains domain violation information of all symbols (if any) in the database.
- returns call from a new method **GAMSDatabase.getDatabaseDomainViolations**.

New enumeration class **GAMSGlobals.DebugLevel**:

- defines values of different GAMS Debug Levels.

New methods in **GAMSModelInstance**:

- **copyModelInstance**: to copy a **GAMSModelInstance** object.
- **interrupt** : to send an interrupt signal to a running **GAMSModelInstance** object.

New enumerated value of **GAMSModelInstance.SymbolUpdateType**:

- **GAMSModelInstance.SymbolUpdateType.INHERIT**: to specify **GAMSModelInstance.SymbolUpdateType** separately for each **GAMSModifier**.

New methods in **GAMSModifier**:

- `constructor`: to specify **SymbolUpdateType** for each **GAMSModifier** object.
- **getUpdateType**: to retrieve **SymbolUpdateType** property of the object.

New methods in **GAMSSymbol**:

- **checkDomains**: to check whether or not all records of the symbol are within the specified domain.
- **getDomains**: to retrieve a list of domains of the symbol, each element is either a **GAMSSet** (real domain) or a `String` (relaxed domain).

- **getSymbolDomainViolations**: to retrieve a domain violation information as a list of **GAMSSymbolDomainViolation** objects.
- **mergeRecord**: to add a new symbol record in case the record does not exist.

New class **GAMSSymbolDomainViolation**:

- contains domain violation information of the symbol (if any).
- returns call from a new method **GAMSSymbol.getSymbolDomainViolations**.

New methods in **GAMSWorkspace**

- **getGAMSVersion**: to retrieve information about GAMS Version.
- **getGoldReleaseNumber**: to retrieve GAMS GOLD Release Number.
- **getMajorReleaseNumber**: to retrieve GAMS Major Release Number.
- **getMinorReleaseNumber**: to retrieve GAMS Minor Release Number.

Fixed a bug when iterating through the records of a **GAMSSymbol**

Changes of location of examples:

- **Bender Examples**: from `[Path/To/GAMS]/apifiles/Java/Benders*.java` to `[Path/To/GAMS]/apifiles/Java/`
- **Cutstock**: from `[Path/To/GAMS]/apifiles/Java/Custock.java` to `[Path/To/GAMS]/apifiles/Java/cutstock/`
- **ConsoleInterrupt**: from `[Path/To/GAMS]/apifiles/Java/ConsoleInterrupt.java` to `[Path/To/GAMS]/apifiles/Java/interrupt/ConsoleInterrupt.java`
- **Transport Examples**: from `[Path/To/GAMS]/apifiles/Java/Transport*.java` to `[Path/To/GAMS]/apifiles/Java/transport/Transport*.java`
- **Warehouse**: from `[Path/To/GAMS]/apifiles/Java/Warehouse.java` to `[Path/To/GAMS]/apifiles/Java/wareh`

New examples :

- **SimpleCutstock** at `[Path/To/GAMS]/apifiles/Java/cutstock/SimpleCutstock.java`
- **DomainCheck** at `[Path/To/GAMS]/apifiles/Java/domain/DomainCheck.java`
- **Transport13** at `` `[Path/To/GAMS]/apifiles/Java/transport/Transport13.java ``
- **Tsp** at `[Path/To/GAMS]/apifiles/Java/tsp/Tsp.java`

7.13.25 24.0.2 (February 2013)

New method in **GAMSSymbol**

- added new method **copySymbol**.

7.13.26 24.0.1 (December 2012)

This release contains a beta version of the object-oriented Java API that can be used to control GAMS from a Java program. It allows the seamless integration of GAMS into Java by providing appropriate classes for the interaction with GAMS. GAMS Java API objects allow a convenient way to exchange input data and model results with in-memory representation of data (**GAMSDatabase**), and to create and run GAMS models (**GAMSJob**) that can be customized by GAMS options (**GAMSOptions**). Furthermore, they introduce a way to solve a sequence of closely related model instances in the more efficient way (**GAMSModelInstance**).

- A Java program that uses object-oriented Java API requires at least Java SE 5 to compile and run.
- All classes are distributed within one single jar file `GAMSJavaAPI.jar` with a namespace `com.gams.api`, located under the `[Path/To/GAMS]/apifiles/Java/api/` directory.
- Java program examples are distributed with namespace `com.gams.examples`, located under `[Path/To/GAMS]/apifiles/Java/` directory.
- Installation and detailed documents can be found in `[Path/To/GAMS]/apifiles/readme.txt` and `[Path/To/GAMS]/docs/API/GAMS_java.pdf`.
- Javadoc for `GAMSJavaAPI.jar` can be found under `[Path/To/GAMS]/apifiles/java/api/javadoc` directory.

7.14 GAMS Environment Object Options

Option	Description	Default
AlgFileType	File type of scratch files	1
Cheat	optCheat solver default Range: [-∞, ∞]	0
CurSolver	Solver id of the last gevcallsolver	0
CutOff	CutOff solver default Range: [-∞, ∞]	0
DomLim	Domain violation limit solver default	0
FDDelta	Step size for finite differences Range: [1e-9, 1]	1e-5
FDOpt	Options for finite differences Range: [0, 14]	0
GamsVersion	GAMS CMEX Version	999
GenSolver	Indicator for generic solver executable	0
HeapLimit	Maximum Heap size allowed	1e20
IDEFlag	IDE Flag 0 unknown environment 1 runs under GAMS IDE	0
Integer1	Range: [-∞, ∞]	0
Integer2	Range: [-∞, ∞]	0
Integer3	Range: [-∞, ∞]	0

Option	Description	Default
Integer4	Range: [-∞, ∞]	0
Integer5	Range: [-∞, ∞]	0
IterLim	Iteration limit solver default	2000000000
Keep	Keep scratch directory Range: [0, 1]	0
License1	License line 1	
License2	License line 2	
License3	License line 3	
License4	License line 4	
License5	License line 5	
LogOption	Log option 0 no log output 1 log output to screen (console) 2 log output to logfile 3 log output to standard output 4 log output to logfile and standard output	1
NameCtrFile	Control file name	
NameCurDir	Current directory	
NameExtFFile	External Function file name	
NameGamsDate	GAMS Date	
NameGamsTime	GAMS Time	
NameInstr	Instruction file name	
NameLogFile	Log file name	
NameMatrix	Matrix file name	
NameParams	Parameter file name	
NameScenFile	Scenario file name	
NameScrDir	Scratch directory	
NameScrExt	Scratch extension	dat
NameStaFile	Status file name	
NameSysDir	System directory	
NameWrkDir	Working directory	
NodeLim	Node limit solver default	0
OptCA	optAbsolute Optimality criterion solver default	0
OptCR	optRelative Optimality criterion solver default	0.1
PageSize	Output file page size	78
PageWidth	Output file page width	123

Option	Description	Default
Real1	Range: [-∞, ∞]	0
Real2	Range: [-∞, ∞]	0
Real3	Range: [-∞, ∞]	0
Real4	Range: [-∞, ∞]	0
Real5	Range: [-∞, ∞]	0
Reform	Reform option	100
ResLim	Resource (CPU) solver default limit	1000
SysOut	Solver Status file reporting option Range: [0, 1]	0
ThreadsRaw	Range: [-∞, ∞]	1
TryInt	TryInt solver default	0
UseCheat	UseCheat solver default	0
UseCutOff	UseCutOff solver default	0
WorkFactor	Work space multiplier for some solvers	1
WorkSpace	Work space for some solvers	0

7.15 GAMS Modeling Object Design

This document gives a brief introduction to GAMS' new model-solver API, including a description of its design goals and philosophy. This document is not intended to be used as a user's manual but rather to give a picture of what the new API can do and a description of some of the conceptual underpinnings for the new API.

Further material: Presentation at ICS 2011

7.15.1 Introduction

Historically, there have been a number of different GAMS solver interface libraries, i.e. libraries that enabled a solver to get all required information from a GAMS model, compute the solution, and return the solution, status information, and other information (e.g. solver log messages, nonlinear function evaluation error messages, node counts, time used) to GAMS. Each library was unique in some way, e.g. in the computer language or subset of GAMS model types it supported. Each library had its own advantages and disadvantages, but all of them were feature-poor and supported a limited scheme of operation: GAMS/Base would write *scratch files* containing the model to be solved, the GAMS solver link executable would use one of the solver interface libraries to read the model from scratch file and write a solution file, and GAMS/Base would read this solution file. While this scheme worked for many years it was inconvenient for the library user, difficult to maintain, slow, and a barrier to further development.

To address these issues and others, GAMS developed a new model-solver interface library, called GMO (GAMS Model Object), and a companion library called GEV (GAMS Environment Object). The two libraries are distinct and help to separate the purely model-specific tasks and information handled by

GMO from the computing and solver environment that the model will be solved in, handled by GEV. The two libraries are typically used together and we will sometimes refer to them as simply GMO. For legacy reasons the new libraries support the old file-based scheme to communicate a model through scratch files.

In what follows, we first describe the basic structure, organization, and usage of GMO (Basic organization). Client applications can be split into two types: those defining or updating the model (Modeler Access to GMO) and those passing the model to a solver (Solver Access to GMO). We next describe some ways to update a model in Updating a GMO instance.

7.15.2 Basic organization

In this section we outline the basic organization of GMO. GMO is implemented as a shared library that interfaces to different languages via language-specific header files and interface code. The header files and code are included in source form with the GAMS distribution. Since the interface code employs dynamic loading, each GMO session begins with a call to initialize the required shared library (i.e. load it and all its symbols) and return an empty model object.

The next step is to load a model into the model object. This is typically done with a single wrapper call, as in the case of loading a model from model scratch files prior to solving in one of the GAMS solver links. It is also possible build up a model from scratch with a sequence of calls to provide the model column-wise or row-wise. If the model is built up column-wise or row-wise, additional information (e.g. about nonlinearities) may be passed to the object after the basic structure is built up. Once the model is complete, a finalization routine is called. This routine prepares the model object for efficient access by the user. Typically the model does not change after this point. There are exceptions to this (see Updating a GMO instance), but they usually involve small modifications to the existing model (like adding alternative bounds) or the addition of constraints and variables that require the *re-finalization* of the model.

When a model is fully loaded, the GMO client can start accessing the model. Typically the GMO client doing this will be a solver, but other clients (e.g. the CONVERT *solver*, the Examiner utility, or a model structure browser) exist also. In addition to querying the model, the client can also request that the model be presented or viewed in various different ways. For example, GMO can be requested to use an objective function or an objective variable. Most types of information are available immediately when the model is loaded, but some type of queries (e.g. Hessians, alternative QP forms) are less common and relatively expensive in time and/or space: these require that an initialization routine be called before the information is requested. Once the client has completed its work it reports on the solution found, if any. Typically, the model object is destroyed at this point. Model access by a client, especially solvers, is an important topic that is discussed in Solver Access to GMO.

The variety of data that go into the interface library when a model is defined and that a solver can request during solution is large, but the data can be divided into two types - data defining the model (e.g. number of rows and columns, coefficients, nonlinear functions) and data defining the solver environment (e.g. available and default GAMS solvers, license info, algorithm tolerances and limits). It is sometimes useful to separate these two types of data, so that it is possible to define and work with a model without specifying anything about the environment it exists in. The environment can be specified separately, and multiple models can make use of the same basic environment. To facilitate this, the solver interface is in two parts: the GMO interface deals with model-specific information and the GEV interface with information about the environment. The two environments are linked, since they are typically used together, with one (or multiple) models existing in one environment, but the separation makes it possible to define a model without defining its environment, etc. Since the bulk of the information and the code lies with GMO, we usually speak of the two libraries as one, called GMO, but it is worth remembering that GEV is a separate object.

7.15.3 Solver Access to GMO

One of the primary purposes of GMO is to provide convenient, efficient access to the model. GAMS links to many solvers, each solver link using GMO to access the model in different ways, so there is a wide array of possible methods to access the model in GMO. We describe them in this section.

There are multiple but equivalent views possible for the same model. For example, the model can be formulated with an objective function or with an objective variable. Free rows can be removed or left in, and the rows and/or columns can be permuted. Prior to accessing the model, the solver link can choose what view it would like to use. The view can be changed later, but since this can obviate the data obtained previously, the choice of view is usually made early on and left unchanged. Moreover, a view can be stored and restored in case multiple users access the same model.

All solvers will need to get basic information about a model - simple things like row and column counts, nonzero counts, etc. These are available via simple access routines. Since the model is essentially fixed (assuming the view does not change), basic information is pre-computed when the model takes its final form and is available at essentially no cost. Information about the rows and columns (e.g. variable level values, marginal values, variable priorities, row types) is also available, either for single rows/cols or in an array for all rows/cols at once.

For linear models, the only information left to get is the matrix of coefficients. This is available in a number of different forms e.g. column-wise or row-wise, either as a complete matrix returned with one call or as separate rows or columns returned via many calls. For nonlinear models, there are more variants on this theme: calls to return the function value or the function value and derivatives, calls for constraints and calls for the objective function. You can request that the nonlinear model be treated as its linearization around a certain point. There are also routines to do interval evaluations of functions and gradients. The GMO library prepares the model object to handle all these calls efficiently when the model takes its final form.

Nonlinear models present several additional challenges. When NL functions are evaluated, mathematical errors can occur (e.g. sqrt(-1)). GMO supports different ways to log and handle these errors. Some algorithms will require or want 2nd-order information. Since this can be relatively expensive to compute, the solver must first call an initialization routine specifying what is desired (e.g. only Hessian-vector products, Lagrangian Hessian, single-row Hessians) and potentially a limit on the memory dedicated to computing Hessians. After this initialization, 2nd-order information is available.

Once the solver finishes, the solution (if any) and related status values must be reported back to GMO. There are a number of convenience routines included in GMO for this purpose.

A convenient interface was and is an important design goal for GMO, especially in the area of solver access to the model. Convenience takes many forms: a number of ways to get the same model data in order to support a variety of different solver libraries, self-contained calls with minimal requirements on previous setup for correct behavior,and a modern interface adhering to design principles like information hiding, encapsulation, and an object model. By keeping these goals at the fore when designing GMO we have made accessing the model convenient and efficient, which greatly simplifies the task of building a correctly-functioning solver link.

In addition to providing convenient access to the model, a solver link needs to expose a particular API to be recognized by the GAMS system. This is not directly connected to the GMO API itself but knowledge about the expected solver link library API is required so we include it here.

GAMS calls a solver through the GEV API function `gevCallSolver` which eventually results in a sequence of calls to functions in the solver link library. GAMS loads solver link libraries dynamically at run time. The name of the library is specified in the configuration file together with the three letter audit code, e.g., `cpx` for CPLEX. All functions in the solver link library are prefixed by this audit code (we use `xyz` in this example). The following functions are recognized:

- `void xyzInitialize(void)` (optional): Function called when loading the solver link library.

- `void xyzFinalize(void)` (optional): Function called when unloading the solver link library.
- `int xyzCreate(void** handle, char* msg, int sizeBuf)`: Function to create a solver link *object*. Returns 0 and stores a non-NULL pointer to the object in `handle` on success, otherwise an error message should be stored in `msg`.
- `void xyzFree(void** handle)`: Function to destroy a solver link object.

Depending on the *solver interface type* (again communicated as part of the configuration file) different functions need to be in the API. For solver interface type 1 we have:

- `int xyzReadyAPI(void* handle, void* gmoHandle)`: Function to set up the model instance in the solver space. Returns 0 on success.
- `int xyzCallSolver(void* handle)`: Function to do the actual solve and solution reporting. Returns 0 on success.
- `int xyzModifyProblem(void* handle)` (optional): Function to modify the problem. Returns 0 on success.

For solver interface type 2 we only have a single call:

- `int xyzCallSolver(void* handle, void* gmoHandle)`: Function to setup and solve. Returns 0 on success.

Usually, a *solve* in GAMS triggers the following sequence of function: `xyzCreate`, `xyzReadyAPI`, `xyzCallSolver`, `xyzFree` (interface type 1) or `xyzCreate`, `xyzCallSolver`, `xyzFree` (interface type 2) and the separation of `xyzReadyAPI` and `xyzCallSolver` seems unnecessary.

With Gather-Update-Solve-Scatter (GUSS) and the `GAMSModelInstance` class in various APIs we repeatedly solve a model with the same rim but different data (bounds, rhs, matrix coefficients). In this case the sequence of solves results in a sequence of function calls. Without the presence of `xyzModifyProblem` the following sequences will be called: `xyzCreate`, `xyzReadyAPI`, `xyzCallSolver`, `xyzReadyAPI`, `xyzCallSolver`, `xyzReadyAPI`, `xyzCallSolver`, ..., `xyzFree` (interface type 1) or `xyzCreate`, `xyzCallSolver`, `xyzCallSolver`, `xyzCallSolver`, ..., `xyzFree` (interface type 2). So, again there is little point (but possible) to split the link work in `xyzReadyAPI` and `xyzCallSolver` without the presence of `xyzModifyProblem`. If `xyzModifyProblem` is present (only with flavor 1) the following sequences will be called: `xyzCreate`, `xyzReadyAPI`, `xyzCallSolver`, `xyzModifyProblem`, `xyzCallSolver`, `xyzModifyProblem`, `xyzCallSolver`, ..., `xyzFree`.

7.15.4 Modeler Access to GMO

In normal operation the modeler creates a model in GAMS source form and the GAMS/Base module handles all the details necessary to build up model instances in GMO. However, there are cases (e.g. solving a model many times over with different scenario data) where it is useful to access and modify a GMO instance outside of GAMS/Base. GMO provides routines to support this, specifically routines to update the model (see Updating a GMO instance), solve the model with one of the GAMS solvers available as part of the GAMS environment and GEV, and retrieve the solution.

Access to a GMO instance is usually done using integer indices, `[1..m]` for the rows and `[1..n]` for the columns. However, it is also possible to reference rows and columns using the notation from the original GAMS model, e.g. `x('seattle','topeka')` or `demand('chicago')`. This is more convenient when specifying what data should be updated to define a new scenario. In order to update a model instance in this case, it is useful to have a convenient, efficient way to translate row and column indices from one form to the other. This translation is done by the *model dictionary* interface DCT.

The DCT API provides information about the variables, equations, and sets used to define the GMO model instance. In addition, given a row or column index, it will translate this index into the equation or variable name and the list of set labels that correspond to this index. It will also do the inverse operation, translating an equation/variable name and list of labels into a row or column index. GAMS builds models in a way that makes this translation very efficient.

This organization is also useful when updating models to create new scenarios. Models are organized so that:

- There is one ordered universe of all labels appearing in the variables and equations of a given model. The order for the labeling does not change from symbol to symbol within a model.
- All of the columns in a model corresponding to a given variable are contiguous. (Ditto for rows and equations)
- The columns corresponding to a given variable are sorted by their index labels, using the universal order, with the left-most index varying slowest. (Ditto for rows and equations)
- GDX data is sorted in the same way: sorted by index label order, left-most index varying slowest. This can be very useful, *provided the model and the GDX data use the same label ordering.*

Using the various GAMS component libraries, it is possible to create a model updating interface that is both convenient for the user (by using the symbolic names referenced in the GAMS source) and efficient in updating the model data.

7.15.5 Updating a GMO instance

There are several possible ways to modify a model contained in a GMO instance. By updating, we mean an actual model change, rather than a change in the view that doesn't alter the set of solutions.

In perhaps the simplest case, the user can specify alternate variable bounds, as is useful when implementing a branch and bound code. The original bounds are retained in this case, so that the model can be reverted to the original bounds with a single call.

It is possible to linearize a model around a given point. This requires no information from the point to linearize around and can be undone easily. This is almost a change in view but since the solutions of the original and linearized models may be quite different we can also consider this a model update.

Many algorithms (e.g. outer approximation, Benders decomposition, Danzig-Wolfe decomposition) involve the addition of rows, and columns incident on these rows, to a core model. Such algorithms can be implemented effectively using GMO by taking advantage of GMO facilities to add rows and columns. There will be only a small amount of work required to prepare the GMO instance for use after each round of additional rows and columns. Note that all the changes to the model in this case are *additions*: the existing structure and data are left as is.

Finally, there are facilities in GMO to update variable bounds (as mentioned before), variable types (e.g. to relax single discrete variables), and the right-hand-side of constraints. The user cannot directly modify matrix coefficients. There is a much better way to safely update exogenous data in a model: GAMS can provide a model instance in which GAMS parameters are visible and can be updated. A reevaluation of the expressions in the models provides the updated matrix elements. The underlying concept is also used in Gather-Update-Solve-Scatter (GUSS) and described in more detail in `this paper`. This is a powerful tool to efficiently do e.g. Monte-Carlo simulation without the need for GAMS to regenerate the entire model over and over.

Chapter 8

Appendix

- Glossary - An alphabetical list of GAMS terms.
- Third-Party Codes - A list of third-party codes that are included in the GAMS distribution.
- Bibliography

8.1 Glossary

acronym	A GAMS data type used for storing non-numeric (aka logical or categorical) data. For more details, see section Acronyms.
alias	An alternative name for a set, used to have multiple indices running over a common set. For more information, see section The Alias Statement: Multiple Names for a Set.
algorithm	A sequence of actions to perform in order to solve a problem. GAMS *solvers* contain algorithm implementations, so the two terms are sometimes used interchangeably.
assignment	The statement used to assign values to an identifier. For a detailed introduction, see section The Assignment Statement.
basic	A column is basic if it is in the basis maintained by the solution method for the problem in question (e.g. by the simplex method for LP). A row is basic if the elementary column associated with its slack variable is basic.
binding	An inequality constraint or variable bound is binding when the value of the associated slack is zero, i.e. when it is satisfied as an equality.
bounds	Upper and lower limits on the possible values that a column may assume in a feasible solution. May be infinite, i.e. no limit is imposed.
column	An individual decision variable in the model passed to a solver. Also called a `single variable` in the GAMS listing file. An indexed GAMS variable typically contains many columns.

compilation	The initial phase of GAMS processing, when the program is being checked for syntax and consistency.
compile-time constant	A compile-time constant is a string that will be replaced at compile-time with a fixed value. These constants are usually related to a function, model attribute or option and are used for their mnemonic and descriptive value. See section Compile-Time Constants.
compile-time variable	Special variables that are substituted with their values at compile-time. For further information, see section Compile-Time Variables.
constraint	A relationship between variables that must hold in a feasible solution. A constraint can refer to a single relationship (i.e. a row) or to a collection of rows, such as an equation.
continuous	Used to describe functions (in the usual mathematical way) and variables. Roughly speaking, a function is continuous if its graph has no breaks or jumps in it. A continuous variable may assume any value within its bounds, in contrast to binary or integer variables.
controlling sets	See driving sets.
data types	Each symbol or identifier must be declared as one of the available data types, e.g set, parameter, equation, or `model`. Note that scalars and tables are not separate data types, but convenient input formats for certain parameters. See also section Data Types and Definitions.
decision variable	A decision variable (aka endogenous variable) represents a decision to be made, e.g. the amount to produce or consume or the unit price to charge. Decision variables are the unknowns of a mathematical programming model. In contrast, exogenous variables or parameters are outside of the decision maker's control..
declaration	The introduction of an identifier along with the specification of its data type. A declaration may also include the specification of data or initial values, in which case it also acts as a definition. See also section Classification of GAMS Statements.
default	The value used, or the action taken, if the user provides no information.
definition	A statement specifying the content or data associated with a GAMS identifier, e.g. the equations in a model, the algebra of an equation, the elements of a set, or the initial values of a parameter. Definitions are processed during compilation. See also section Classification of GAMS Statements.
direction	The direction of optimization, i.e. maximization or minimization.
discontinuous	A function is discontinuous if it is not continuous. Sometimes, we call a function discontinuous if its derivatives are not continuous, i.e. it is nonsmooth.
discrete	A discrete variable is any variable that is not continuous. For example, binary and integer variables are discrete. For more details, see section Types of Discrete Variables.

dollar control option	Directives or options used to control input or output details associated with the GAMS compiler. They are introduced in chapter Dollar Control Options.
dollar operator	An operator used to indicate and define conditional expressions in assignment statements and equation definitions. For more information, see section The Dollar Condition.
domain checking	A system of checks that ensures that operations and relationships involving sets and domains are logically consistent. For example, referencing `pop(t,city)` when we have a declaration of `pop(city,t)` is a logical error that domain checking will catch. For more information, see section Domain Checking.
domain of definition	The set of tuples (i.e. label combinations) for which an indexed equation is defined. When a solve statement is executed, a row of the equation is generated for each member of the domain of definition.
domain restriction condition	The narrowing of the domain of definition with a dollar operator. When used in this way, the dollar operator occurs on the left of the symbol `'..'` in an equation definition. For more details, see section Dollar Control over the Domain of Definition.
driving set	The set or list of sets that an indexed operation or domain runs or is defined over. Loops, indexed operations like `sum`, and domains of definition all make use of driving sets.
dynamic set	A set is dynamic if it has been changed with an assignment statement. Dynamic sets cannot be used with lag operations or in domain declarations. For more information, see chapter Dynamic Sets.
echo print	In the output file, the echo print is a listing of the input with added line numbers. Details are given in section The Echo Print of the Input File.
endogenous	Endogenous variables are used in economics, statistics, and other disciplines. In the GAMS context, they are synonymous with variables, i.e. values that change when a solve statement is processed. See also the discussion in section Functions in Equation Definitions.
equation	The GAMS data type used to specify required relationships between activity levels of variables. They are introduced and discussed in detail in chapter Equations.
execution	The second phase of GAMS processing, when GAMS is actually carrying out data transformations or generating a model.
execution statements	Instructions to carry out actions such as data transformations, model solves and report generation. Examples include the assignment statement, the loop statement and the solve statement. For a full list, see section Classification of GAMS Statements.

exogenous	Exogenous variables are used in economics, statistics, and other disciplines. In the GAMS context, they are synonymous with constants, i.e. values that do *not* change when a solve statement is processed. These are most often parameters but any variable or equation field will be treated exogenously when it appears in an equation. See also the discussion in section Functions in Equation Definitions.
explanatory text	See text.
extended arithmetic	In GAMS, the usual computer arithmetic is extended to include special values (e.g. `inf`) and the results of operations and functions that use them. For example, `6 + inf` is `inf` and `min(6, inf)` is `6`. See section Extended Range Arithmetic and Error Handling for details.
external equation	An equation defined in an external module. For example, on Windows systems, external equations are defined by a DLL. See section External Equations for details.
extrinsic function	A function that is imported into a GAMS program from an external function library. Once imported, extrinsic functions can be used in the same way as intrinsic functions like `cos` and `exp`. See section Extrinsic Functions for details.
e-format	A convenient text-based (i.e. unsuperscripted) way to represent numbers in scientific notation in which the exponent of the 10 is prefixed by the letter `e`. For example, one US mile = 1.609344e+03 meters.
feasible	Often used to describe a model (or a subset of constraints within a model) that has at least one feasible solution, but also used to describe a point that satisfies a set of constraints. See also infeasible.
feasible solution	A solution to a model in which all column activity levels are within the bounds and all the constraints are satisfied.
identifier	The name given to a data entity in a GAMS program. Also called a symbol. See section Identifiers for further details.
indexed operation	An operation (e.g. sum or max) that is performed over one or more indices. See also section Indexed Operations.
inequality constraint	A constraint in which the imposed relationship between the columns is not specified with an equality but instead with an inequality (e.g. "greater than or equal to", "less than or equal to"). The GAMS syntax `=G=` and `=L=` is used in equation definitions to specify these relationships.
infeasible	Not feasible. Used to describe either a model that has no feasible solution or an intermediate solution or point that is not feasible (although feasible solutions may exist).
initialization	Associating or assigning *initial values* to an identifier as part of the declaration or definition of the identifier. We typically talk about the initialization of only those identifiers (e.g. sets and parameters) that can be assigned different values later. For identifiers like models and equations, we say they are *defined*, not *initialized*.

intrinsic function	Functions that are provided by and part of GAMS, including mathematical, string, logical, time/calendar, and utility functions. Contrast to extrinsic functions.
label	Sets are built up from labels (aka *elements*). One-dimensional sets are collections of labels, while multi-dimensional sets contain tuples of labels. See section Labels for details.
list	See list format.
list format	GAMS data may be initialized by data in list format, i.e. a list where each tuple of labels in the data is specified in full. Data may also be displayed in list format, in which case each nondefault value is displayed via a fully-specified tuple of labels.
macro	A macro is a fragment of code that has been given a name: whenever the name is used, that name is replaced by the contents of the macro (i.e. the fragment of code). This is useful for defining and automating structured text replacement, e.g. to replace the text `MYOBJ(x)` with `[sqr(x) + exp(x-2)/7]`. See section Macros in GAMS for details.
marginal	Marginal values (aka "dual values", "reduced costs", "shadow prices", or "multipliers") are stored in the `".m"` variable attribute or equation attribute. The GAMS sign convention is this: the marginal value represents the amount and direction of change in the objective value given a *unit increase* in the binding constant (e.g. an active variable bound or right-hand side).
matrix element	See nonzero element.
model generation	An initial step in processing a solve statement, where a model instance is generated (based on the equation definitions and the data referenced by those definitions) for the solver.
model list	The list of equations that are part of the model, as specified in the model statement. For further details, see section The Model Statement.
model status	An integer returned by the solver that gives information about the model (e.g. INFEASIBLE), about the point returned by the solver (e.g. LOCALLY INFEASIBLE), or both (UNBOUNDED NO SOLUTION RETURNED). For an overview of all values, see section Model Status.
nonbasic	A column that is not basic and (in nonlinear problems) not superbasic. Typically, nonbasic columns behave or are treated as if they are fixed at a bound. If the solution is feasible, the value of a nonbasic column will equal a finite bound (or zero if there are no finite bounds).
nonlinear nonzero	In a linear programming problem, the nonzero elements of the constraint matrix are constant. In a nonlinear problem, elements of the Jacobian matrix vary where variables appear nonlinearly. These non-constant Jacobian elements are called nonlinear nonzeros.

nonoptimal	Not optimal. A solution is nonoptimal if other solutions exist with better objective values. A variable is nonoptimal if its marginal (aka reduced cost) has the wrong sign. For example, in a minimization, a variable at lower bound having a negative marginal is nonoptimal. In the simplex method, a variable is nonoptimal if it is nonbasic and would improve the objective if it entered the basis.
nonsmooth	A function is nonsmooth if it is not smooth, i.e. if the function itself or its derivatives are not continuous. For example, the absolute value function is nonsmooth because of the kink at the origin: its derivative is not continuous there.
nonzero element	An element or coefficient of a matrix (e.g. the constraint matrix of an LP or the Jacobian matrix of an NLP) that is not zero. Most mathematical programming problems are *sparse*, i.e. only a small proportion of the elements in the constraint or Jacobian matrix are nonzero.
objective row (or function)	Solver systems often require the specification of a row or (for nonlinear systems) a function whose value will be optimized. A GAMS model, in contrast, is solved by specifying a scalar objective variable to be optimized.
objective value	The current value of the objective row or of the objective variable.
objective variable	The variable to be optimized, as specified in the solve statement.
optimal	A feasible solution in which the objective value is the best possible..
optimality gap	The optimality gap is a metric for the distance (or an upper bound on the distance) between the objective value of the current feasible solution and the optimal objective value.. In GAMS, MIP and global solvers terminate when the optimality gap is sufficiently reduced: see the options optCA and optCR for details.
option	A control that allows users to change or influence the behavior or parameters in many different parts of the system. Options in GAMS may be set in three different ways: with model attributes, command line parameters, and option statements. A full list of all options with detailed descriptions is given in section Detailed Descriptions of All Options.
ordered set	A one-dimensional set is ordered if the definition or initialization of the elements in the set corresponds to the order of the labels in the universe. Only sets that are ordered (and static) can be treated as sequences, i.e. used in lags and leads and with `Ord`-type operations. For more information, see chapter Sets as Sequences: Ordered Sets.
output file	A file, also called the listing file, produced (by default) by the run of a GAMS program. It contains output that describes or logs the run in question. For details see chapter GAMS Output.
parameter	A data type in GAMS used to store (indexed) constant data. For details on parameters, see chapter Data Entry: Parameters, Scalars and Tables.

problem type	A class of models that is characterized or defined by the type of algebra accepted (e.g. linear or nonlinear), the GLOSSARY_variable_type "variable types" allowed (e.g. continuous vs. discrete), and the definition of what it means to solve the model or problem (e.g. optimization vs. MCP). A list of all problem types in GAMS is given in section Classification of Models.
program	A GAMS input file. Typically a program defines a model and solves it, but programs can also work only with data or act as scripts that call other programs.
relational operator	This term may be used in two ways. First, in an equation definition, it describes the type of relationship that the equation specifies. An example is equality, as specified with the symbol `=e=`. For more information, see Table Equation Types. Second, in a logical expression, a relational operator compares two numerical expressions and returns a logical value. For details see section Logical Conditions: Numerical Relational Operators.
right-hand side	The value of the constant term in a constraint.
row	An individual constraint in the model passed to a solver. Also called a `single equation` in the GAMS listing file. An indexed GAMS equation typically contains many rows.
scalar	An un-indexed parameter, or the statement used to declare or define this parameter. See also section Scalars.
set	A collection of elements (aka *labels*) or element tuples. The set statement is used to declare and define a set. Sets are introduced and discussed in detail in chapter Set Definition.
simplex method	An algorithm often used to solve linear programming problems.
singleton set	A special set that has at most one element (zero elements are allowed as well). For more information, see section Singleton Sets.
slack	The amount by which an inequality constraint or variable bound is not binding.
slack variable	A non-negative variable introduced to represent the slack in an inequality constraint and to convert the inequality into an equality.
smooth	A function that is continuous and whose derivatives are all continuous is smooth. Sometime we consider a function smooth if enough of its derivatives are continuous.
solver	An implementation of an algorithm or algorithms for solving models of a given problem type or types. An example is MINOS and QUADMINOS, which is used to solve both linear and nonlinear programming problems.
solver status	An integer returned by the solver that indicates the solver termination condition, i.e. why the solver stopped. For details, see section Solver Status.

statements	Statements (aka units or sentences) are the fundamental building blocks of GAMS programs: each program is a sequence of statements. Statements are used to declare identifiers, define equations, create loops, and solve models. A full list of GAMS statements is given in section Classification of GAMS Statements.
static set	Used in two slightly different ways: a set is static if it *has not* changed (i.e. it is not dynamic) or if it *cannot* change, i.e. it is *immutable*. Usually, sets are either used in ways that make them immutable (e.g. as domains or with lags and leads) or in ways that make them dynamic (i.e. by assigning to them): in such cases, static is synonymous with immutable.
superbasic	Some algorithms for nonlinear programming make use of superbasic variables that are neither basic (i.e. in the basis) nor nonbasic (i.e. temporarily fixed at a bound). These algorithms often search in the space defined by these superbasic variables.
symbol	An identifier.
table	A table statement is often a convenient way to define a parameter having two or more dimensions. For details see section Tables.
text	An optional description associated with an identifier or with an element of a set. For details see section Text.
unique element	A label used to define set membership.
universe	Also called the *universal set*, it is denoted by the symbol '$*$' and contains all the labels that have been declared in the program plus any labels used as if they were declared (e.g. `link('sink',j) = NO;`). If an identifier is declared using the universe, that effectively turns off or limits domain checking for that identifier. For more details, see section The Universal Set: $*$ as Set Identifier.
variable type	Variables are classified or typed based on their default bounds and the values they are allowed to take. This latter classification partitions variables into continuous or discrete. For example, `free` variables have no default bounds and can take any value between their bounds: they are continuous. In contrast, `binary` variables can take only the values zero and one. See section Variable Types for details.
zero default	Some GAMS data (e.g. parameters, lower bounds for positive variables) have a default value of zero. These default zero values are not stored. However, unless other values are provided via initialization or assignment, zero will be used when the data value is taken.

8.2 Third-Party Codes

The following table lists third-party codes that are included in the GAMS distribution.

Software	Version	Description	License
Addict	3.4	Spell Checker	proprietary

Software	Version	Description	License
ALPHAECP	2.11.01	Mixed-Integer Nonlinear Programming Solver	proprietary
ANTIGONE	1.1	Mixed-Integer Nonlinear Programming Solver	proprietary
BARON	24.03.10	Mixed-Integer Nonlinear Programming Solver	proprietary
BLISS	0.77	Compute canonical labelings and automorphism groups of graphs	LGPL 3.0
Boost	1.84	Portable C++ source libraries	Boost 1.0
CBC	2.10.11	Mixed-Integer Linear Optimization Solver	EPL 2.0
Cerberus	1.3.5	Data validation library	ISC
certifi	2023.11.17	Mozilla's bundle of Certificate Authority certificates	MPL 2.0
Cgl	0.60.8	Cut Generators for Mixed-Integer Linear Optimization	EPL 2.0
cJSON	1.7.12	Ultralightweight JSON parser in ANSI C	MIT
Clp	1.17.9	Linear Optimization Solver	EPL 2.0
CoinUtils	2.11.10	Classes and helper functions used by COIN-OR projects	EPL 2.0
CONOPT 3	3.17P	Nonlinear Programming Solver	proprietary
CONOPT 4	4.33	Nonlinear Programming Solver	proprietary
COPT	7.0.6	Mixed-Integer Linear and Quadratic Programming Solver	proprietary
CPLEX	22.1.1.0	Mixed-Integer Linear and Quadratic Programming Solver	proprietary
CppAD (in SCIP)	20180000.0	C++ Automatic Differentiation	EPL 2.0
CppAD (in SHOT)	20210606	C++ Automatic Differentiation	EPL 2.0
cpr	1.10.2	C++ Requests: Curl for People, a spiritual port of Python Requests	MIT
Curl	8.6.0	transferring data with URLs	curl
D6OnHelpFix		Fixes OnHelp events in Delphi 6 and 7	proprietary
DataTables	1.13.4	Advanced Features for HTML Tables	MIT

Software	Version	Description	License
DECISC		Solver for two-stage stochastic LPs with recourse	proprietary
DECISM		Solver for two-stage stochastic LPs with recourse	proprietary
Delphi		Borland Runtime Package	Borland
DIERCKX/FITPACK		Calculation of smoothing splines for various kinds of data and geometries, with automatic knot selection	public domain
Doxygen Awesome	2.1.0	CSS theme for Doxygen HTML-documentation	MIT
Doxygen Output	1.8.13	Doxygen HTML-documentation Style and JavaScript Files	none
Dream		Editor	proprietary
dtoa		Conversion between decimal and binary format of floating point numbers	MIT-like
EC Software Help Suite		Help Viewer	proprietary
Eigen	3.3.90	C++ template library for linear algebra	MPL 2.0
et-xmlfile	1.1.0	Creation of XML files	MIT
f2clib		Runtime library for programs converted with f2c	unnamed
Fast Memory Manager	4.73	A fast replacement memory manager	MPL 1.1
finlib		Models from A. Consiglio, S.S. Nielsen and S.A. Zenios, Practical Financial Optimization: A library of GAMS models, John Wiley, UK, 2009	none
FMT	6.1.2	C++ string formatting library	MIT
freetds	1.3.18	Implementation of Sybase's DB-, CT-, and ODBC libraries	LGPL 2.0
GAMS-Kestrel		Kestrel solver for remote solver execution	MIT
GAMSCHK	1.3	A system for examining the structure and solution properties of GAMS model instances	proprietary
GCC	13.2.0	C++ and Fortran compiler runtime libraries	GPL 3.0 + exception

Software	Version	Description	License
GKLib	METIS-v5.1.1-DistDGL-0.5	Helper Routines and Frameworks used by METIS	Apache 2.0
GNU Awk	3.1.0	Pattern Scanning and Processing Language	GPL 2
GNU diffutils	2.7	Find Differences between Files	GPL 2
GNU fileutils	3.16	File Management Utilities	GPL 2
GNU grep	2.4	Search one or more input files for lines containing a match to a specified pattern	GPL 2
GNU Gzip	1.2.4	Data Compression	GPL 2
GNU Make	3.78.1	Control the generation of executables and other non-source files from source files	GPL 2
GNU sed	3.02	Non-interactive Command-line Text Editor	GPL 2
GNU shellutils	1.9.4 (1.13 for date)	Shell-manipulation Utilities	GPL 2
GNU Tar	1.12	Archiving Utility	GPL 2
GNU textutils	2.0 (2.1 for md5sum)	Shell Utilities for Text File Processing	GPL 2
GNU xargs	4.1	Build and Execute Command Lines from Standard Input	GPL 2
greenlet	3.0.3	Concurrent programming	MIT
Gurobi	11.0.1	Mixed-Integer Linear and Quadratic Programming Solver	proprietary
HiGHS	1.6.0 (c070f1253)	Mixed-Integer Linear Optimization Solver	MIT
HSL MA27	2003-03-19	Sparse Linear Equations System Solver	proprietary
HSL MA57	3.11.0	Sparse Linear Equations System Solver	proprietary
HSL MA86	1.6.0	Sparse Linear Equations System Solver	proprietary
HSL MA97	2.6.0	Sparse Linear Equations System Solver	proprietary
HSL MC19	1989-03-09	Scaling Linear Equation Systems	proprietary
HSL MC68	3.3.3	Elimination Ordering for Sparse Direct Solver	proprietary
HTMLHelpViewer		HTML Help Viewer	proprietary
HTMLLITE	7.7	HTML Viewer	MIT
Icono	1.3.0	Icon Pack in CSS	MIT

Software	Version	Description	License
Intel Compiler	2021.2.0	C/C++ and Fortran compiler runtime libraries	Intel EULA
Ipopt	3.14.14	Nonlinear Optimization Solver	EPL 2.0
Jinja2	3.1.3	Templating engine	BSD-3-Clause
JQuery	3.5.1	JavaScript Utilities Library	MIT
Json.NET	13.0.2	High-performance JSON framework for .NET	MIT
JSON.simple	1.1.1	A simple Java library for JSON process, read and write JSON data	Apache 2.0
Knitro	14.0.0	Mixed-Integer and Continuous Nonlinear Programming Solver	proprietary
krb5	1.20.1	Network authentication	MIT
libiconv	1.16	Conversion of character encodings	LGPL 2.1
LibYAML	0.2.2	A C library for parsing and emitting YAML	MIT
LINDO API	14.0	Stochastic and global MINLP solver	proprietary
Lunr	2.3.6	Search Engine in JavaScript	MIT
madCollection	1.6f	Collection of Delphi utility functions	proprietary
MarkupSafe	2.1.4	Escapes characters for safe use in HTML and XML	BSD-3-Clause
MathJax	3.2.2	JavaScript Display Engine for Mathematics	Apache 2.0
MC++	2.0.1	Toolkit for Bounding Factorable Functions	EPL 1.0
McCarl GAMS User Guide		A GAMS User Guide written by Bruce McCarl	public domain
METIS	5.2.1	Serial Graph Partitioning and Fill-reducing Matrix Ordering	Apache 2.0
MILES		MCP solver	proprietary
MINOS	5.6	Nonlinear Programming Solver	proprietary
MKL	2023.1 (2021.2 on Windows)	Math Functions optimized for Intel CPUs and GPUs	ISSL
MOSEK	10.1.28	Mixed-Integer Conic Programming Solver	proprietary
MPSGE		Mathematical Programming System for General Equilibrium	proprietary

Software	Version	Description	License
MSVC 2019	14.29.30133	Visual C/C++ compiler runtime libraries	MS EULA
MUMPS	5.6.2	Multifrontal massively parallel sparse direct solver	CeCILL-C 1.0
nauty	2.8.8	Graph canonical labeling and automorphism group computation	Apache 2.0
nlohmann::json	v3.11.2	JSON for Modern C++	MIT
noalib		Models from Neculai Andrei, Nonlinear Optimization Applications Using the GAMS Technology, Springer Optimization and Its Applications, Springer, edition 127, 2013	none
numpy	1.26.3	Scientific computing in Python	BSD-3-Clause
ODHeuristics	7.0.7	Primal Heuristics for Mixed-Integer Linear Optimization	proprietary
oneTBB	2021.11.0	Multithreading Library	Apache 2.0
oneTBB (Windows)	2021.2.0	Multithreading library	ISSL
OpenMP (Intel)		Library for high-level parallelism in Fortran and C/C++	BSD-3-Clause
OpenMP (LLVM)	16.0.5	Library for high-level parallelism in Fortran and C/C++	MIT
openpyxl	3.1.2	Read and write Excel 2010 xlsx/xlsm files	MIT
OpenSSL	3.0.12	Toolkit for General-Purpose Cryptography and Secure Communication	Apache 2.0
Osi	0.108.10	Abstract base class to a generic linear programming (LP) solver	EPL 2.0
pandas	2.1.4	Data analysis and manipulation in Python	BSD-3-Clause
PaPILO	2.1.4.0 (ee0677c4)	Parallel Presolve for Integer and Linear Optimization	LGPL 3.0
PATH	5.0.07	MCP solver	proprietary
PDQSort		Pattern-defeating quicksort	zlib
ply	3.11	lex and yacc implementation in Python	BSD-3-Clause
Prism	1.29.0	JavaScript Syntax Highlighter	MIT

Software	Version	Description	License
psoptlib		Models from Alireza Soroudi, Power system optimization modeling in GAMS, Springer, 2017	none
psycopg2-binary	2.9.9	PostgreSQL adapter	LGPL 3.0
pthreads-win32	2.9.1	POSIX threads library for Microsoft Windows	LGPL 2.1
PyExcelerate	0.10.0	Excel XLSX writing	BSD-2-Clause
pymssql	2.2.10 (2.2.8 on macOS on ARM64)	DB-API interface to Microsoft SQL Server	LGPL 2.1
PyMySQL	1.1.0	MySQL Driver	MIT
pyodbc	5.0.1	Access to ODBC databases	MIT
Python	3.12.1	Scripting language	PSF
python-dateutil	2.8.2	Extensions to the standard Python datetime module	Apache 2.0
pytz	2023.3.post1	World timezone definitions	MIT
pywin32	306	Python for Window Extensions	BSD-3-Clause
PyYAML	6.0.1	YAML parser and emitter	MIT
Qt	6.5.1	Toolkit for creating cross-platform graphical user interfaces and applications	LGPL 3.0
quadMINOS	5.6	Quad-Precision Nonlinear Programming Solver	proprietary
Rave Reports	5.1.4	Components for database reporting	proprietary
sassy	1.1	Preprocessor for symmetry detection	MIT
SBB		Mixed-Integer Nonlinear Programming Solver	proprietary
SCENRED	05/08/2002	Scenario tree reduction tool for Stochastic Programming	proprietary
SCENRED 2	12/12/2008	Scenario tree construction and reduction tool for Stochastic Programming	proprietary
SCIP	8.1.0 (0dff9f8a3e)	Constraint Integer Programming Framework	Apache 2.0
scipy	1.11.4	Fundamental algorithms for scientific computing	BSD-3-Clause
SendMail		Send e-mails	proprietary

Software	Version	Description	License
SHOT	1.1.0 (424b5e48)	Mixed-Integer Nonlinear Programming Solver	EPL 2.0
six	1.16.0	Python 2 and 3 compatibility utilities	MIT
SKA	2c46874	Hashtables	Boost 1.0
SNOPT	7.7.7	Nonlinear Programming Solver	proprietary
SNOPT (in ANTIGONE)	5.3-5	Nonlinear Programming Solver	proprietary
SoPlex	6.0.4 (5a9b664d)	Linear Programming Solver	Apache 2.0
SPDLog	1.8.2	C++ Logging Library	MIT
SQLAlchemy	1.4.51	Database Abstraction Library	MIT
sqlalchemy-access	1.1.4	MS Access for SQLAlchemy	MIT
sqlite	3.8.5	SQL database engine	public domain
TeeTree	2.0	Diagram and FlowChart Controls	proprietary
TFileDrag		Support dropping files from explorer onto a Delphi form	proprietary
TFolderDialog		Wrap Shell32 dialog 'Browse For Folder' into a Delphi component	LGPL 2.0
TinyCThread	1.2	C11 Threads API implementation	zlib
TinyExpr	4dfb202c	Tiny recursive descent parser and evaluation engine	zlib
TinyXML	2.6.2	XML parser	zlib
TinyXML2	7.1.0	C++ XML parser	zlib
TOneInstance	1.02	Prevent application to be started more than once	freeware
TStringAlignGrid	2.0	Text alignment in Delphi forms	freeware
TurboPower Orpheus	4.05	UI toolkit for Borland Delphi	MPL 1.1
TurboPower ShellShock	1.02	Customize applications	MPL 1.1
TurboPower SysTools	4.02	Library of utility routines & classes for Borland Delphi	MPL 1.1
tzdata	2023.4	Compiled IANA time zone database	Apache 2.0
unixodbc	2.3.11	ODBC for Unix platforms	LGPL 2.1

Software	Version	Description	License
UnZip	6.0-28 (6.00 on Windows)	Decompression tool	Info-ZIP
urllib3	2.1.0	HTTP library	MIT
Virtual Treeview	4.8.6	Treeview control	MPL 1.1
XLSReadWriteII	5.20.30	Excel Reader and Writer	proprietary
XlsxWriter	3.1.9	Creation of Excel XLSX files	BSD-2-Clause
Xpress	41.01.03	Mixed-Integer Linear and Nonlinear Programming Solver	proprietary
yaml-cpp	0.7.0	A YAML parser and emitter in C++	MIT
Zip	3.0-13 (3.1c02 BETA on Windows)	Compression tool	Info-ZIP
zlib	1.3.1	Compression Library	zlib
zstr	1.0.5	C++ Zlib Wrapper	MIT

Index

Bibliography

[1] T. Achterberg. *Constraint Integer Programming*. PhD thesis, Technische Universität Berlin, 2007. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-11129.

[2] T. Achterberg. SCIP: Solving Constraint Integer Programs. *Mathematical Programming Computations*, 1(1):1–41, 2009. URL http://doi.org/10.1007/s12532-008-0001-1.

[3] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33 (1):42–54, 2005. URL http://doi.org/10.1016/j.orl.2004.04.002.

[4] T. Achterberg, T. Berthold, T. Koch, and K. Wolter. Constraint integer programming: A new approach to integrate CP and MIP. In L. Perron and M.A. Trick, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008*, volume 5015 of *LNCS*, pages 6–20. Springer, 2008. URL http://doi.org/10.1007/978-3-540-68155-7_4.

[5] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. A global optimization method, αBB, for general twice differentiable NLPs – II. Implementation and computational results. *Computers & Chemical Engineering*, 22(9):1159–1179, 1998. URL http://doi.org/10.1016/S0098-1354(98)00218-X.

[6] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice differentiable NLPs – I. Theoretical advances. *Computers & Chemical Engineering*, 22(9):1137–1158, 1998. URL http://doi.org/10.1016/S0098-1354(98)00027-1.

[7] S. Ahmed, M. Tawarmalani, and N. V. Sahinidis. A finite branch-and-bound algorithm for two-stage stochastic integer programs. *Mathematical Programming*, 100(2):355–377, 2004. URL http://doi.org/10.1007/s10107-003-0475-6.

[8] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1): 15–24, 2001. URL http://doi.org/10.1137/S0895479899358194.

[9] P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing*, 32(2):136–156, 2006. URL http://doi.org/10.1016/j.parco.2005.07.004.

[10] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB: A global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7(4):337–363, 1995. URL http://doi.org/10.1007/BF01099647.

[11] C. Audet, P. Hansen, B. Jaumard, and G. Savard. A branch and cut algorithm for nonconvex quadratically constrained quadratic programming. *Mathematical Programming*, 87(1):131–152, 2000. URL http://doi.org/10.1007/s101079900106.

[12] X. Bao, N. V. Sahinidis, and M. Tawarmalani. Multiterm polyhedral relaxations for nonconvex, quadratically-constrained quadratic programs. *Optimization Methods and Software*, 24(4-5):485–504, 2009. URL http://doi.org/10.1080/10556780902883184.

[13] X. Bao, N. V. Sahinidis, and M. Tawarmalani. Semidefinite relaxations for quadratically constrained quadratic programming: A review and comparisons. *Mathematical Programming*, 129(1):129–157, 2011. URL http://doi.org/10.1007/s10107-011-0462-2.

[14] X. Bao, A. Khajavirad, N. V. Sahinidis, and M. Tawarmalani. Global optimization of nonconvex problems with multilinear intermediates. *Mathematical Programming Computation*, 7(1):1–37, 2015. ISSN 1867-2949. URL http://doi.org/10.1007/s12532-014-0073-z.

[15] J. F. Bard. *Practical bilevel optimization: Algorithms and applications*, volume 30 of *Nonconvex optimization and its applications*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1998. URL http://doi.org/10.1007/978-1-4757-2836-1.

[16] R. H. Bartels. A stabilization of the simplex method. *Numerische Mathematik*, 16(5):414–434, 1971. URL http://doi.org/10.1007/BF02169151.

[17] R. H. Bartels and G. H. Golub. The simplex method of linear programming using the LU decomposition. *Communications of the ACM*, 12(5):266–268, 1969. URL http://doi.org/10.1145/362946.362974.

[18] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4–5):597–634, 2009. URL http://doi.org/10.1080/10556780903087124.

[19] D. E. Bernal, S. Vigerske, F. Trespalacios, and I. E. Grossmann. Improving the performance of DICOPT in convex MINLP problems using a feasibility pump, 2017. URL http://www.optimization-online.org/DB_HTML/2017/08/6171.html.

[20] T. Berthold. *Heuristic algorithms in global MINLP solvers*. PhD thesis, TU Berlin, 2014.

[21] T. Berthold, S. Heinz, and S. Vigerske. Extending a CIP framework to solve MIQCPs. In Jon Lee and Sven Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 427–444. Springer, 2012. URL http://doi.org/10.1007/978-1-4614-1927-3_15.

[22] Ksenia Bestuzheva, Mathieu Besançon, Wei-Kun Chen, Antonia Chmiela, Tim Donkiewicz, Jasper van Doornmalen, Leon Eifler, Oliver Gaul, Gerald Gamrath, Ambros Gleixner, Leona Gottwald, Christoph Graczyk, Katrin Halbig, Alexander Hoen, Christopher Hojny, Rolf van der Hulst, Thorsten Koch, Marco Lübbecke, Stephen J. Maher, Frederic Matter, Erik Mühmer, Benjamin Müller, Marc E. Pfetsch, Daniel Rehfeldt, Steffan Schlein, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Boro Sofranac, Mark Turner, Stefan Vigerske, Fabian Wegscheider, Philipp Wellner, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 8.0. ZIB Report 21-41, Zuse Institute Berlin, 2021. URL https://nbn-resolving.org/urn:nbn:de:0297-zib-85309.

[23] L. T. Biegler. *Nonlinear Programming: Concepts, Algorithms and Applications to Chemical Processes*. MOS-SIAM Series on Optimization. SIAM, Philadelphia, 2010. URL http://doi.org/10.1137/1.9780898719383.

[24] S. C. Billups. *Algorithms for Complementarity Problems and Generalized Equations*. PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, 1995. URL http://pages.cs.wisc.edu/~ferris/theses/StephenBillups.pdf.

[25] S. C. Billups. Improving the robustness of descent-based methods for semismooth equations using proximal perturbations. *Mathematical Programming*, 87(1):153–175, 2000. URL http://doi.org/10.1007/s101079900105.

[26] S. C. Billups and M. C. Ferris. QPCOMP: A quadratic program based solver for mixed complementarity problems. *Mathematical Programming*, 76(3):533–562, 1997. URL http://doi.org/10.1007/BF02614397.

[27] A. Bompadre and A. Mitsos. Convergence rate of McCormick relaxations. *Journal of Global Optimization*, 52(1):1–28, 2011. URL http://doi.org/10.1007/s10898-011-9685-2.

[28] P. Bonami, G. Cornuéjols, A. Lodi, and F. Margot. A feasibility pump for mixed integer nonlinear programs. *Mathematical Programming*, 119(2):331–352, 2009. URL http://doi.org/10.1007/s10107-008-0212-2.

[29] J. Bracken and J. T. McGill. Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44, 1973. URL http://doi.org/10.1287/opre.21.1.37.

[30] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 1999. URL http://doi.org/10.1137/S1052623497325107.

[31] R. H. Byrd, J. C. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000. URL http://doi.org/10.1007/PL00011391.

[32] R. H. Byrd, J. Nocedal, and R. A. Waltz. Feasible interior methods using slacks for nonlinear optimization. *Computational Optimization and Applications*, 26(1):35–61, 2003. URL http://doi.org/10.1023/A:1025136421370.

[33] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Mathematical Programming, Series B*, 100(1):27–48, 2004. URL http://doi.org/10.1007/s10107-003-0485-4.

[34] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. On the convergence of successive linear-quadratic programming algorithms. *SIAM Journal on Optimization*, 16(2):471–489, 2005. URL http://doi.org/10.1137/S1052623403426532.

[35] R. M. Chamberlain, M. J. D. Powell, and C. Lemaréchal. *The watchdog technique for forcing convergence in algorithms for constrained optimization*, volume 16 of *Mathematical Programming Studies*, pages 1–17. Springer, Berlin, Heidelberg, 1982. URL http://doi.org/10.1007/BFb0120945.

[36] Y. Chang and N. V. Sahinidis. Global optimization in stabilizing controller design. *Journal of Global Optimization*, 38(4):509–526, 2007. URL http://doi.org/10.1007/s10898-006-9092-2.

[37] V. Chvátal. *Linear Programming*. W. H. Freeman and Compan, New York, 1983.

[38] A. R. Conn. Constrained optimization using a nondifferentiable penalty function. *SIAM Journal on Numerical Analysis*, 10(4):760–784, 1973. URL http://doi.org/10.1137/0710063.

[39] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and its Applications*, 1(1):103–125, 1968. URL http://doi.org/10.1016/0024-3795(68)90052-9.

[40] R. W. Cottle, J. S. Pang, and R. E. Stone. *The Linear Complementarity Problem*. Academic Press, Boston, 1992.

[41] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.

[42] S. Devarajan, J. D. Lewis, and S. Robinson. Policy lessons from trade-focused, two-sector models. *Journal of Policy Modeling*, 12(4):625–657, 1990. URL http://doi.org/10.1016/0161-8938(90)90002-V.

[43] Paul Dierckx. *Curve and Surface Fitting with Splines*. Oxford University Press, 1993. ISBN 0-19-853441-8.

[44] S. P. Dirkse. *Robust Solution of Mixed Complementarity Problems*. PhD thesis, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1994. URL ftp://ftp.cs.wisc.edu/math-prog/tech-reports/94-12.ps.

[45] S. P. Dirkse and M. C. Ferris. MCPLIB: a collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5(4):319–345, 1995. URL http://doi.org/10.1080/10556789508805619.

[46] S. P. Dirkse and M. C. Ferris. A path search damped Newton method for computing general equilibria. *Annals of Operations Research*, 68(2):211–232, 1996. URL http://doi.org/10.1007/BF02209613.

[47] S. P. Dirkse and M. C. Ferris. Crash techniques for large-scale complementarity problems. In M. C. Ferris and J. S. Pang, editors, *Complementarity and Variational Problems: State of the Art*, pages 40–61. SIAM Publications, 1997.

[48] S. P. Dirkse and M. C. Ferris. Traffic modeling and variational inequalities using GAMS. In K. Tanczos Ph. L. Toint, M. Labbe and G. Laporte, editors, *Operations Research and Decision Aid Methodologies in Traffic and Transportation Management*, volume 166, pages 136–163. NATO ASI Series F, Philadelphia, Pennsylvania, 1998. URL http://doi.org/10.1007/978-3-662-03514-6_6.

[49] F. Domes and A. Neumaier. Constraint propagation on quadratic constraints. *Constraints*, 15(3): 404–429, 2010. URL http://doi.org/10.1007/s10601-009-9076-1.

[50] F. Domes and A. Neumaier. Rigorous enclosures of ellipsoids and directed cholesky factorizations. *SIAM Journal on Matrix Analysis and Applications*, 32(1):262–285, 2011. URL http://doi.org/10.1137/090778110.

[51] M. C. Dorneich and N. V. Sahinidis. Global optimization algorithms for chip layout and compaction. *Engineering Optimization*, 25(2):131–154, 1995. URL http://doi.org/10.1080/03052159508941259.

[52] Marco A. Duran and Ignacio E. Grossmann. An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3):307–339, 1986. URL http://doi.org/10.1007/BF02592064.

[53] Marco A. Duran and Ignacio E. Grossmann. A mixed-integer nonlinear programming algorithm for process systems synthesis. *AIChE Journal*, 32(4):592–606, 1986. URL http://doi.org/10.1002/aic.690320408.

[54] S. K. Eldersveld. *Large-scale sequential quadratic programming algorithms*. PhD thesis, Department of Operations Research, Stanford University, Stanford, CA, 1991.

[55] R. Ericson and A. Pakes. Markov perfect industry dynamics: A framework for empirical analysis. *The Review of Economic Studies*, 62(1):53–82, 1995. URL http://doi.org/10.2307/2297841.

[56] M. C. Ferris and S. Lucidi. Nonmonotone stabilization methods for nonlinear equations. *Journal of Optimization Theory and Applications*, 81(1):53–71, 1994. URL http://doi.org/10.1007/BF02190313.

[57] M. C. Ferris and T. S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 12(1-3):207–227, 1999. URL http://doi.org/10.1023/A:1008636318275.

[58] M. C. Ferris and T. S. Munson. Preprocessing complementarity problems. In Michael C. Ferris, Olvi L. Mangasarian, and Jong-Shi Pang, editors, *Complementarity: Applications, Algorithms and Extensions*, pages 143–164. Springer US, Boston, MA, 2001. URL http://doi.org/10.1007/978-1-4757-3279-5_7.

[59] M. C. Ferris and J. S. Pang, editors. *Complementarity and Variational Problems: State of the Art*. SIAM Publications, Philadelphia, Pennsylvania, 1997.

[60] M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39(4):669–713, 1997. URL http://doi.org/10.1137/S0036144595285963.

[61] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming*, 86(3):475–497, 1999. URL http://doi.org/10.1007/s101070050101.

[62] M. C. Ferris, A. Meeraus, and T. F. Rutherford. Computing Wardropian equilibria in a complementarity framework. *Optimization Methods and Software*, 10(5):669–685, 1999. URL http://doi.org/10.1080/10556789908805733.

[63] A. Fischer. A special Newton–type optimization method. *Optimization*, 24(3-4):269–284, 1992. URL http://doi.org/10.1080/02331939208843795.

[64] R. Fletcher. An ℓ_1 penalty method for nonlinear constraints. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 26–40, Philadelphia, 1985. SIAM.

[65] C. A. Floudas. *Nonlinear and Mixed-Integer Optimization: Fundamentals and Applications*. Oxford University Press, New York, NY, 1995. ISBN 0195100565.

[66] C. A. Floudas. *Deterministic Global Optimization: Theory, Algorithms and Applications*, volume 37 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, Dordrecht, Netherlands, 2000. ISBN 978-0-7923-6014-8.

[67] C. A. Floudas and C. E. Gounaris. A review of recent advances in global optimization. *Journal of Global Optimization*, 45(1):3–38, 2009. URL http://doi.org/10.1007/s10898-008-9332-8.

[68] C. A. Floudas, I. G. Akrotirianakis, S. Caratzoulas, C. A. Meyer, and J. Kallrath. Global optimization in the 21st century: Advances and challenges. *Computers & Chemical Engineering*, 29(6):1185–1202, 2005. URL http://doi.org/10.1016/j.compchemeng.2005.02.006.

[69] R. Fourer. Solving staircase linear programs by the simplex method – 1: Inversion. *Mathematical Programming*, 23(1):274–313, 1982. URL http://doi.org/10.1007/BF01583795.

[70] Robert Fourer, Jun Ma, and Kipp Martin. OSiL: An instance language for optimization. *Computational Optimization and Applications*, 45(1):181–203, 2010. URL http://doi.org/10.1007/s10589-008-9169-6.

[71] J. Fourtany-Amat and B. McCarl. A representation and economic interpretation of a two-level programming problem. *Journal of the Operational Research Society*, 32(9):783–792, 1981. URL http://doi.org/10.1057/jors.1981.156.

[72] K. Furman and I. P. Androulakis. A novel MINLP-based representation of the original complex model for predicting gasoline emissions. *Computers & Chemical Engineering*, 32(12):2857–2876, 2008. URL http://doi.org/10.1016/j.compchemeng.2008.02.002.

[73] K. Furman, N. Sawaya, and I. E. Grossmann. A computationally useful algebraic representation of nonlinear disjunctive convex sets using the perspective function. E-Print 5544, Optimization Online, 2016. URL http://www.optimization-online.org/DB_HTML/2016/07/5544.html.

[74] Gerald Gamrath, Tobias Fischer, Tristan Gally, Ambros M. Gleixner, Gregor Hendel, Thorsten Koch, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Stefan Vigerske, Dieter Weninger, Michael Winkler, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 3.2. ZIB Report 15-60, Zuse Institute Berlin, 2016. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-57675.

[75] Gerald Gamrath, Daniel Anderson, Ksenia Bestuzheva, Wei-Kun Chen, Leon Eifler, Maxime Gasse, Patrick Gemander, Ambros Gleixner, Leona Gottwald, Katrin Halbig, Gregor Hendel, Christopher Hojny, Thorsten Koch, Pierre Le Bodic, Stephen J. Maher, Frederic Matter, Matthias Miltenberger, Erik Mühmer, Benjamin Müller, Marc Pfetsch, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Christine Tawfik, Stefan Vigerske, Fabian Wegscheider, Dieter Weninger, and Jakob Witzig. The SCIP Optimization Suite 7.0. ZIB Report 20-10, Zuse Institute Berlin, 2020. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-78023.

[76] E. P. Gatzke, J. E. Tolsma, and P. I. Barton. Construction of convex relaxations using automated code generation techniques. *Optimization and Engineering*, 3(3):305–326, 2002. URL http://doi.org/10.1023/A:1021095211251.

[77] A. M. Geoffrion. Elements of Large-Scale Mathematical Programming – I. Concepts. *Management Science*, 16(11):652–675, 1970. URL http://doi.org/10.1287/mnsc.16.11.652.

[78] V. Ghildyal. Design and development of a global optimization system. Master's thesis, Department of Mechanical & Industrial Engineering, University of Illinois, Urbana, IL, 1997.

[79] V. Ghildyal and N. V. Sahinidis. Solving global optimization problems with BARON. In A. Migdalas, P. M. Pardalos, and P. Värbrand, editors, *From Local to Global Optimization*, volume 53 of *Nonconvex Optimization and Its Applications*, chapter 10, pages 205–230. Springer, 2001. URL http://doi.org/10.1007/978-1-4757-5284-7_10.

[80] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Two step-length algorithms for numerical optimization. Technical Report SOL 79–25, Department of Operations Research, Stanford University, Stanford, California, 1979.

[81] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. User's guide for NPSOL (Version 4.0): a Fortran package for nonlinear programming. Technical Report SOL 86–2, Department of Operations Research, Stanford University, Stanford, CA, 1986.

[82] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra and its Applications*, 88-89:239–270, 1987. URL http://doi.org/10.1016/0024-3795(87)90112-1.

[83] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. A practical anti-cycling procedure for linearly constrained optimization. *Mathematical Programming*, 45(1-3):437–474, 1989. URL http://doi.org/10.1007/BF01589114.

[84] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Some theoretical properties of an augmented Lagrangian merit function. In P. M. Pardalos, editor, *Advances in Optimization and Parallel Computing*, pages 101–128. Elsevier Science Inc, 1992.

[85] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002. URL http://doi.org/10.1137/S1052623499350013.

[86] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131, 2005. URL http://doi.org/10.1137/S0036144504446096.

[87] P. E. Gill, W. Murray, and M. A. Saunders. User's guide for SQOPT version 7: Software for large-scale linear and quadratic programming. Numerical analysis report, Department of Mathematics, University of California, San Diego, La Jolla, CA, 2006.

[88] Ambros Gleixner, Leon Eifler, Tristan Gally, Gerald Gamrath, Patrick Gemander, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Stefan Vigerske, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 5.0. ZIB Report 17-61, Zuse Institute Berlin, 2017. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-66297.

[89] Ambros Gleixner, Michael Bastubbe, Leon Eifler, Tristan Gally, Gerald Gamrath, Robert Lion Gottwald, Gregor Hendel, Christopher Hojny, Thorsten Koch, Marco E. Lübbecke, Stephen J. Maher, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Franziska Schlösser, Christoph Schubert, Felipe Serrano, Yuji Shinano, Jan Merlin Viernickel, Matthias Walter, Fabian Wegscheider, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 6.0. ZIB-Report 18-26, Zuse Institute Berlin, 2018. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-69361.

[90] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986. URL http://doi.org/10.1137/0723046.

[91] L. Grippo, F. Lampariello, and S. Lucidi. A class of nonmonotone stabilization methods in unconstrained optimization. *Numerische Mathematik*, 59(1):779–805, 1991. URL http://doi.org/10.1007/BF01385810.

[92] Computational Mathematics Group. HSL 2002 – a catalogue of subroutines. Technical report, STFC Rutherford Appleton Laboratory, Harwell Oxford, 2002.

[93] R. A. Gutiérrez and N. V. Sahinidis. A branch-and-bound approach for machine selection in just-in-time manufacturing systems. *International Journal of Production Research*, 34(3):797–818, 1996. URL http://doi.org/10.1080/00207549608904935.

[94] P. T. Harker and J. S. Pang. Finite-dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms and applications. *Mathematical Programming*, 48(1-3):161–220, 1990. URL http://doi.org/10.1007/BF01582255.

[95] G. W. Harrison, T. F. Rutherford, and D. Tarr. Quantifying the Uruguay round. *The Economic Journal*, 107(444):1405–1430, 1997. URL http://doi.org/10.1111/j.1468-0297.1997.tb00055.x.

[96] Alain Haurie and Jacek B. Krawczyk. Optimal charges on river effluent from lumped and distributed sources environmental modeling and assessment. *Environmental Modeling & Assessment*, 2(3): 177–189, 1997. URL http://doi.org/10.1023/A:1019049008557.

[97] J. Huang and J. S. Pang. Option pricing and linear complementarity. *Journal of Computational Finance*, 2(1):31–60, 1998. URL http://doi.org/10.21314/JCF.1998.018.

[98] Q. Huangfu and J. A. J. Hall. Parallelizing the dual revised simplex method. *Mathematical Programming Computation*, 10(1):119–142, 2018. URL http://doi.org/10.1007/s12532-017-0130-5.

[99] P. J. Huber. *Robust statistics*. John Wiley & Sons, New York, 1981.

[100] G. Infanger. *DECIS User's Guide*. 1590 Escondido Way, Belmont, CA 94002, 1997.

[101] N. H. Josephy. Newton's method for generalized equations. Technical Summary Report 1965, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1979.

[102] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, 1939.

[103] George Karypis and Vipin Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999. URL http://doi.org/10.1137/S1064827595287997.

[104] Yoshiaki Kawajir, Carl Laird, and Andreas Wächter. *Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt*, 2087 edition, February 2012. https://github.com/coin-or/Ipopt.

[105] J. E. Kelley. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial and Applied Mathematics*, 8(4):703–712, 1960. URL http://doi.org/10.1137/0108053.

[106] A. Khajavirad and N. V. Sahinidis. Convex envelopes of products of convex and component-wise concave functions. *Journal of Global Optimization*, 52(3):391–409, 2012. URL http://doi.org/10.1007/s10898-011-9747-5.

[107] A. Khajavirad and N. V. Sahinidis. Convex envelopes generated from finitely many compact convex sets. *Mathematical Programming*, 137(1-2):371–408, 2013. URL http://doi.org/10.1007/s10107-011-0496-5.

[108] A. Khajavirad, J. J. Michalek, and N. V. Sahinidis. Relaxations of factorable functions with convex-transformable intermediates. *Mathematical Programming*, 144(1-2):107–140, 2014. URL http://doi.org/10.1007/s10107-012-0618-8.

[109] Y. Kim and M.C. Ferris. Solving equilibrium problems using extended mathematical programming, 2017.

[110] G. R. Kocis and I. E. Grossmann. Relaxation strategy for the structural optimization of process flowsheets. *Industrial & Engineering Chemistry Research*, 26(9):1869–1880, 1987. URL http://doi.org/10.1021/ie00069a026.

[111] Jacek B. Krawczyk and Stanislav Uryasev. Relaxation algorithms to find nash equilibria with economic applications. *Environmental Modeling & Assessment*, 5(1):63–73, 2000. URL http://doi.org/10.1023/A:1019097208499.

[112] J. Kronqvist, A. Lundell, and T. Westerlund. The extended supporting hyperplane algorithm for convex mixed-integer nonlinear programming. *Journal of Global Optimization*, 64(2):249–272, 2016. doi: 10.1007/s10898-015-0322-3.

[113] J. Kronqvist, D. E. Bernal, A. Lundell, and I. E. Grossmann. A review and comparison of solvers for convex minlp. *Optimization and Engineering*, 20(2):397–455, 2019. doi: 10.1007/s11081-018-9411-8.

[114] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley and Los Angeles, 1951. University of California Press.

[115] B. Lanz and T. F. Rutherford. Gtapingams: Multiregional and small open economy models. *Journal of Global Economic Analysis*, 1(2):1–77, 2016. URL http://doi.org/10.21642/JGEA.010201AF.

[116] Y. Lebbah, C. Michel, and M. Rueher. A rigorous global filtering algorithm for quadratic constraints. *Constraints*, 10(1):47–65, 2005. URL http://doi.org/10.1007/s10601-004-5307-7.

[117] C. E. Lemke and J. T. Howson, Jr. Equilibrium points of bimatrix games. *Journal of the Society for Industrial and Applied Mathematics*, 12(2):413–423, 1964. URL http://doi.org/10.1137/0112033.

[118] L. Liberti and C. C. Pantelides. Convex envelopes of monomials of odd degree. *Journal of Global Optimization*, 25(2):157–168, 2003. URL http://doi.org/10.1023/A:1021924706467.

[119] L. Liberti and C. C. Pantelides. An exact reformulation algorithm for large nonconvex NLPs involving bilinear terms. *Journal of Global Optimization*, 36(2):161–189, 2006. URL http://doi.org/10.1007/s10898-006-9005-4.

[120] M.-L. Liu, N. V. Sahinidis, and J. P. Shectman. Planning of chemical process networks via global concave minimization. In I. E. Grossmann, editor, *Global Optimization in Engineering Design*, volume 9 of *Nonconvex Optimization and Its Applications*, pages 195–230. Springer, 1996. URL http://doi.org/10.1007/978-1-4757-5331-8_7.

[121] A. Lundell and J. Kronqvist. On solving nonconvex MINLP problems with SHOT. In Le Thi H., Le H., and Pham Dinh T., editors, *Optimization of Complex Systems: Theory, Models, Algorithms and Applications*, volume 991 of *Advances in Intelligent Systems and Computing*. Springer, Cham., 2019.

[122] A. Lundell and J. Kronqvist. Polyhedral approximation strategies for nonconvex mixed-integer nonlinear programming in SHOT. *Journal of Global Optimization*, 2021. URL https://doi.org/10.1007/s10898-021-01006-1.

[123] A. Lundell and T. Westerlund. Convex underestimation strategies for signomial functions. *Optimization Methods and Software*, 24(4-5):505–522, 2009. URL http://doi.org/10.1080/10556780802702278.

[124] A. Lundell and T. Westerlund. Global optimization of mixed-integer signomial programming problems. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes in Mathematics and its Applications*, pages 349–369. Springer New York, 2012. URL http://doi.org/10.1007/978-1-4614-1927-3_12.

[125] A. Lundell, J. Westerlund, and T. Westerlund. Some transformation techniques with applications in global optimization. *Journal of Global Optimization*, 43(2-3):391–405, 2009. URL http://doi.org/10.1007/s10898-007-9223-4.

[126] A. Lundell, J. Kronqvist, and T. Westerlund. The supporting hyperplane optimization toolkit: A polyhedral outer approximation based convex minlp solver utilizing a single branching tree approach. Technical report, Optimization Online, 2018. URL http://www.optimization-online.org/DB_FILE/2018/06/6680.pdf.

[127] Stephen J. Maher, Tobias Fischer, Tristan Gally, Gerald Gamrath, Ambros Gleixner, Robert Lion Gottwald, Gregor Hendel, Thorsten Koch, Marco E. Lübbecke, Matthias Miltenberger, Benjamin Müller, Marc E. Pfetsch, Christian Puchert, Daniel Rehfeldt, Sebastian Schenker, Robert Schwarz, Felipe Serrano, Yuji Shinano, Dieter Weninger, Jonas T. Witt, and Jakob Witzig. The SCIP Optimization Suite 4.0. ZIB Report 17-12, Zuse Institute Berlin, 2017. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-62170.

[128] C. D. Maranas and C. A. Floudas. Finding all solutions of nonlinearly constrained systems of equations. *Journal of Global Optimization*, 7(2):143–182, 1995. URL http://doi.org/10.1007/BF01097059.

[129] C. D. Maranas and C. A. Floudas. Global optimization in generalized geometric programming. *Computers & Chemical Engineering*, 21(4):351–369, 1997. URL http://doi.org/10.1016/S0098-1354(96)00282-7.

[130] J. Markusen and T. F. Rutherford. MPSGE: A user's guide, February 2004. URL https://www.scribd.com/document/212467121/MPSGE-Users-Guide. Lecture Notes Prepared for the UNSW Workshop.

[131] L. Mathiesen. Computation of economic equilibria by a sequence of linear complementarity problems. In Alan S. Manne, editor, *Economic Equilibrium: Model Formulation and Solution*, volume 23 of *Mathematical Programming Studies*, pages 144–162. Springer, Berlin, Heidelberg, 1985. URL http://doi.org/10.1007/BFb0121030.

[132] L. Mathiesen. An algorithm based on a sequence of linear complementarity problems applied to a Walrasian equilibrium model: An example. *Mathematical Programming*, 37(1):1–18, 1987. URL http://doi.org/10.1007/BF02591680.

[133] C. A. Meyer and C. A. Floudas. Convex envelopes for edge-concave functions. *Mathematical Programming*, 103(2):207–224, 2005. URL http://doi.org/10.1007/s10107-005-0580-9.

[134] R. Misener and C. A. Floudas. Global optimization of mixed-integer quadratically-constrained quadratic programs (MIQCQP) through piecewise-linear and edge-concave relaxations. *Mathematical Programming*, 136(1):155–182, 2012. URL http://doi.org/10.1007/s10107-012-0555-6.

[135] R. Misener and C. A. Floudas. GloMIQO: Global Mixed-Integer Quadratic Optimizer. *Journal of Global Optimization*, 57(1):3–50, 2013. URL http://doi.org/10.1007/s10898-012-9874-7.

[136] R. Misener and C. A. Floudas. A framework for globally optimizing mixed-integer signomial programs. *Journal of Optimization Theory and Applications*, 161(3):905–932, 2014. URL http://doi.org/10.1007/s10957-013-0396-3.

[137] R. Misener and C. A. Floudas. ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. *Journal of Global Optimization*, 59(2-3):503–526, 2014. URL http://doi.org/10.1007/s10898-014-0166-2.

[138] R. Misener, J. B. Smadbeck, and C. A. Floudas. Dynamically generated cutting planes for mixed-integer quadratically constrained quadratic programs and their incorporation into GloMIQO 2. *Optimization Methods and Software*, 30(1):215–249, 2015. URL http://doi.org/10.1080/10556788.2014.916287.

[139] B. A. Murtagh and M. A. Saunders. Large-scale linearly constrained optimization. *Mathematical Programming*, 14(1):41–72, 1978. URL http://doi.org/10.1007/BF01588950.

[140] B. A. Murtagh and M. A. Saunders. A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints. In A. G. Buckley and J.-L. Goffin, editors, *Algorithms for Constrained Minimization of Smooth Nonlinear Functions*, volume 16 of *Mathematic Programming Studies*, pages 84–117. Springer, Berlin, Heidelberg, 1982. URL http://doi.org/10.1007/BFb0120949.

[141] B. A. Murtagh and M. A. Saunders. Minos 5.5 user's guide. Technical Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, CA, 1983. Revised 1998.

[142] B. A. Murtagh and M. A. Saunders. Minos user's guide. Technical Report SOL 83-20, Department of Operations Research, Stanford University, Stanford, CA, 1983.

[143] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999. URL http://doi.org/10.1007/978-0-387-40065-5.

[144] Jorge Nocedal, Andreas Wächter, and Richard A. Waltz. Adaptive barrier strategies for nonlinear interior methods. *SIAM Journal on Optimization*, 19(4):1674–1693, 2008. URL http://doi.org/10.1137/060649513.

[145] Ray Pörn and Tapio Westerlund. A cutting plane method for minimizing pseudo-convex functions in the mixed integer case. *Computers & Chemical Engineering*, 24(12):2655–2665, 2000. URL http://doi.org/10.1016/S0098-1354(00)00622-0.

[146] I. Quesada and I. E. Grossmann. A global optimization algorithm for linear fractional and bilinear programs. *Journal of Global Optimization*, 6(1):39–76, 1995. URL http://doi.org/10.1007/BF01106605.

[147] J. K. Reid. Fortran subroutines for handling sparse linear programming bases. Technical Report Report R8269, Atomic Energy Research Establishment, Harwell, England, 1976.

[148] J. K. Reid. A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases. *Mathematical Programming*, 24(1):55–69, 1982. URL http://doi.org/10.1007/BF01585094.

[149] L. M. Rios and N. V. Sahinidis. Portfolio optimization for wealth-dependent risk preferences. *Annals of Operations Research*, 177(1):63–90, 2010. URL http://doi.org/10.1007/s10479-009-0592-6.

[150] S. M. Robinson. A quadratically-convergent algorithm for general nonlinear programming problems. *Mathematical Programming*, 3(1):145–156, 1972. URL http://doi.org/10.1007/BF01584986.

[151] S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17(3):691–714, 1992. URL http://doi.org/10.1287/moor.17.3.691.

[152] R. T. Rockafellar. Linear-quadratic programming and optimal control. *SIAM Journal on Control and Optimization*, 25(3):781–814, 1987. URL http://doi.org/10.1137/0325045.

[153] R. T. Rockafellar. Lagrange multipliers and optimality. *SIAM Review*, 35(2):183–238, 1993. URL http://doi.org/10.1137/1035044.

[154] R. T. Rockafellar. Extended nonlinear programming. In Gianni Di Pillo and Franco Giannessi, editors, *Nonlinear Optimization and Related Topics*, volume 36 of *Applied Optimization*, pages 381–399. Springer, Boston, MA, 1999. URL http://doi.org/10.1007/978-1-4757-3226-9_20.

[155] T. F. Rutherford. Extensions of GAMS for complementarity problems arising in applied economic analysis. *Journal of Economic Dynamics and Control*, 19(8):1299–1324, 1995. URL http://doi.org/10.1016/0165-1889(94)00831-2.

[156] T. F. Rutherford. Applied general equilibrium modeling with mpsge as a gams subsystem: An overview of the modeling framework and syntax. *Computational Economics*, 14(1-2):1–46, 1999. URL http://doi.org/10.1023/A:1008655831209.

[157] H. S. Ryoo and N. V. Sahinidis. Global optimization of nonconvex NLPs and MINLPs with applications in process design. *Computers & Chemical Engineering*, 19(5):551–556, 1995. URL http://doi.org/10.1016/0098-1354(94)00097-2.

[158] H. S. Ryoo and N. V. Sahinidis. A branch-and-reduce approach to global optimization. *Journal of Global Optimization*, 8(2):107–138, 1996. URL http://doi.org/10.1007/BF00138689.

[159] H. S. Ryoo and N. V. Sahinidis. Analysis of bounds for multilinear functions. *Journal of Global Optimization*, 19(4):403–424, 2001. URL http://doi.org/10.1023/A:1011295715398.

[160] H. S. Ryoo and N. V. Sahinidis. Global optimization of multiplicative programs. *Journal of Global Optimization*, 26(4):387–418, 2003. URL http://doi.org/10.1023/A:1024700901538.

[161] N. V. Sahinidis. BARON: A general purpose global optimization software package. *Journal of Global Optimization*, 8(2):201–205, 1996. URL http://doi.org/10.1007/BF00138693.

[162] N. V. Sahinidis. Global optimization and constraint satisfaction: The branch-and-reduce approach. In Ch. Bliek, Ch. Jermann, and A. Neumaier, editors, *Global Optimization and Constraint Satisfaction*, volume 2861 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2003. URL http://doi.org/10.1007/978-3-540-39901-8_1.

[163] N. V. Sahinidis and M. Tawarmalani. Applications of global optimization to process and molecular design. *Computers & Chemical Engineering*, 24(9-10):2157–2169, 2000. URL http://doi.org/10.1016/S0098-1354(00)00583-4.

[164] N. V. Sahinidis and M. Tawarmalani. Accelerating branch-and-bound through a modeling language construct for relaxation-specific constraints. *Journal of Global Optimization*, 32(2):259–280, 2005. ISSN 0925-5001. URL http://doi.org/10.1007/s10898-004-2705-8.

[165] N. V. Sahinidis, M. Tawarmalani, and M. Yu. Design of alternative refrigerants via global optimization. *AIChE Journal*, 49(7):1761–1775, 2003. URL http://doi.org/10.1002/aic.690490714.

[166] Olaf Schenk and Klaus Gärtner. Solving unsymmetric sparse systems of linear equations with pardiso. *Journal of Future Generation Computer Systems*, 20(3):475–487, 2004. URL http://doi.org/10.1016/j.future.2003.07.011.

[167] Olaf Schenk and Klaus Gärtner. On fast factorization pivoting methods for sparse symmetric indefinite systems. *Electronic Transactions on Numerical Analysis*, 23:158–179, 2006.

[168] J. P. Shectman and N. V. Sahinidis. A finite algorithm for global minimization of separable concave programs. *Journal of Global Optimization*, 12(1):1–36, 1998. URL http://doi.org/10.1023/A:1008241411395.

[169] H. Sherali, E. Dalkiran, and L. Liberti. Reduced RLT representations for nonconvex polynomial programming problems. *Journal of Global Optimization*, 52(3):447–469, 2012. URL http://doi.org/10.1007/s10898-011-9757-3.

[170] H. D. Sherali and W. P. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*, volume 31 of *Nonconvex Optimization and Its Applications*. Springer, Boston, MA, 1999. URL http://doi.org/10.1007/978-1-4757-4388-3.

[171] H. D. Sherali and A. Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global Optimization*, 2(4):379–410, 1992. URL http://doi.org/10.1007/BF00122429.

[172] H. D. Sherali and C. H. Tuncbilek. A reformulation-convexification approach for solving nonconvex quadratic-programming problems. *Journal of Global Optimization*, 7(1):1–31, 1995. URL http://doi.org/10.1007/BF01100203.

[173] H. D. Sherali and C. H. Tuncbilek. New reformulation linearization/convexification relaxations for univariate and multivariate polynomial programming problems. *Operations Research Letters*, 21(1):1–9, 1997. URL http://doi.org/10.1016/S0167-6377(97)00013-8.

[174] Claus Still and Tapio Westerlund. Extended cutting plane algorithm. In C. A. Floudas and P. Pardalos, editors, *Encyclopedia of Optimization*, pages 593–601. Kluwer Academic Publishers, 2001. URL http://doi.org/10.1007/0-306-48332-7_126.

[175] F. Tardella. On a class of functions attaining their maximum at the vertices of a polyhedron. *Discrete Applied Mathematics*, 22(2):191–195, 1988/89. URL http://doi.org/10.1016/0166-218X(88)90093-5.

[176] F. Tardella. On the existence of polyhedral convex envelopes. In C. A. Floudas and P. M. Pardalos, editors, *Frontiers in Global Optimization*, volume 74 of *Nonconvex Optimization and Its Applications*, pages 563–573. Kluwer Academic Publishers, 2004. URL http://doi.org/10.1007/978-1-4613-0251-3_30.

[177] F. Tardella. Existence and sum decomposition of vertex polyhedral convex envelopes. *Optimization Letters*, 2(3):363–375, 2008. URL http://doi.org/10.1007/s11590-007-0065-2.

[178] M. Tawarmalani and N. V. Sahinidis. Semidefinite relaxations of fractional programs via novel convexification techniques. *Journal of Global Optimization*, 20(2):133–154, 2001. ISSN 0925-5001. URL http://doi.org/10.1023/A:1011233805045.

[179] M. Tawarmalani and N. V. Sahinidis. Convex extensions and envelopes of lower semi-continuous functions. *Mathematical Programming*, 93(2):247–263, 2002. URL http://doi.org/10.1007/s10107-002-0308-z.

[180] M. Tawarmalani and N. V. Sahinidis. *Convexification and Global Optimization in Continuous and Mixed-Integer Nonlinear Programming: Theory, Algorithms, Software, and Applications*, volume 65 of *Nonconvex Optimization and Its Applications*. Kluwer Academic Publishers, 2002. URL http://doi.org/10.1007/978-1-4757-3532-1.

[181] M. Tawarmalani and N. V. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Mathematical Programming*, 99(3):563–591, 2004. URL http://doi.org/10.1007/s10107-003-0467-6.

[182] M. Tawarmalani and N. V. Sahinidis. A polyhedral branch-and-cut approach to global optimization. *Mathematical Programming*, 103(2):225–249, 2005. URL http://doi.org/10.1007/s10107-005-0581-8.

[183] M. Tawarmalani, S. Ahmed, and N. V. Sahinidis. Global optimization of 0-1 hyperbolic programs. *Journal of Global Optimization*, 24(4):385–416, 2002. URL http://doi.org/10.1023/A:1021279918708.

[184] M. Tawarmalani, S. Ahmed, and N. V. Sahinidis. Product disaggregation in global optimization and relaxations of rational programs. *Optimization and Engineering*, 3(3):281–303, 2002. ISSN 1389-4420. URL http://doi.org/10.1023/A:1021043227181.

[185] P. Tseng. Growth behavior of a class of merit functions for the nonlinear complementarity problem. *Journal of Optimization Theory and Applications*, 89(1):17–37, 1996. URL http://doi.org/10.1007/BF02192639.

[186] M. Türkay and I. E. Grossmannn. Logic-based MINLP algorithms for the optimal synthesis of process networks. *Computers & Chemical Engineering*, 20(8):959–978, 1996. URL http://doi.org/10.1016/0098-1354(95)00219-7.

[187] J. G. VanAntwerp, R. D. Braatz, and N. V. Sahinidis. Globally optimal robust control for systems with time-varying nonlinear perturbations. *Computers & Chemical Engineering*, 21, Supplement: S125–S130, 1997. URL http://doi.org/10.1016/S0098-1354(97)87490-X.

[188] J. G. VanAntwerp, R. D. Braatz, and N. V. Sahinidis. Globally optimal robust process control. *Journal of Process Control*, 9(5):375–383, 1999. URL http://doi.org/10.1016/S0959-1524(99)00012-8.

[189] A. Vecchietti and I. E. Grossmannn. LOGMIP: a disjunctive 0-1 non-linear optimizer for process system models. *Computers & Chemical Engineering*, 23(4-5):555–565, 1999. URL http://doi.org/10.1016/S0098-1354(98)00293-2.

[190] A. Vecchietti, S. Lee, and I. E. Grossmannn. Modeling of discrete/continuous optimization problems: Characterizaton and formulations of disjunctions and their relaxations. *Computers & Chemical Engineering*, 27(3):433–448, 2003. URL http://doi.org/10.1016/S0098-1354(02)00220-X.

[191] Stefan Vigerske. *Decomposition of Multistage Stochastic Programs and a Constraint Integer Programming Approach to Mixed-Integer Nonlinear Programming*. PhD thesis, Humboldt-Universität zu Berlin, 2013. URL http://nbn-resolving.de/urn:nbn:de:kobv:11-100208240.

[192] Stefan Vigerske and Ambros Gleixner. SCIP: Global optimization of mixed-integer nonlinear programs in a branch-and-cut framework. ZIB Report 16-24, Zuse Institute Berlin, 2016. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-59377.

[193] J. Viswanathan and I. E. Grossmann. A combined penalty function and outer approximation method for minlp optimization. *Computers & Chemical Engineering*, 14(7):769–782, 1990. URL http://doi.org/10.1016/0098-1354(90)87085-4.

[194] Andreas Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, January 2002.

[195] Andreas Wächter and Lorenz T. Biegler. Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, 16(1):1–31, 2005. URL http://doi.org/10.1137/S1052623403426556.

[196] Andreas Wächter and Lorenz T. Biegler. Line search filter methods for nonlinear programming: Local convergence. *SIAM Journal on Optimization*, 16(1):32–48, 2005. URL http://doi.org/10.1137/S1052623403426544.

[197] Andreas Wächter and Lorenz T. Biegler. On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57, 2006. URL http://doi.org/10.1007/s10107-004-0559-y. http://github.com/coin-or/Ipopt.

[198] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming*, 107(3): 391–408, 2006. URL http://doi.org/10.1007/s10107-004-0560-5.

[199] Tapio Westerlund and Frank Petterson. An extended cutting plane method for solving convex MINLP problems. *Computers & Chemical Engineering*, 19(suppl.):131–136, 1995. URL http://doi.org/10.1016/0098-1354(95)87027-X.

[200] Tapio Westerlund and Ray Pörn. Solving pseudo-convex mixed integer optimization problems by cutting plane techniques. *Optimization and Engineering*, 3(3):253–280, 2002. URL http://doi.org/10.1023/A:1021091110342.

[201] Tapio Westerlund, Hans Skrifvars, Iiro Harjunkoski, and Ray Pörn. An extended cutting plane method for solving a class of non-convex minlp problems. *Computers & Chemical Engineering*, 22 (3):357–365, 1998. URL http://doi.org/10.1016/S0098-1354(97)00000-8.

[202] H. P. Williams. *Model Building in Mathematical Programming*. Wiley, 4th edition, 1999. ISBN 978-1-118-44333-0.

[203] P. Wolfe. The reduced gradient method. RAND Corporation, 1962.

[204] Kati Wolter. Implementation of cutting plane separators for mixed integer programs. Diploma thesis, Technische Universität Berlin, 2006.

[205] S. J. Wright. *Primal-Dual Interior-Point Methods*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1997. URL http://doi.org/10.1137/1.9781611971453.

[206] Roland Wunderling. *Paralleler und objektorientierter Simplex-Algorithmus*. PhD thesis, Technische Universität Berlin, 1996. URL http://nbn-resolving.de/urn:nbn:de:0297-zib-5386. http://soplex.zib.de.

[207] Maryam Yashtini and Alaeddin Malek. Solving complementarity and variational inequalities problems using neural networks. *Applied Mathematics*, 190(1):216–230, 2007. URL http://doi.org/10.1016/j.amc.2007.01.036.

[208] K. Zorn and N. V. Sahinidis. Computational experience with applications of bilinear cutting planes. *Industrial & Engineering Chemistry Research*, 52(22):7514–7525, 2013. URL http://doi.org/10.1021/ie3033763.

[209] K. Zorn and N. V. Sahinidis. Global optimization of general non-convex problems with intermediate bilinear substructures. *Optimization Methods and Software*, 29(3):442–462, 2014. URL http://doi.org/10.1080/10556788.2013.783032.

[210] K. Zorn and N. V. Sahinidis. Global optimization of general nonconvex problems with intermediate polynomial substructures. *Journal of Global Optimization*, 59(2-3):673–693, 2014. URL http://doi.org/10.1007/s10898-014-0190-2.